ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN MULTIPROCESSOR SYSTEMS-ON-CHIP (MPSOC), INTERCONECTADO POR UNA NETWORKS-ON-CHIP (NOC)

DANIEL GONZALO VERDEZOTO DÁVALOS

WILSON MAURICIO CHICAIZA CASA

SANGOLQUÍ – ECUADOR

2013

ESCUELA POLITÉCNICA DEL EJÉRCITO

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICADO

Ing. Vanessa Vargas
Ing. Pablo Ramos

Certificamos que el proyecto titulado “Diseño e Implementación de un Multiprocessor Systems-On-Chip Interconectado por una Networks-On-Chip”, ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la ESPE, en el Reglamento de Estudiantes de la Escuela Politécnica del Ejército.

Debido a que se trata de un trabajo de investigación recomiendan su publicación.

El mencionado trabajo consta de un documento empastado y un disco compacto el cual contiene los archivos en formato portátil de Acrobat (pdf). Autorizan a él Sr. Chicaiza Casa Wilson Mauricio y a él Sr. Verdezoto Dávalos Daniel Gonzalo que lo entreguen al Ingeniero Luis Orozco, en su calidad de Director de la Carrera.

Sangolquí, 12 de agosto del 2013

Ing. Vanessa Vargas	Ing. Pablo Ramos
DIRECTOR	CODIRECTOR

ESCUELA POLITÉCNICA DEL EJÉRCITO

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

DECLARACIÓN DE RESPONSABILIDAD

CHICAIZA CASA WILSON MAURICIO
VERDEZOTO DÁVALOS DANIEL GONZALO

El proyecto de grado denominado “Diseño e Implementación de un Multiprocessor Systems-On-Chip Interconectado por una Networks-On-Chip”, ha sido desarrollado con base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Sangolquí, 12 de agosto del 2013

Chicaiza Casa Wilson Mauricio	Verdezoto Dávalos Daniel Gonzalo

ESCUELA POLITÉCNICA DEL EJÉRCITO

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

Nosotros, Chicaiza Casa Wilson Mauricio y Verdezoto Dávalos Daniel Gonzalo

Autorizamos a la Escuela Politécnica del Ejército la publicación, en la biblioteca virtual de la Institución del proyecto “Diseño e Implementación de un Multiprocessor Systems-On-Chip Interconectado por una Networks-On-Chip”, cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y autoría.

Sangolquí, 12 de agosto del 2013

Chicaiza Casa Wilson Mauricio	Verdezoto Dávalos Daniel Gonzalo

DEDICATORIA

Este trabajo está dedicado a mi madre María Dolores Casa por ser para mí una valiosa persona la cual, me brinda su apoyo y confianza en todos los retos que me propongo. Por enseñarme a dar mis primeros pasos y por apoyarme en mis estudios universitarios. Por enseñarme que las decepciones y los reveses forman parte de la vida de cualquier ser humano. Es a través de las virtudes que ella me ha inculcado que desarrolle las capacidades que me permitieron terminar con éxito esta fase de mi vida.

Además, el esfuerzo de este proyecto, está dedicado a todas las mentes curiosas que utilicen este trabajo para desarrollar futuras investigaciones dentro del Departamento de Eléctrica y Electrónica de la Escuela Politécnica del Ejército.

Wilson Mauricio Chicaiza Casa

"*Todo lo que conocemos es parcial, incierto e inseguro*"
Bertrand Russell

DEDICATORIA

La concepción de este proyecto está dedicada a mi madre, pilar fundamental en mi vida. Sin ella, jamás hubiese podido conseguir lo que hasta ahora. Su tenacidad y lucha incansable han hecho de ella el gran ejemplo a seguir y destacar, no solo para mí, sino para mis hermanos y familia en general.

También dedico este proyecto a mis amigos, compañeros inseparables de cada jornada. Ellos representaron un gran esfuerzo y tesón en momentos de decline y cansancio.

A ellos este proyecto, que sin ellos, no hubiese podido ser.

Verdezoto Dávalos Daniel Gonzalo

AGRADECIMIENTO

En la culminación de un trabajo tan arduo y lleno de dificultadas me es inevitable que me asalte una emoción de orgullo. Es una emoción que en ocasiones me hace olvidar que existe una gran cantidad de personas que sin las cuales no hubiese sido posible el desarrollo y el feliz término de esta tesis. Por ello, es para mí un verdadero placer utilizar este espacio para expresar mis más sinceros agradecimientos.

En primera instancia agradezco desde lo más profundo de mi corazón a mi madre. Es gracias a su infinito amor que nunca descanso por ser cada día mejor persona. Por ella nunca me he dejado vencer por los problemas, mejor dicho siempre me ha impulsado a buscar más aunque los problemas siempre me han encontrado a mí.

Agradezco a la Ing. Vanessa Vargas por aceptarme en el desarrollo de tan importante proyecto bajo su dirección como directora de tesis y al Ing. Pablo Ramos como codirector. Gracias por su paciencia y por facilitarme los recursos necesarios para llevar a cabo todas las actividades propuestas en este proyecto.

A mi compañero en esta batalla el Sr. Daniel Verdezoto, con el cual acepté el reto de desarrollar el conocimiento plasmado en esta tesis que ahora se convierte en un legado para futuras investigaciones que se desarrollen el Departamento de Eléctrica y Electrónica de la Escuela Politécnica del Ejército.

Finalmente un sincero agradecimiento a todos mi profesores que se encargaron de mantener mi mente siempre ocupada a lo largo de toda mi vida universitaria.

Wilson Mauricio Chicaiza Casa

AGRADECIMIENTO

Este proyecto es el resultado del esfuerzo conjunto de todos los que formamos el grupo de trabajo. Por esto agradezco a nuestra directora de tesis Ing. Vanessa Vargas, a nuestro codirector Ing. Pablo Ramos, a mi compañero Wilson Chicaiza, quienes a lo largo de este tiempo han puesto a prueba sus capacidades y conocimientos en el desarrollo de esta tesis, la cual ha finalizado llenando todas nuestras expectativas.

A mi familia quienes a lo largo de toda mi vida han apoyado y motivado mi formación académica, creyeron en mí en todo momento y no dudaron de mis habilidades. A mis profesores a quienes les debo gran parte de mis conocimientos, gracias a su paciencia y enseñanza y finalmente un eterno agradecimiento a esta prestigiosa universidad la cual abre sus puertas a jóvenes como nosotros, preparándonos para un futuro competitivo y formándonos como personas de bien.

Verdezoto Dávalos Daniel Gonzalo

ÍNDICE GENERAL

RESUMEN

En la actualidad la tecnología de integración ha llegado al desarrollo de chips en escalas nanométricas. La tendencia de integrar un sistema completo dentro de un circuito integrado o chip denominado Systems-On-Chip (SoC) es muy común en dispositivos embebidos tales como los robots industriales, sistemas de control de freno, controles de vuelo, equipos médicos, etc.

La alta complejidad computacional requerida para el desarrollo de dispositivos embebidos involucra a los sistemas multiprocesados para mejorar su desempeño. Al aumentar el número de módulos internos dentro de un circuito integrado, se ven comprometidas las comunicaciones entre dispositivos tanto en su ancho de banda como en la dependencia de sincronismo, así como la disipación de potencia. Esta realidad obliga a buscar nuevas técnicas para maximizar la eficiencia de las comunicaciones reemplazando los buses tradicionales por la conmutación de paquetes con el fin de aumentar la fiabilidad de las comunicaciones, disminuir retardos y aumentar la tasa de transferencia de datos.

Es así que se establece un nuevo paradigma, las Networks-On-Chip (NoC) para intercambiar información entre módulos SoC que establecen una red como un subsistema de transporte público para el tráfico de información. De esta manera la información pasa por varios enlaces, de acuerdo a las decisiones que se tomen dentro del algoritmo de la NoC.

El empleo de buses dentro de un chip se ha convertido en una opción nada deseable, especialmente por la complejidad cada vez más creciente de los modernos Multiprocessor-Systems-On-Chip (MPSoC). Los diversos módulos de un SoC que emplea NoC, tales como procesadores, memorias y otros IP-Cores (bloques de propiedad intelectual), intercambian datos mediante una red, como subsistemas de transporte público para el tráfico de información.

CAPÍTULO 1

DISEÑO E IMPLEMENTACIÓN DE UN MULTIPROCESSOR SYSTEMS-ON-CHIP INTERCONECTADO POR UNA NETWORKS-ON-CHIP

En la actualidad, el desarrollo e innovación de circuitos integrados dentro de un mismo encapsulado está creciendo rápidamente gracias a la utilización de nuevas técnicas y herramientas de diseño e implementación. Este hecho ha permitido disminuir la brecha existente entre la capacidad de integración de transistores dentro de un chip y la capacidad de diseño de los mismos. La tendencia actual exige que los diseñadores de dispositivos electrónicos modernos implementen sistemas completos dentro de un circuito integrado o chip, denominados *Systems on Chip* (SoC) con lo que aparecen nuevos desafíos como disipación de potencia, comunicaciones, portabilidad, time to market, etc. (Rajesh, 2008)

La integración de sistemas completos dentro de un chip surge como respuesta a la necesidad de incrementar la confiabilidad de los sistemas y disminuir los costos de fabricación. Por otra parte, la tecnología de fabricación de circuitos integrados ha permitido incrementar enormemente el número de transistores disponibles por chip dando como resultado un aumento en la complejidad de los circuitos y sistemas electrónicos. Actualmente la CPU y la mayoría de circuitos de soporte para un sistema completo se integran eficazmente en un solo chip. Uno

de los primeros ejemplos de SoC es el circuito integrado 80186 de Intel quien incluye un circuito estándar 8086 como CPU (Ver Figura. 1.1).

Figura. 1.1. Microprocesador 80186 de Intel

Generalmente, en el diseño de los SoC se incluye un procesador programable de arquitectura RISC (Reduced Instruction Set Computer) y un DSP (Digital Signal Processor). Consta también de una estructura de comunicación interna mediante buses de conexión para procesadores, periféricos y memorias externas. Además, como interfaz con el mundo exterior, los SoC incluyen varios bloques para procesamiento de periféricos y convertidores digital/análogo.

Tradicionalmente, los SoC se han implementado en ASIC (Application Specific Integrated Circuit), sin embargo en estos últimos años los SoC han empezado a desarrollarse sobre FPGA (Field Programmable Gate Array). Los FPGA son dispositivos lógicos que tienen un arreglo bidimensional de celdas lógicas genéricas e interconexiones programables. Una celda lógica puede configurarse para cumplir una determinada función y las interconexiones pueden configurarse para interconectar las celdas.

Físicamente un FPGA es un tipo de ASIC programable, lo que le otorga flexibilidad ya que las celdas lógicas pueden ser dispuestas según las necesidades del diseñador. Los ASIC poseen una única programación y son más costosos.

Para facilitar el diseño de un SoC, se interconectan varias arquitecturas ya desarrolladas, concebidas como métodos para la reutilización de hardware en bibliotecas estándar o bloques en lenguaje HDL, también conocidos como Intellectual Property Core (IP-Core), según sea necesario para el desarrollo del sistema de aplicación específica. (Martínez)

En la búsqueda de satisfacer los requerimientos del mercado actual, se inició utilizando sistemas embebidos basados en SoC que incorporan un solo procesador o chip maestro. A fin de solventar una aplicación de alta complejidad computacional, se utilizaron varios SoCs que debían compartir el control y procesamiento general de la aplicación, creándose un problema de paralelismo, desperdicio de transistores, consumo de potencia y limitación física. La Figura. 1.2 expone la evolución del SoC hacia los Multiprocessor Systems on Chip (MPSoC), en donde se incrementan los retos ya conocidos como la reducción de tiempo de desarrollo (time to market), disipación de potencia y en especial las comunicaciones dentro de una chip, generando así nuevos sistemas digitales que son cada vez más complejos.

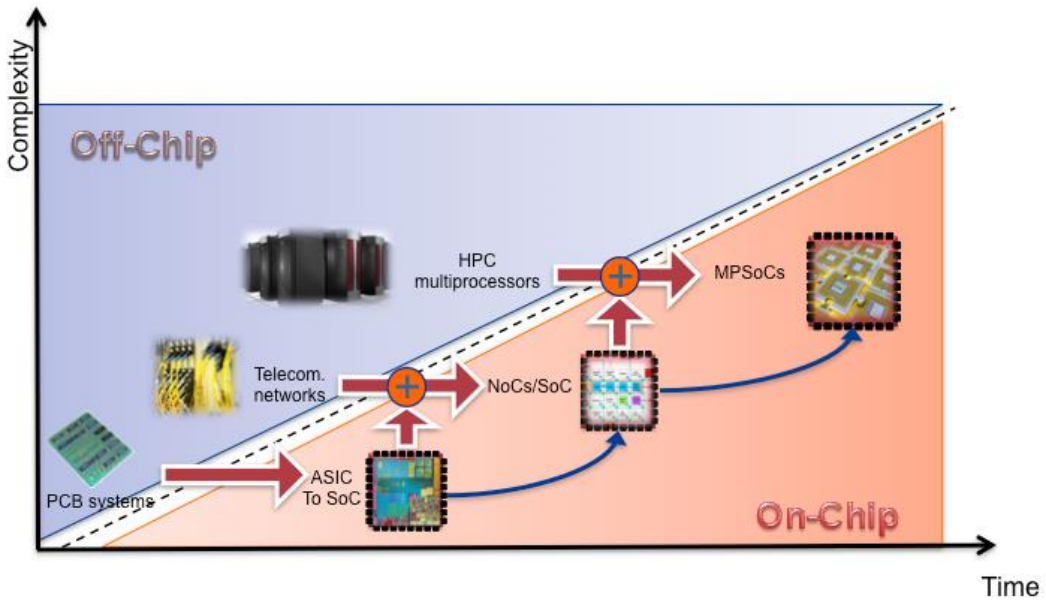


Figura. 1.2. Evolución del desarrollo de SoC y aparición de los MPSoC

El MPSoC es un SoC compuesto por múltiples procesadores, memorias y circuitería especializada, interconectados a través de una infraestructura de comunicaciones. Tradicionalmente, los MPSoC se han comunicado a través de arquitecturas basadas en buses, sin embargo, la convergencia de aplicaciones demanda la mezcla de varios tipos de tráfico en el mismo SoC. Es así, que al compartir el medio de transporte, se requiere manejar el concepto de Quality of Service (QoS). Por tanto, las arquitecturas modernas exigen una estructura flexible, reconfigurable y reprogramable. Esta exigencia ha dado paso al nacimiento de un nuevo paradigma de interconexión, llamado Networks-On-Chip (NoC).

La arquitectura de interconexión NoC se ha convertido en una solución prometedora para la comunicación on chip. Su puesta en marcha proporciona diversas razones para enfocarnos en su desarrollo tales como: eficiencia energética, fiabilidad, escalabilidad de ancho de banda y reutilización. Las NoC emplean teorías y métodos de comunicación por red dentro del chip, introduciendo notables mejoras. Los módulos MPSoC intercambian información entre sí gracias a la intervención de las NoC que establecen una red como un subsistema de transporte público para el tráfico de información. De esta forma la información pasa por varios enlaces, de acuerdo a las decisiones que se tomen dentro del algoritmo de la NoC.

En la Figura. 1.3 se muestra una arquitectura NoC compuesta de elementos de conmutación (Network Elements), que transportan la información de un lugar a otro, e interfaces de red (Network Inteface), que interactúan con los IP-Cores. Por tanto las NoC emergen como una solución de interconexión para la realización de productos flexibles que serán reconfigurables y reprogramables, ofreciendo beneficios tales como:

- Compartir conexiones, lo que incrementaría el uso de NoC.
- Ser confiables y eficientes, desde el punto de vista energético.
- Ser escalables, a diferencia de los buses tradicionales.

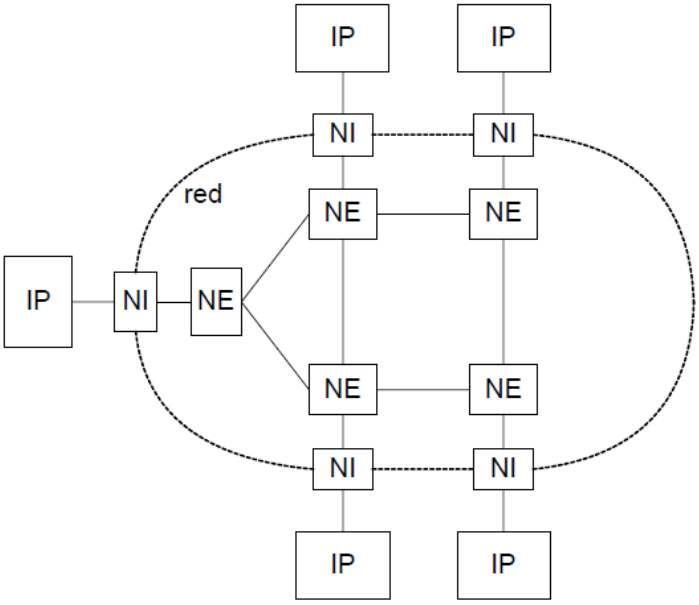


Figura. 1.3. Arquitectura NoC

La escalabilidad, eficiencia energética y reutilización, eleva la probabilidad de que se empleen NoC en la interconexión de los diferentes componentes de los sistemas MPSoC, que serán cada vez más robustos y complejos.

CAPÍTULO 2

FUNDAMENTOS TEÓRICOS

2.1. SYSTEMS ON CHIP (SoC)

Un System on Chip (SoC), es un circuito que integra todos los componentes de un ordenador u otro sistema electrónico en un solo circuito integrado o chip (ver Figura. 2.1). Un SoC puede contener circuitos digitales, analógicos y de señal mixta, todo sobre una misma oblea de silicio. Una aplicación típica de este tipo de circuitos, constituyen los sistemas embebidos.

Figura. 2.1. Estructura de un SoC

Una de las ventajas en el desarrollo de un SoC es el uso de un conjunto de bloques de propiedad intelectual o IP-Cores, los mismos que son capaces de soportar una integración del tipo plug-and-play y permiten adaptarse a las necesidades de una aplicación específica.

2.1.1. IP-Core

Definición

Es un bloque funcional completo, pre diseñado y pre configurado, que forma una unidad lógica reutilizable la cual puede ser desarrollada por una o varias personas y puede ser adquirida de forma gratuita o pagada. El acrónimo IP-Cores significa bloque de propiedad intelectual. (Baena, 2010)

Los IP-Cores comenzaron a ser empleados desde el año 1990 y han tenido un gran impacto en el diseño de un SoC, inicialmente concebidos como método para la reutilización de hardware y software, su enfoque prioritario es la reducción del time-to-market, eliminación de riesgos en la etapa de diseño y reducción de costos de desarrollo. Además de mantener una dependencia tecnológica variable, un IP-Core ha sido creado con el uso de estándares, altamente independiente del entorno de desarrollo y pre-diseñado a diferentes niveles de implementación.

2.1.2. Tipos de IP-Cores

- **Hard Cores**

En este tipo de IP-Core se otorga el layout de una tecnología en particular, con ello se procede a generar la máscara del componente, y la documentación asociada de su interfaz, funcionalidad, etc. Este tipo de IP-Core se caracteriza por ser muy poco flexible pero son sistemas muy completos y fiables en implementación. (Baena, 2010)

- **Soft Cores**

En un Soft Core se adquiere el código HDL, por tanto este tipo de IP-Core es una alternativa más flexible que la anterior, puede ser personalizado dependiendo de una aplicación específica. La mayoría de Soft Cores están escritos en Lenguaje de Descripción de Hardware (VHDL) o Verilog. (Baena, 2010)

- **Firm Cores**

En el Firm Core se otorga una descripción detallada del componente o Netlist, posee una flexibilidad tecnológica limitada pero buena predictibilidad en implementación, ya que el diseñador no puede acceder al código fuente, pues este está cifrado, únicamente se lo conecta según las especificaciones de la Netlist a nivel de compuertas durante el diseño del chip. (Pedram, Zainalabedin, & Lombardi)

2.1.3. Comunicación en un SoC

La implementación de buses para la interconexión de IP-Cores dentro un SoC es la arquitectura más simple y ampliamente utilizada. Los componentes de un SoC tienen gran relación con el mundo exterior mediante un conjunto de pines responsables de enviar y recibir información mediante interfaces de comunicación llamadas On-Chip-Bus, cuya funcionalidad es de importancia para la integración del Soc. (Milica & Stojcev, 2006)

On-Chip-Bus (OCB)

A un On-Chip-Bus se lo considera como una colección de señales o cables para que uno o más IP-Cores estén conectados y puedan comunicar los datos con los demás, tomando como principales características los parámetros de latencia y ancho de banda. Un bus típico está constituido por tres tipos de señales:

- Señal de Direccionamiento
- Señal de Datos
- Señal de Control

La señal de direccionamiento es la encargada de apuntar a la dirección de destino para que la transferencia se inicie; la señal de datos transporta la información entre la fuente y los componentes de destino tomando en cuenta la cantidad de ancho de banda a utilizar; y la señal de control registra peticiones de comunicación y acuses de recibo entre los distintos IP-Cores. (Pasricha & Dutt, 2008)

2.1.4. Topologías

Topologías de bus Compartido

Como se muestra en la Figura. 2.2, en este tipo de topología varios maestros y esclavos pueden conectarse a un bus compartido. Periódicamente se controlan las solicitudes acumuladas por los múltiples maestros y se da acceso a un maestro mediante mecanismos de arbitraje, así se evita sobrecargar al bus y limitar su ancho de banda. Las ventajas de esta topología son su fácil construcción e implementación, la extensibilidad para acoplar varios maestros y esclavos a un mismo bus y su bajo costo. Mientras que sus desventajas se centran en el retardo para la transferencia de datos, alto consumo de energía y mínimo ancho de banda. (Milica & Stojcev, 2006)

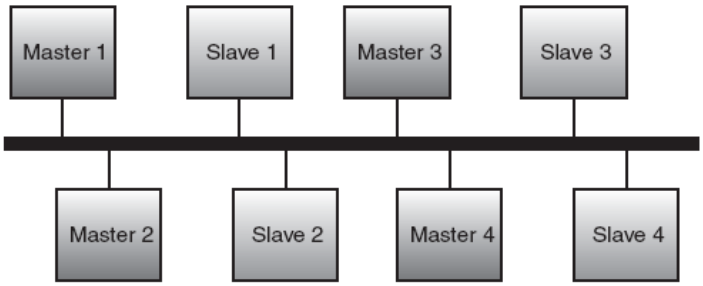


Figura. 2.2. Topología de Bus Compartido

Topología de Bus Jerárquico

Como se muestra en la Figura. 2.3, la topología de Bus Jerárquico se compone de varios buses compartidos interconectados mediante un puente o bridge para formar una jerarquía. Los buses jerárquicos ofrecen grandes mejoras de rendimiento y throughput debido a la disminución de la carga en el bus y la

segmentación de comunicaciones que permite potencializar las transacciones paralelas. (Milica & Stojcev, 2006)

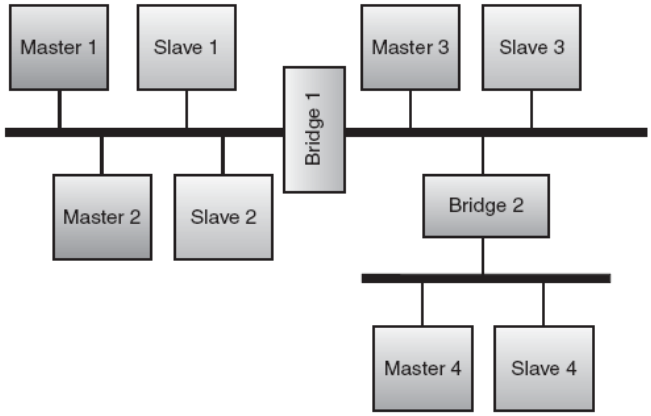


Figura. 2.3. Topología de Bus Jerárquico

Topología de Anillo

En la configuración por anillo, cada componente del nodo se comunica mediante una interfaz de llamada, por lo general esta topología se implementa mediante un protocolo tokenpass que ayuda a la comunicación entre maestros y esclavos (Ver Figura. 2.4). (Pasricha & Dutt, 2008)

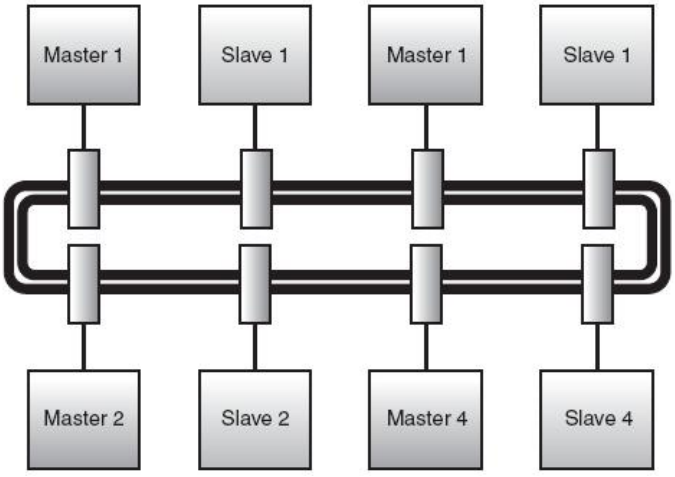


Figura. 2.4. Topología de Anillo

2.1.5. Arquitectura de buses

AMBA (Arquitectura Avanzada de Bus para Microcontroladores)

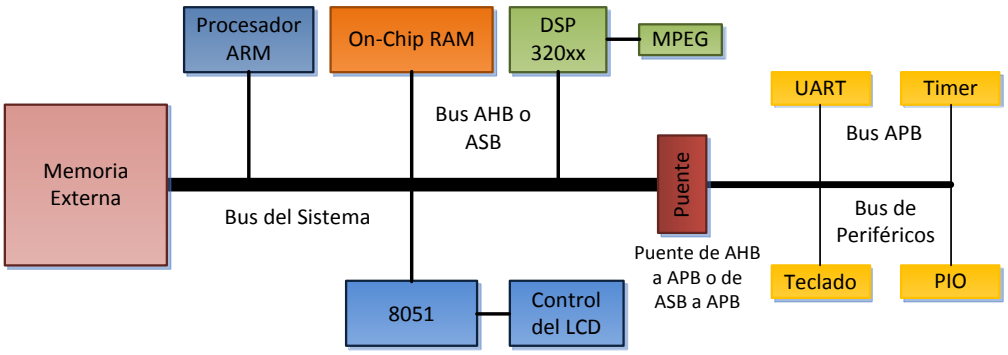


Figura. 2.5. Arquitectura AMBA

AMBA es un bus estándar creado por ARM Holdings plc con la finalidad de comunicar procesadores ARM (*Advanced RISC Machine*) en un chip. Su estructura está definida por dos segmentos: el bus del sistema y el bus de periféricos, enlazados por medio de un bridge o puente que almacena los datos y las operaciones de cada segmento (AMBA Specification, 1999). AMBA no define un método de arbitraje en la comunicación, en su lugar, permite que el arbitraje sea diseñado para ajustarse a las necesidades de las aplicaciones. Tres buses principales conforman AMBA:

- ASB (Advanced System Bus)
- AHB (Advanced High-Performance Bus)
- APB (Advanced Peripheral Bus)

- **ASB (Advanced System Bus)**

Es la primera generación del sistema de bus AMBA y soporta gran cantidad de procesadores y Microcontroladores ARM. Es relativamente simple de

implementar, el tamaño del bus de datos varia de 32 a 256 bits, el tamaño del bus de direccionamiento es de 32 bits y utiliza los dos flancos de reloj para su funcionamiento. (Milica & Stojcev, 2006) (Hessabi)

- **AHB (Advanced High-Performance Bus)**

AHB es un bus desarrollado para diseños sintetizables de alto rendimiento. Proporciona funciones de transferencia de información del tipo ráfaga y del tipo transacciones divididas. Constituye el canal de comunicaciones entre el procesador integrado, los periféricos, aceleradores de hardware y el bridge APB. El tamaño del bus de datos varia de 32 a 256 bits, el tamaño del bus de direccionamiento es de 32 bits y utiliza únicamente el flanco ascendente de reloj para su funcionamiento. (Milica & Stojcev, 2006) (Hessabi)

- **APB (Advanced Peripheral Bus)**

Es un bus desarrollado para conectar los dispositivos periféricos mediante una interfaz de complejidad reducida y optimizado para el consumo mínimo de energía. Su arquitectura comprende un maestro que es el bridge y varios esclavos como el Timer o el UART (Transmisor Receptor Asíncrono Universal). El tamaño del bus de datos varia de 8 a 32 bits y el tamaño del bus de direccionamiento es de 32 bits. (Milica & Stojcev, 2006) (Hessabi)

Altera Avalon Bus

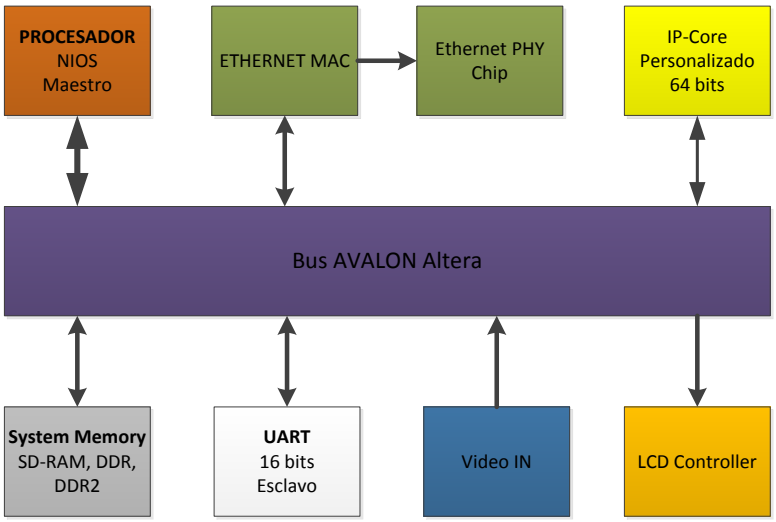


Figura. 2.6. Arquitectura de Altera Avalon

Avalon es una arquitectura de bus implementada para conectar un procesador Maestro Nios y periféricos en un chip. Es utilizado principalmente para el diseño de SoCs sobre FPGA de la marca Altera, mediante direccionamiento multiplexado de datos y decodificación de direcciones. (Milica & Stojcev, 2006)

Avalon posee un conjunto de señales predefinidas con las que el usuario puede interconectar los distintos IP-Cores mediante una interfaz síncrona y únicamente configurando los puertos de conectividad entre el maestro y los dispositivos esclavos.

Avalon posee un ancho de banda comprendido entre 8 y 128 bits para la transferencia de información. La arquitectura Altera Avalon Bus utiliza separadamente las líneas de datos, control y direccionamiento. (Hessabi)

CoreConnect Bus

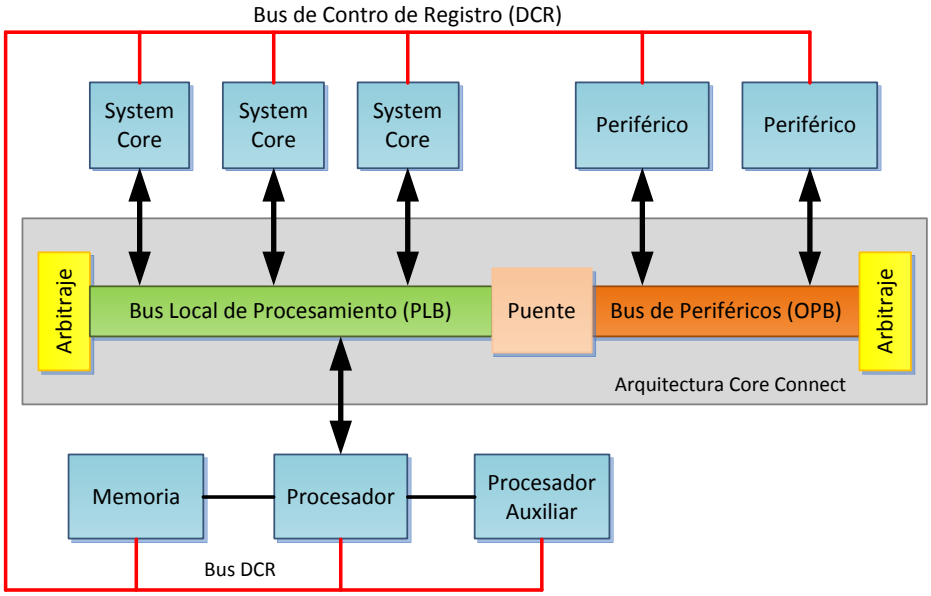


Figura. 2.7. Arquitectura CoreConnect

CoreConnect es una arquitectura desarrollada por IBM que facilita la integración y reutilización del procesador, el sistema y los distintos periféricos. Esta arquitectura está organizada jerárquicamente y se compone por tres buses que proporcionan una eficiente interconexión entre los núcleos y las bibliotecas de memoria dentro del SoC. Estos buses son:

- PLB (Processor Local Bus)
- OPB (On-chip Peripheral Bus)
- DCR (Device Control Register Bus)

- **PLB (Processor Local Bus)**

PLB es un bus síncrono de alto rendimiento que comparte direcciones y posee sus buses de lectura y escritura separados. El tamaño del bus de datos varia de 16 a 128 bits, el tamaño del bus de direccionamiento es de 32 bits, pudiéndose conectar hasta 16 maestros y un infinito número de esclavos. PLB posee una

central de arbitraje que permite la reducción de latencia en la comunicación dentro del chip. (Pasricha & Dutt, 2008)

- **OPB (On-chip Peripheral Bus)**

OPB es un bus síncrono optimizado para conectar dispositivos de bajo rendimiento y reducir la carga capacitiva en el bus PLB, comparte direcciones y posee varios buses de datos. El tamaño del bus de lectura varia de 32 a 64 bits, el tamaño del bus de direccionamiento es de 64 bits. OPB posee varios maestros y su comunicación con los esclavos la realiza en un solo ciclo de reloj mediante un bus de arbitraje. Los dispositivos maestros del bus PLB pueden acceder a los periféricos mediante el bridge OPB, en donde el bridge OPB actúa como esclavo en el bus PLB y como maestro en el bus OPB. (Milica & Stojcev, 2006) (Pasricha & Dutt, 2008)

- **DCR (Device Control Register bus)**

DCR es un bus síncrono de baja velocidad que transmite información de configuración por medio de un solo maestro desde el núcleo del procesador hasta los diversos componentes del SoC tales como memoria del chip, procesadores auxiliares y periféricos.

Su arquitectura está basada en la topología por anillo, convirtiéndose en un distribuidor de información entre los distintos dispositivos del SoC a través de tareas de multiplexado, tratándose de un bus de direcciones de 10 bits y un bus de 32 bits para datos. (Milica & Stojcev, 2006)

Wishbone

Wishbone es un bus desarrollado por Silicore Corporation y lanzado al dominio público por OpenCore en agosto de 2002 con la finalidad de mejorar la portabilidad y fiabilidad del sistema mediante una interfaz común entre los distintos IP-Cores. (Hessabi)

La principal característica de Wishbone es que no está protegido, pudiendo ser copiado, configurado y distribuido libremente; su topología interna define dos interfaces, un maestro que es capaz de iniciar los ciclos del bus y un esclavo capaz de ejecutar dichos ciclos emitidos por el maestro. Su implementación admite distintos tipos de topologías de interconexión:

- Conexiones Punto a Punto
- Flujo de Datos
- Bus Compartido
- Switch de Interconexiones

- **Conexiones Punto a Punto**

La topología de conexiones punto a punto es la metodología de interconexión más sencilla entre dos IP-Cores, siendo solo un maestro y el otro esclavo, en donde se fija una tasa de transferencia mediante el protocolo Handshake. (Wishbone System-On-Chip (SoC) Interconnection Architecture for Portable IP-Cores, 2010)(Ver Figura. 2.8)

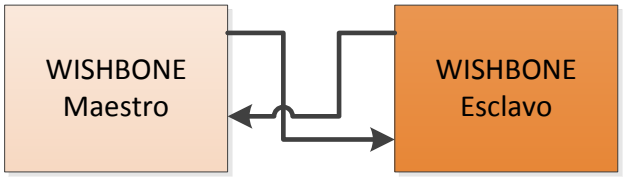


Figura. 2.8. Interconexión Punto a Punto

- **Flujo de Datos**

La topología de Flujo de Datos es empleada cuando la información se procesa de forma secuencial, donde cada IP-Core posee su propia interfaz de comunicación. El paso de datos entre IP-Cores está definido por un maestro y un esclavo. La topología de Flujo de Datos basa su comunicación en el paralelismo, lo cual acelera el tiempo de ejecución de una instrucción. (Wishbone System-On-Chip (SoC) Interconnection Architecture for Portable IP-Cores, 2010) (Ver Figura. 2.9)

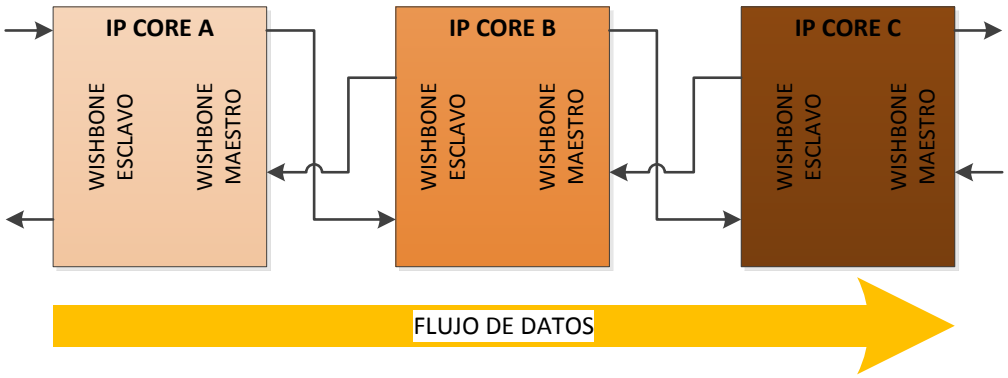


Figura. 2.9. Interconexión Flujo de Datos

- **Bus Compartido**

La topología de bus compartido es utilizada para interconectar dos o más maestros con dos o más esclavos, implementado principalmente en buses PCI

(Peripheral Component Interconnect). Un maestro inicia la comunicación a un nuevo esclavo y un árbitro determina cuando un maestro puede tener acceso al bus compartido. En la topología de bus compartido el árbitro es definido por el usuario que configurará la comunicación. (Wishbone System-On-Chip (SoC) Interconnection Architecture for Portable IP-Cores, 2010) (Ver Figura. 2.10)

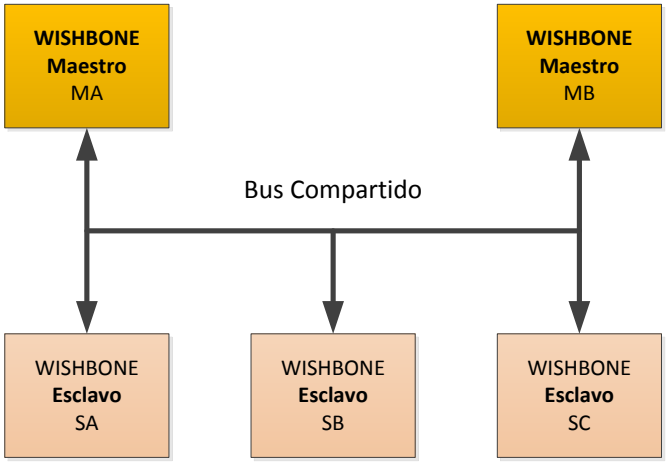


Figura. 2.10. Interconexión Bus Compartido

- **Switch de Interconexiones**

La topología Switch de interconexiones es utilizada en Multiprocessor Systems on Chip (MPSoC) cuando dos o más maestros se interconectan juntos para acceder cada uno de ellos a uno o más esclavos. Cada maestro inicia un ciclo de bus direccionable a un nuevo esclavo y un árbitro determina que cada maestro tenga acceso a los distintos esclavos. De esta manera cada maestro arbitra un canal del Switch al establecer la comunicación. En la topología Switch de Interconexiones los datos se transfieren punto a punto. (Interfaz WISHBONE, 2010) (Ver Figura. 2.11)

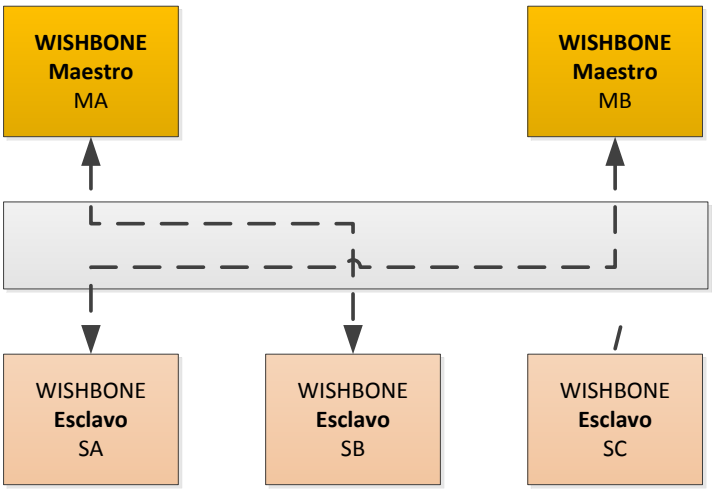


Figura. 2.11. Switch de Interconexiones

2.2. MULTIPROCESSORS SYSTEMS ON CHIP (MPSOC)

La evolución de los sistemas embebidos lleva al diseño de sistemas cada vez más complejos. Los nuevos sistemas embebidos constan de múltiples procesadores interconectados con otros subsistemas de hardware que se integran habitualmente en un mismo circuito integrado o Multiprocessor System on Chip (MPSoC). Como consecuencia del aumento de las capacidades de la plataforma de hardware, los sistemas embebidos permiten soportar mayores funcionalidades, aunque el software que incluyen estas arquitecturas, también se vuelva más complejo. (Nollet, 2008)

Un MPSoC incorpora varios elementos de procesamiento necesarios para una aplicación que utiliza multiprocesadores programables como parte de los elementos de un sistema, unidos por una estructura de interconexión como se muestra en la Figura. 2.12, ajustándose a las necesidades y evolución de la arquitectura de computadores, tales como: operación en tiempo real, consumo de baja potencia y aplicaciones multitarea.

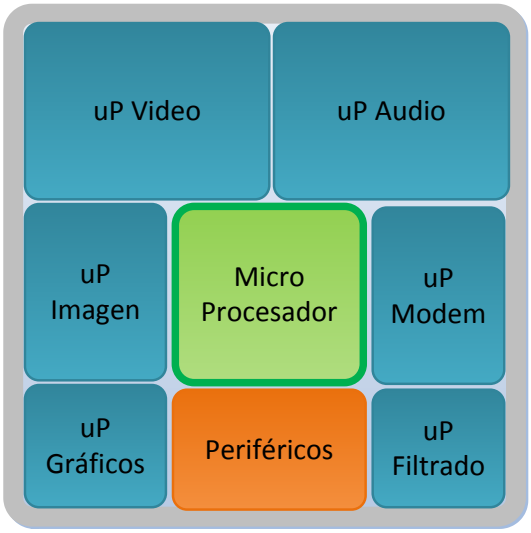


Figura. 2.12. Estructura de un MPSoC

La taxonomía de los MPSoC define la formación de dos importantes familias de multiprocesadores: La familia de Multiprocesadores Homogéneos cuyo primer prototipo fue Lucent Daytona y la familia de Multiprocesadores Heterogéneos, en la que destaca el C-5 Network Processor, Nexperia y OMAP.

2.2.1. Tipos de MSPSoC

Multiprocesadores Homogéneos

Un Multiprocesador homogéneo es un conjunto de elementos de procesamiento integrados en un SoC basados en el modelo de arquitectura computacional paralela, en donde su principal característica se centra en la utilización de un solo tipo de procesadores interconectados para desempeñar funciones de propósito general. (Wolf, Amine, & Martin, Multiprocessor System-On-Chip (MPSoC) Technology, 2008)

El modelo de arquitectura computacional paralela referido en la Figura. 2.13 consiste en aumentar el número de elementos físicos mediante el trabajo concurrente de varias unidades de procesamiento de un solo tipo. La finalidad del modelo de arquitectura computacional paralela es dividir el tiempo de ejecución de cada recurso, para disminuir la frecuencia de operación y su tensión de alimentación; por lo tanto, se consigue la disminución del consumo de potencia dinámica de manera significativa.

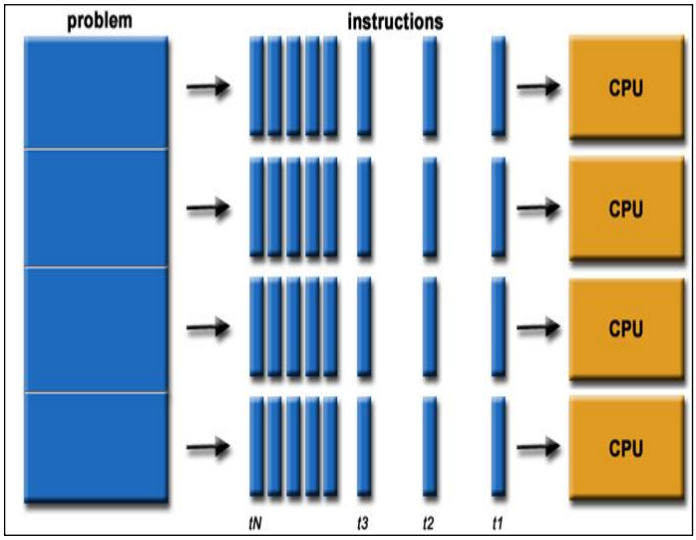


Figura. 2.13. Modelo de Arquitectura Computacional Paralela

2.2.1.1. Multiprocesadores Heterogéneos

Un Multiprocesador heterogéneo es un conjunto de procesadores interconectados con diferentes funcionalidades. Estos sistemas están formados por distintos elementos de procesamiento, uno o varios procesadores de propósito específico, procesadores de señales digitales, aceleradores de hardware, periféricos y una infraestructura de interconexión como la que se puede ver en la Figura. 2.14. (Torres, Benoit, Sassatelli, & Robert)

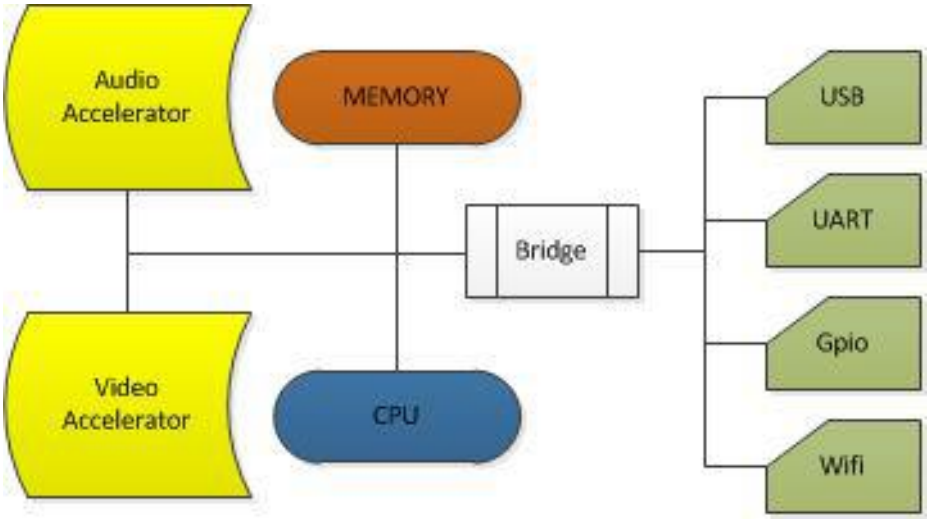


Figura. 2.14. Arquitectura de un MPSoC Heterogéneo.

Los Multiprocesadores Homogéneos son flexibles y escalables, pero menos eficientes en comparación con los sistemas heterogéneos. Debido a su desempeño, los Multiprocesadores Heterogéneos se utilizan en los sistemas portátiles y sistemas integrados, mientras que los Multiprocesadores Homogéneos se utilizan comúnmente para las consolas de videojuegos, servidores y computadores de escritorio.

2.2.2. Los primeros MPSoC

Lucent Daytona

El primer MPSoC conocido es el Lucent Daytona. El procesamiento de señales en este MPSoC se ejecuta en un número de canales de datos específico. Daytona es una arquitectura simétrica con cuatro procesadores bajo el modelo *Single Instruction Multiple Data Stream* (SIMD), conectados a un bus de alta velocidad como se aprecia en la Figura. 2.15. Cada procesador posee 8KB de memoria cache organizados en 16 bancos, en donde cada banco puede configurarse como cache de instrucción o cache de datos mediante un Set de

Instrucciones Reducidas (*Reduced Instruction Set Computer*, RISC). Los procesadores de Lucent Daytona comparten un espacio común de direcciones en la memoria. La implementación de Daytona abarca un chip de 200mm^2, puesto en marcha a una frecuencia de 100MHz y alimentado con 3.3 V en un proceso CMOS de 0.25 micras. (Wolf, Amine, & Martin, Multiprocessor System-On-Chip (MPSoC) Technology, 2008)

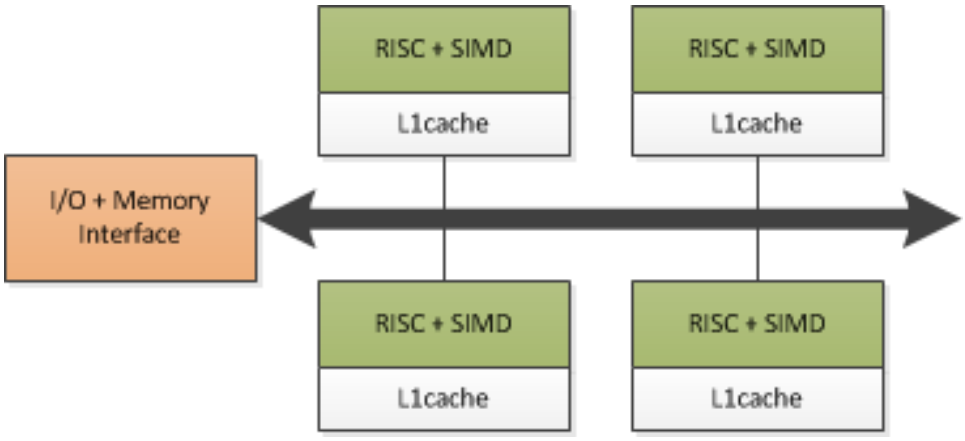


Figura. 2.15. Arquitectura del Lucent Daytona

C-5 Network Processor

El MPSoC C-5 es diseñado para el procesamiento de paquetes en una red, en donde los paquetes son manejados por los procesadores divididos en cuatro grupos de cuatro unidades cada uno y tres buses para manejar diferentes tipos de tráfico en el procesador (Wolf, Amine, & Martin, Multiprocessor System-On-Chip (MPSoC) Technology, 2008). Su frecuencia de operación oscila entre 166MHz y 233MHz, ver Figura. 2.16.

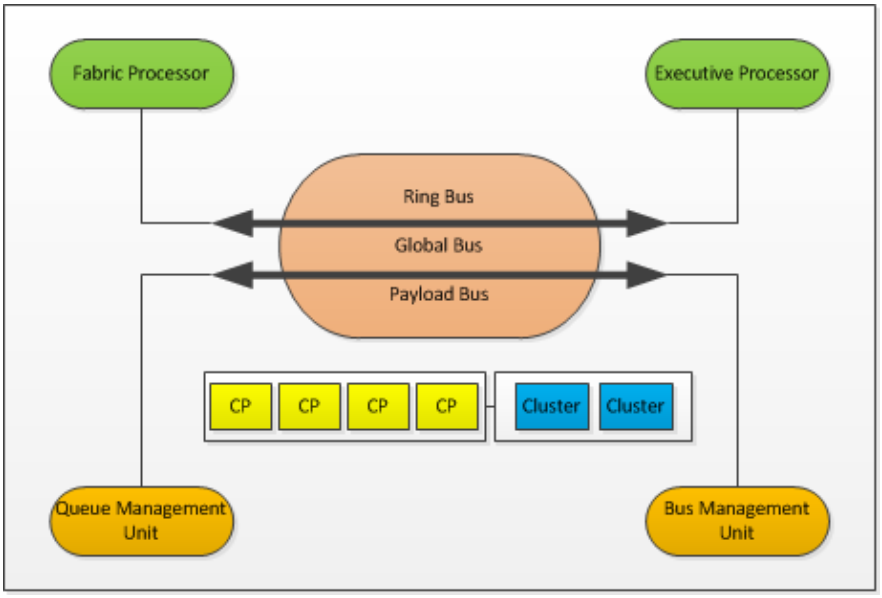


Figura. 2.16. Arquitectura del MPSoC C-5 Network Processor

Philips Viper Nexperia

Uno de los primeros MPSoC multimedia es el Philips Viper Nexperia, que incluye 2 procesadores: un *Multiprocessor without Interlocked Pipeline Stages* (MIPS PR3940) y un procesador *Very Long Instruction Word* (VLIW Trimedia TM32). El MIPS actúa como maestro ejecutando el sistema operativo, mientras que le Trimedia actúa como esclavo y ejecuta las ordenes de procesamiento de video. El sistema incluye tres buses, uno para cada CPU y otro para la interfaz de memoria externa SDRAM (Synchronous Dynamic Random-Access Memory), referidos en la Figura. 2.17. La implementación del Philips Viper Nexperia es multiplataforma, soporta algunos sistemas operativos, como Windows CE, Linux y VxWorks. (Ernst, 2003)

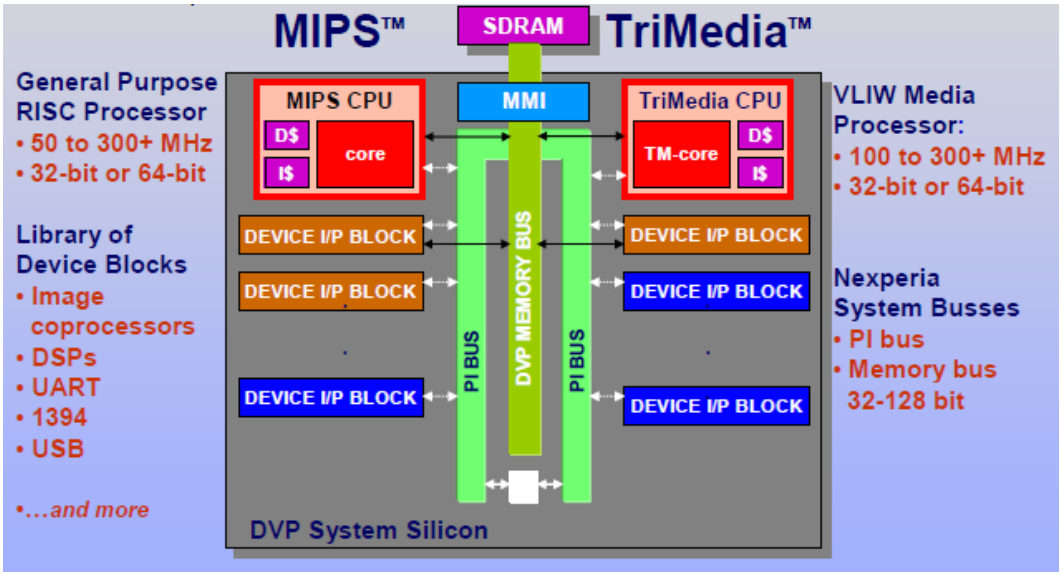


Figura. 2.17. Arquitectura de un MPSoC Philips Viper Nexpiria

OMAP 5912

Una de las aplicaciones más importantes de MPSoC es la célula de procesamiento de un teléfono celular, la cual realiza varias operaciones, incluyendo las comunicaciones y operaciones multimedia.

El MPSoC OMAP 5912 es desarrollado por Texas Instruments y dispone de 2 CPU: un *Advanced RISC Machine* (ARM9) que actúa como maestro ejecutando el sistema operativo, y un *Texas Instrument* (TMS320C55X) que actúa como esclavo para el procesamiento digital de señales (DSP), ver Figura. 2.18. OMAP5912 es diseñado para soportar aplicaciones inalámbricas 2.5G y 3G, procesamiento de voz, servicios basados en localización, seguridad, juegos y multimedia. (Wolf, High Performance Embedded Computing, 2007)

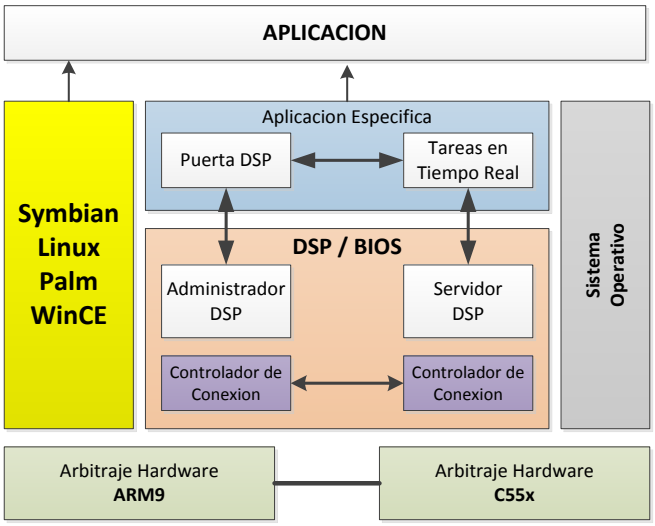


Figura. 2.18. Arquitectura de un OMAP 5912

Nomadik STMicroelectronics

Otro de los MPSoC desarrollado para aplicaciones en teléfonos celulares, es el Nomadik STMicroelectronics que utiliza un procesador ARM926EJ como maestro y posee dos aceleradores programables como esclavos: uno para audio y otro para video. Ver Figura. 2.19. Nomadik ejecuta una instrucción por ciclo de reloj y es utilizado en la gama alta de teléfonos celulares. (Wolf, Amine, & Martin, Multiprocessor System-On-Chip (MPSoC) Technology, 2008)

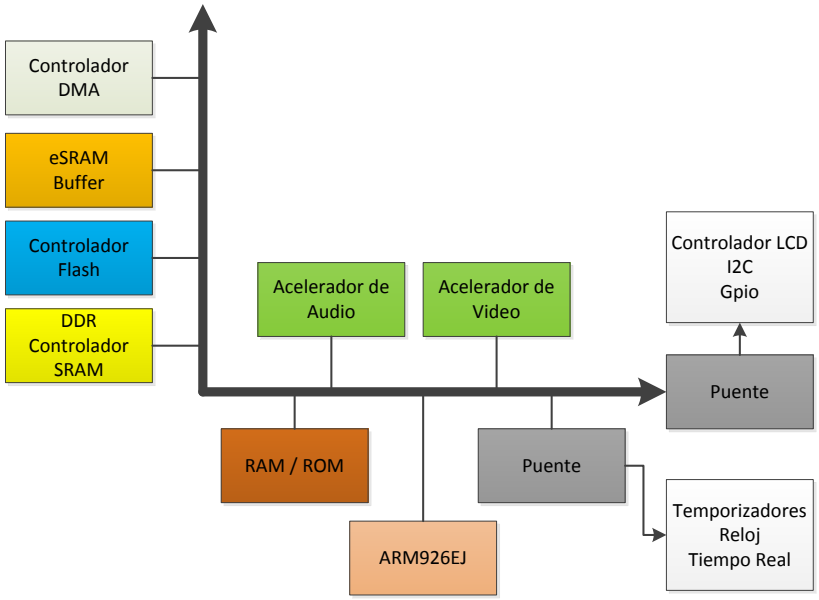


Figura. 2.19. Arquitectura del Nomadik STMicroelectronics

MPCore ARM

El MPCore ARM es un MPSoC homogéneo que también permite algunas configuraciones heterogéneas. Esta arquitectura puede acomodar hasta cuatro procesadores, en donde la memoria controladora puede ser configurada para ofrecer diferentes grados de acceso a diversas partes de la memoria para cada CPU, pudiendo, de ser el caso, que un solo procesador sea capaz de leer una parte del espacio de memoria, mientras que el resto de procesadores tienen un acceso restringido (ver Figura. 2.20). (Wolf, Amine, & Martin, Multiprocessor System-On-Chip (MPSoC) Technology, 2008)

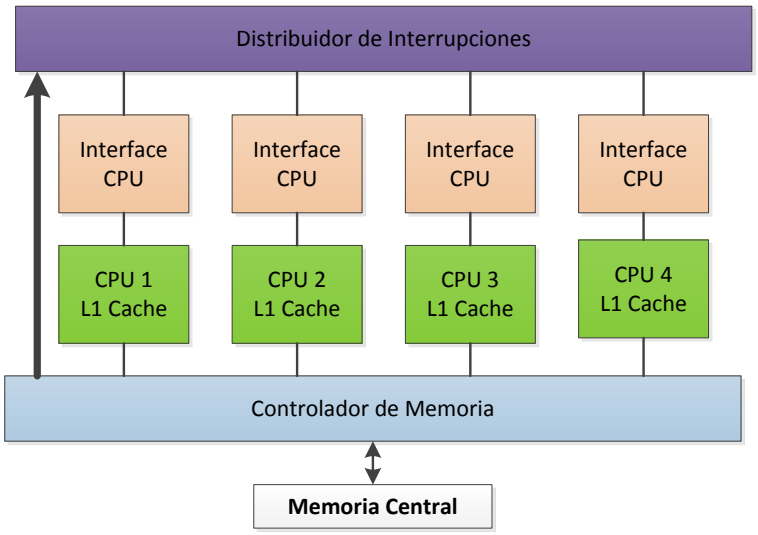


Figura. 2.20. Arquitectura del MPSoC ARM

2.2.3. Arquitectura MPSoC

La gran capacidad de implementar varios procesadores dentro de un mismo diseño permite desarrollar diversas arquitecturas dentro de los MPSoC, con la finalidad de aumentar el rendimiento de una aplicación específica.

La arquitectura más simple se puede observar en la Figura. 2.21, en la cual se incluyen varios procesadores pero cada uno se dedica a una tarea independiente y no existe comunicación entre ellos. De esta manera se llega a combinar el uso de procesadores soft-core y hard-core de acuerdo a la aplicación específica. (Huerta, 2009)

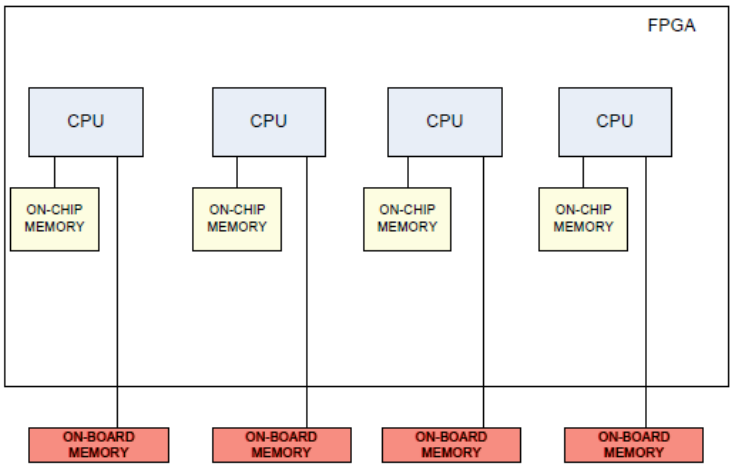


Figura. 2.21. Arquitectura MPSoC de Procesadores no interconectados

Si la aplicación demanda un procesamiento de tareas que involucra a todos los CPU del sistema, es necesario establecer algún medio de comunicación entre los procesadores, para que estos puedan intercambiar información y optimizar el proceso.

El mecanismo de comunicación entre procesadores más utilizados se puede observar en la Figura. 2.22 y hace referencia al uso de buses de propósito general.

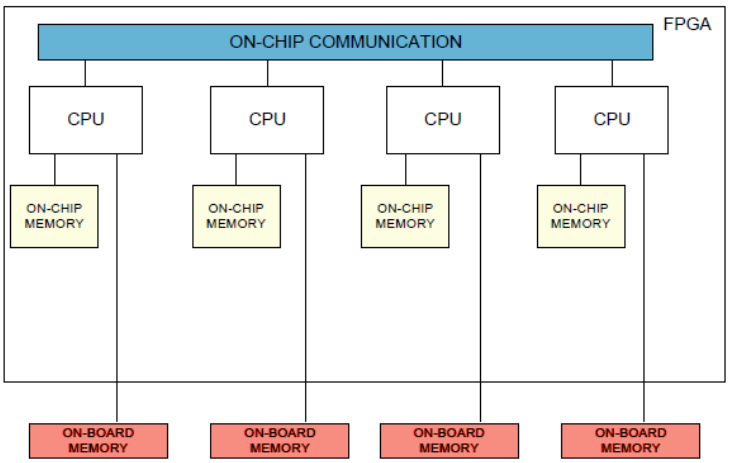


Figura. 2.22. Arquitectura MPSoC mediante Bus de proposito general

Mientras que la Figura. 2.23 hace referencia a un MPSoC con comunicación entre los procesadores a través de memoria compartida.

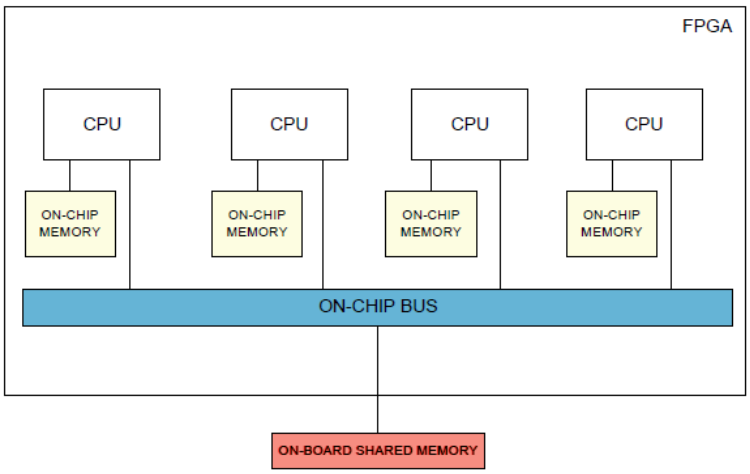


Figura. 2.23. Arquitectura MPSoC de Memoria Compartida

Para el procesamiento masivo de datos, es recomendable la implementación de una arquitectura maestro esclavo, donde todos los procesadores esclavo ejecutan el mismo código y es el procesador maestro, el encargado de recibir la información y repartirla entre los esclavos para su procesamiento. En la Figura. 2.24 se muestra una arquitectura de este tipo. (Huerta, 2009)

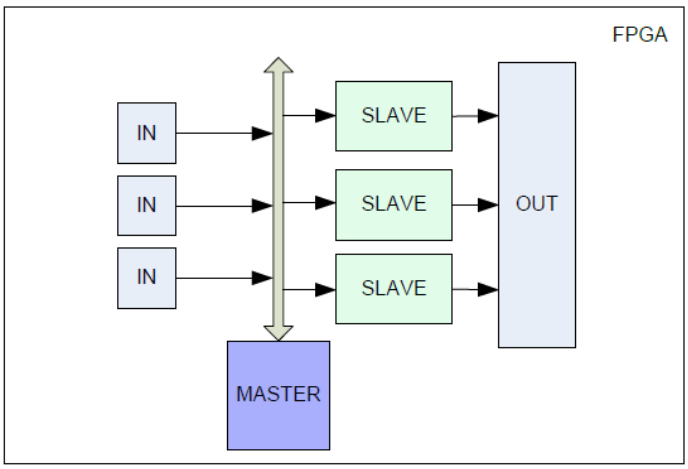


Figura. 2.24. Arquitectura MPSoC en configuracion Maestro - Esclavo

En la Figura. 2.25 se muestra otro tipo de arquitectura MPSoC, en la cual cada procesador realiza una parte de la tarea y pasa sus resultados al siguiente procesador. Esta arquitectura segmentada puede utilizarse en aplicaciones en las que no existe paralelismo intrínseco, pero se pueden dividir las tareas a realizar por los procesadores de forma independiente.

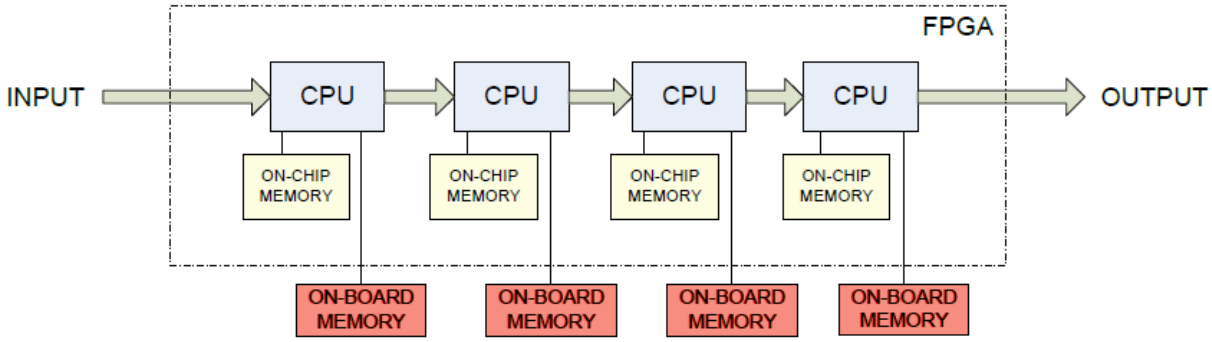


Figura. 2.25. Arquitectura MPSoC Segmentada

A partir de estas arquitecturas básicas se puede desarrollar múltiples MPSoC, combinando o modificando las arquitecturas de acuerdo a las necesidades específicas de cada aplicación.

Las posibilidades de integrar varios procesadores intercomunicados dentro de un chip son múltiples tanto en topología (tipo anillo, maestro-esclavo), como en implementación (buses compartidos, arquitectura segmentada o network on chip). Lo que hace que cada sistema emplee la solución más adecuada para la aplicación que se desea implementar. (Huerta, 2009)

2.2.4. Beneficios del Multiprocesamiento

Existen diversas razones por las cuales se necesita sistemas con múltiples procesadores. De los cuales se citan los siguientes escenarios:

Múltiples funciones independientes: Mejorando el rendimiento del procesamiento en múltiples tareas, de esta forma se asignan distintos periféricos a cada procesador para cumplir con un objetivo de manera independiente. (Asokan, 2007)

Descarga de Control: Cuando existen dos grupos de tareas, uno de tiempo real y el otro para tareas críticas. Un solo procesador puede encargarse de esto sin mayor problema, pero tener un procesador adicional trabajando como esclavo, mejora el rendimiento. De esta forma el procesador esclavo controla las señales en tiempo real y el procesador maestro regula el otro grupo de tareas. Este último administra al esclavo por medio de una comunicación periódica. (Asokan, 2007)

Descarga de Datos: Este es un escenario en donde hay la presencia de una gran cantidad de tareas. Por lo tanto un procesador esclavo es necesario para trabajar con las descargas de datos, mientras que el maestro las coordina y procesa. Algunos ejemplos de este posible escenario son: en las descargas de protocolos de red, procesamiento de señales de comunicación, algoritmos de seguridad, etc. (Asokan, 2007)

Procesamiento de Interfaces: Es común en diseños de red, con lo cual se consigue un sistema que incrementa el rendimiento de los multiprocesadores para la ejecución de una gran variedad de tareas. Las mejoras de rendimiento se logran haciendo que un procesador esclavo administre el flujo de datos hacia las interfaces múltiples mientras que un procesador maestro se encarga de las tareas de alto nivel. (Asokan, 2007)

Flujo de Procesamiento: Los procesadores actúan como uno solo al manejar el flujo de información por partes, es decir que cada procesador realiza una

determinada tarea sobre una porción de datos, antes de pasar al siguiente procesador. (Asokan, 2007)

Procesamiento Simétrico: Cuando un solo procesador no es suficiente para ejecutar una tarea, es necesario usar varios procesadores para incrementar el rendimiento sobre la misma. Por lo tanto el Multiprocesamiento Tradicional Simétrico (SMP) es una solución para incrementar el rendimiento al dividir funciones entre varios procesadores y administrar las tareas en forma paralela. (Asokan, 2007)

Confiabilidad y Redundancia: En los sistemas tolerantes a fallas, las redundancias son necesarias para obtener la máxima confiabilidad en el sistema. La confiabilidad se logra volviendo a replicar varias veces la solución, hasta obtener una respuesta tolerante a la falla de una de las réplicas. (Asokan, 2007)

2.3. NETWORK ON CHIP

El interconexionado por buses presenta una manera sencilla de comunicar los componentes de un sistema entre sí, sin embargo, la jerarquía de buses exterioriza problemas debido a su baja escalabilidad, ya que al aumentar el número de maestros conectados al bus, su rendimiento cae drásticamente. Por lo tanto, se necesita un cambio radical en la integración de los distintos módulos dentro de un chip que se ajuste a la escalabilidad de la tecnología de implementación. En este escenario se evidencia la necesidad de encontrar un nuevo paradigma de interconexión, por lo cual hace aparición las Network on Chip (NoC) tal como se expone en la Figura. 2.26.

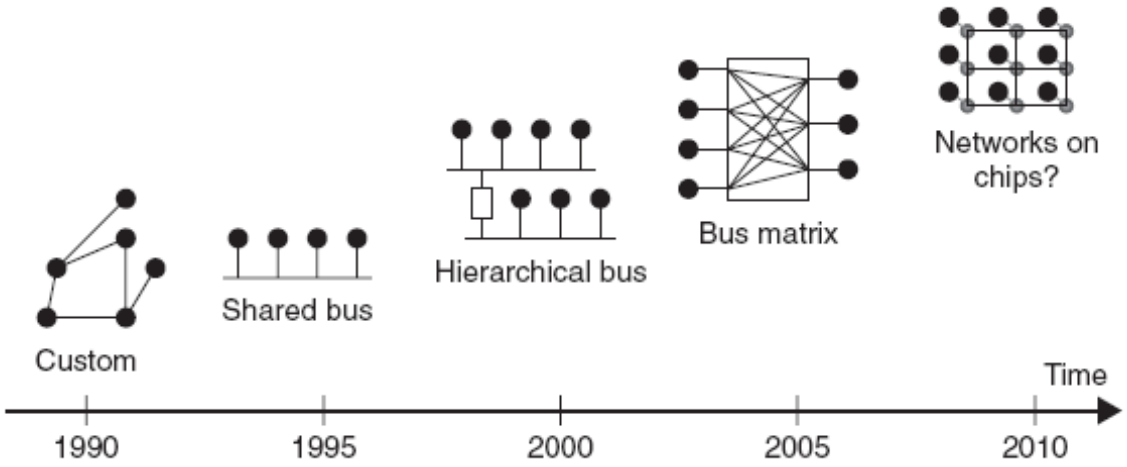


Figura. 2.26. Evolución de las Arquitecturas de Comunicación en un Chip

Una NoC es un conjunto de elementos de conmutación e interfaces de red interconectados para dar servicio de comunicación a los distintos módulos dentro de un chip. Así la NoC proporciona mecanismos de interconexión física para la transferencia de información entre IP-Cores, con la finalidad de dar soporte a aplicaciones de alto rendimiento. (Benini & De Micheli)

A continuación en la Tabla. 2.1 se muestran las ventajas y desventajas del método de interconexionado por buses comparado con el método de interconexionado NoC que está revolucionando el desarrollo de chips cada vez más potentes.

Desventajas de los Buses	Ventajas de las NoC
Problemas de Escalabilidad: Cada unidad agregada afecta al rendimiento eléctrico.	El rendimiento local no se degrada cuando se escala.
Dificultad para sincronizar el bus en un proceso *Distributed Shared Memory* (DSM)	Las decisiones de ruteo son distribuidas, así los protocolos de la red no sean centralizados.
El arbitraje del bus es de instancia específica.	El mismo router puede ser reinstanciado para todos los tamaños de la red.

Desventajas de los Buses	Ventajas de las NoC
Las simulaciones de tráfico en el bus son problemáticas y lentas, debiendo esperar el recorrido del paquete por todo el proceso. El ancho de banda es limitado y compartido con todas las unidades.	El *Built-In Self-Test* (BIST) es rápido y ofrece buena cobertura de pruebas y simulaciones Los anchos de banda son escalables según el tamaño de la red.
Ventajas de los Buses	**Desventajas de las NoC**
La latencia del bus es mínima una vez que el arbitraje concede el control. Casi todos los buses son compatibles con la mayoría de IP-Cores disponibles. Los conceptos son simples y fáciles de entender por los diseñadores.	La congestión interna de la red puede causar latencias prolongadas. Los IP-Cores orientados a buses necesitan wrappers para su implementación en NoC. Los diseñadores de sistemas necesitan re-educarse.

Tabla. 2.1. Comparación de la arquitectura de buses contra la arquitectura de red

2.3.1. Arquitectura de una NoC

Entre todas las posibles arquitecturas que pueden existir en diferentes literaturas que tratan sobre NoC, se puede distinguir la presencia de tres principales componentes que lo conforman (Atienza, et al., 2007), Ver Figura. 2.27:

- Networks Elements, NE (también llamados routers o switches)
- Networks Interface, NI (también llamados adaptadores de red)
- IP-Cores o Nodos

En un nodo o IP-Core, es el lugar en donde se ejecutan todos los elementos de procesamiento (Process Elements, PE), es decir, el nodo es quien genera, recibe y envía paquetes de datos a través de una determinada topología. Por otra parte, los NI reciben los paquetes y los direccionan a su destino por medio del canal de salida correspondiente. En tanto que, los NE son la columna vertebral de la red cuya función es la de encaminar los paquetes desde su fuente hacia su destino. (Mirza-Aghatabar, Koohi, Hessabi, & Pedram)

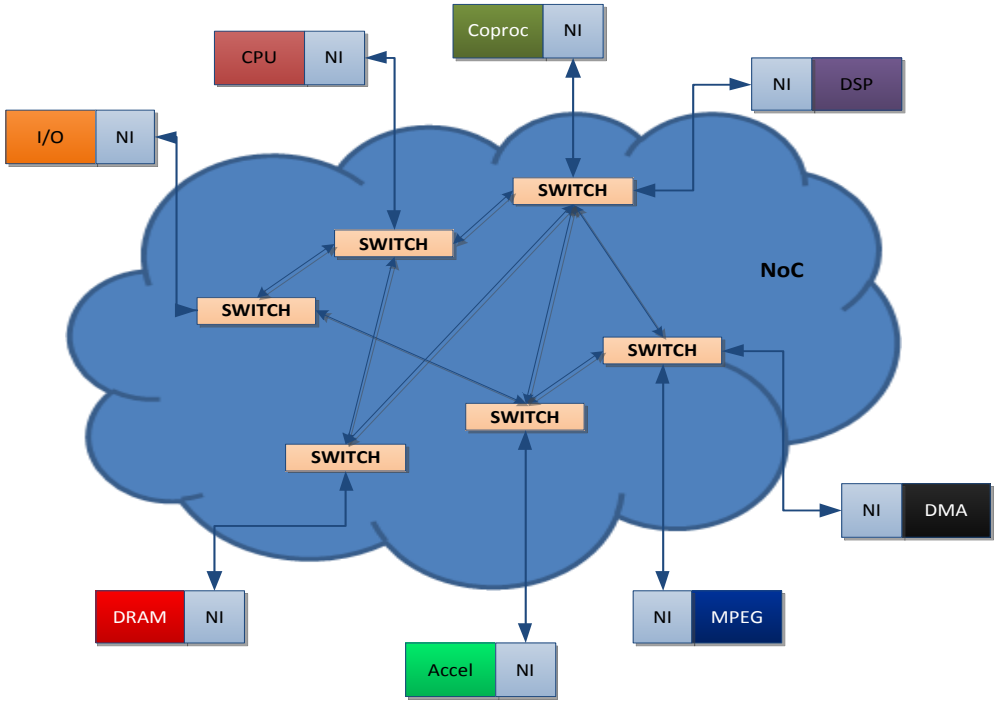


Figura. 2.27. Elementos que conforman la arquitectura de una NoC

2.3.2. Topologías

Existen varias topologías para la implementación de una arquitectura NoC, la cual se escoge dependiendo de la estrategia de enrutamiento y el método de control de flujo usado, pues ambos están estrechamente relacionados con la infraestructura de conmutación basada en NoC. (Rivero, 2005)

Las topologías se dividen en regulares e irregulares, en la Figura. 2.28 se presentan las topologías regulares las cuales son las más utilizadas en el diseño de una arquitectura NoC. (Agarwal, Iskander, & Shankar, 2009)

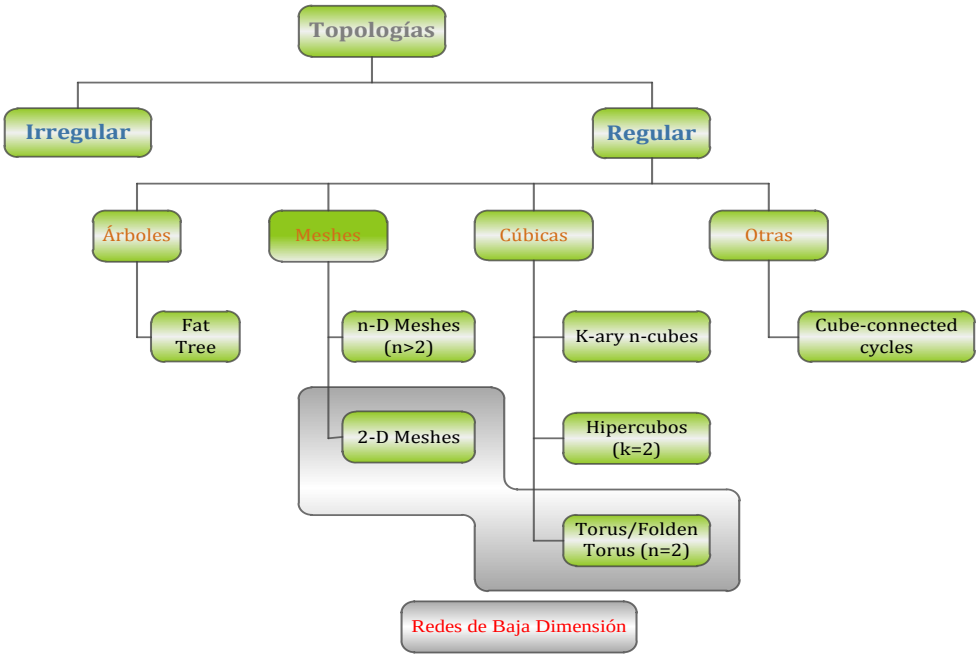


Figura. 2.28. Tipos de Topologías

De las topologías mostradas en la Figura. 2.28, se describen a continuación algunas de ellas, las cuales son:

- 2D-Mesh
- 2D-Torus
- Honeycomb
- Fat-tree

2D-Mesh

2D-Mesh es la topología más simple de una red en la que cada nodo se encuentra conectado a un elemento de conmutación, el cual está dispuesto dentro

de una matriz regular en *n* filas y *n* columnas. En esta topología cada elemento de red se conecta con su respectivo vecino tal y como se puede apreciar en la Figura. 2.29. (Bijlsma, 2005)

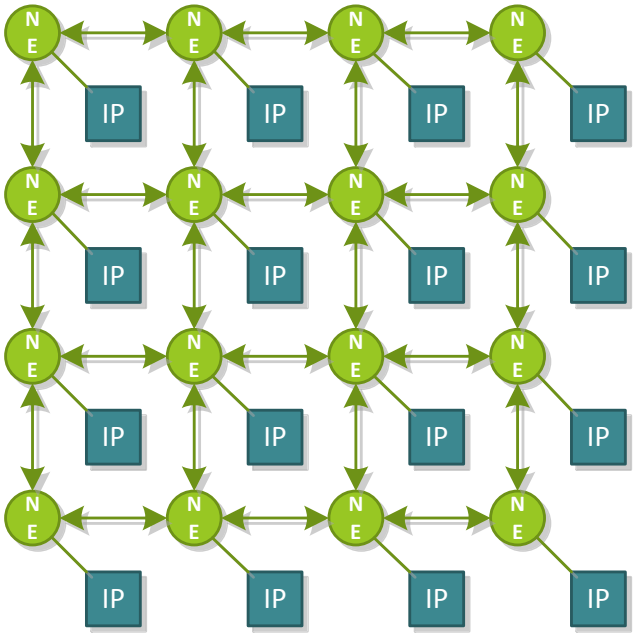


Figura. 2.29. Topología 2D-Mesh

Una topología del tipo mesh no requiere de una alta complejidad en su diseño pues su enrutamiento es sencillo y presenta un bajo consumo de energía, razones que hacen que esta topología sea fácilmente escalable. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

2D-Torus

La topología 2D-Torus es similar a la topología mesh, con la diferencia de que cada elemento de conmutación ubicado en los bordes de esta topología está conectado con su otro extremo a manera de anillo (ver la Figura. 2.30). Esta topología reduce la latencia en la transmisión de mensajes. (Rivero, 2005)

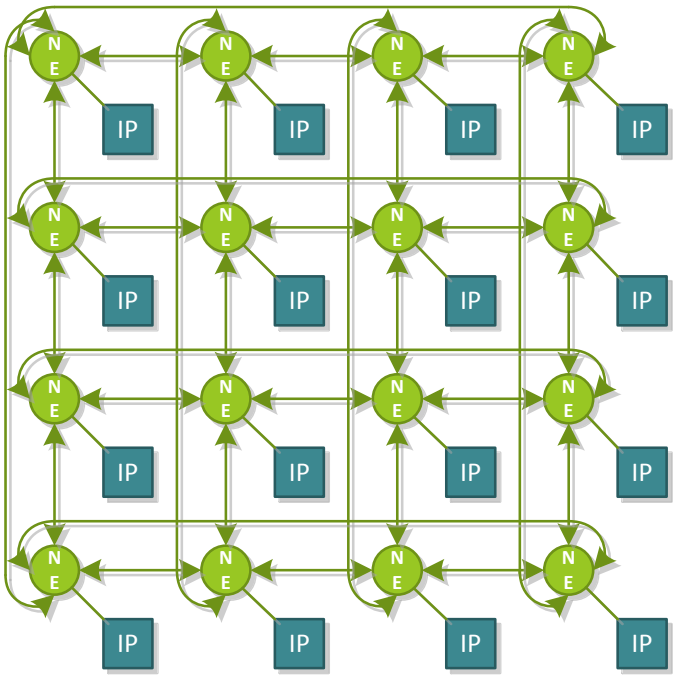


Figura. 2.30. Topología 2D-Torus

2D-Torus es la topología indicada si se desea disminuir latencia y aumentar el ancho de banda. Sin embargo no es muy escalable conforme se incrementa el número de nodos, y además su costo es mayor al momento de ser implementada. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

Honeycomb

En esta topología como se observa en la Figura. 2.31, los nodos forman un hexágono y en su centro se ubica un router que a su vez se interconecta a 2 routers más. Por lo que se puede llegar a 18 nodos con máximo un salto. Sin embargo se requiere gran número de cables y puentes que al miniaturizar dificulta la implementación de esta tecnología.

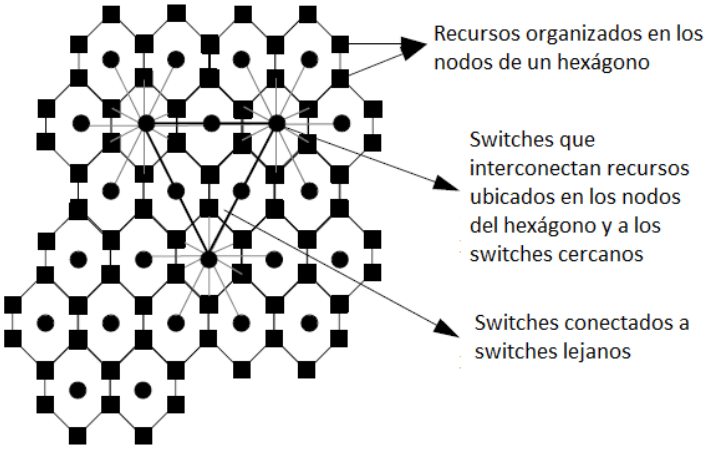


Figura. 2.31. Topología honeycomb

Fat-tree

Fat-tree es un modelo de interconexión indirecta basado en el esquema de árbol binario como se muestra en la Figura. 2.32. Al igual que árboles reales se hace más grueso a medida que nos acercamos a la raíz. El ruteo en fat-tree es muy fácil ya que hay un único camino mínimo entre nodos. Pero a medida que la red crece la congestión central se incrementa.

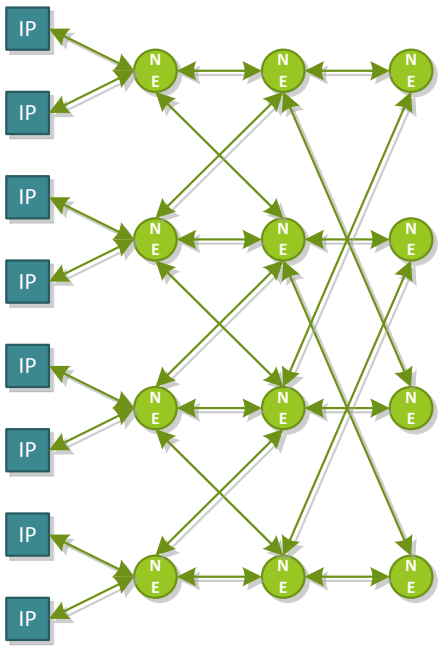


Figura. 2.32. Topología Fat-Tree

2.3.2.1. Clasificación de las topologías

Dentro de las topologías se pueden encontrar dos clasificaciones: las dependientes y las independientes. La primera se basa en arreglos y en topologías bien definidas y la segunda emplea bloques de construcción que pueden tener interconexiones personalizadas en función de su aplicación específica.

- **Topologías NoC Dependientes**

Las Topologías NoC Dependientes son bastante flexibles y son muy empleados para la interconexión de MPSoC Homogéneos pues se pueden ajustar a las necesidades de una aplicación específica. Como se mencionó anteriormente las topologías regulares son las más empleadas y por tanto son muy consideradas debido a su capacidad de mantener propiedades similares como, la misma lógica de enrutamiento y la evasión de estancamientos o *deadlock*. (Guerrier & Greiner, 2000)

En la Figura. 2.33 se puede apreciar un ejemplo de cómo es la implementación de una topología dependiente implantada en la comunicación de un MPSoC Homogéneo.

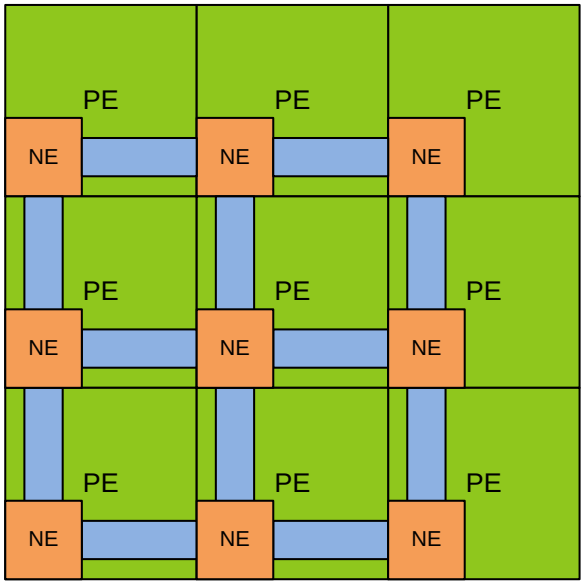


Figura. 2.33. Topología Dependiente para un MPSoC Homogéneo

Definición de estancamiento o deadlock

Un *deadlock* es una situación en la cual un paquete de datos espera por un evento que nunca sucederá, por lo general la liberación de un recuso de red.

La liberación de un recurso de red sucede cuando un mensaje del nodo A debe pasar al nodo B, esto se realiza solicitando un espacio en el búfer del nodo B, atravesar del espacio físico entre ambos nodos y así liberar el espacio del nodo A, en la Figura. 2.34 se puede apreciar este hecho. (Rivero, 2005)

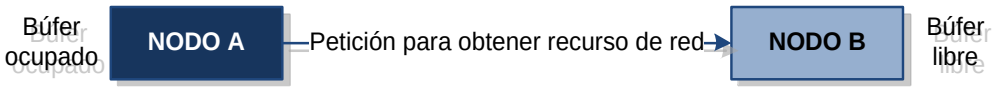


Figura. 2.34. Ejemplo de una dependencia entre nodos

Por lo tanto una situación de estancamiento o *deadlock*, sucede cuando una solicitud de paso de información de un nodo a otro nunca se produce debido a que no se puede liberar espacio en el búfer del nodo de destino.

- **Topologías NoC Independientes**

Las Topologías NoC Independientes son personalizables de acuerdo a la aplicación del MPSoC al cual va orientado, y su diseño requiere de un gran esfuerzo, pues se debe seleccionar la topología más adecuada. Las NoC que son implantadas en MPSoC Heterogéneos deben poseer una topología irregular personalizable bastante flexible, lo que implica la selección de una arquitectura de red compleja y el incremento de costos de implementación. En las topologías NoC independientes, sus complejas interconexiones y su inflexibilidad, aumentan el riesgo de estancamientos o pérdidas de información, razón por la cual se debe prestar más atención al momento de realizar cada algoritmo de interconexión. (Guerrier & Greiner, 2000)

En la Figura. 2.35 se puede apreciar la compleja estructura que puede presentar la implantación de una topología independiente para la comunicación de un MPSoC Heterogéneo.

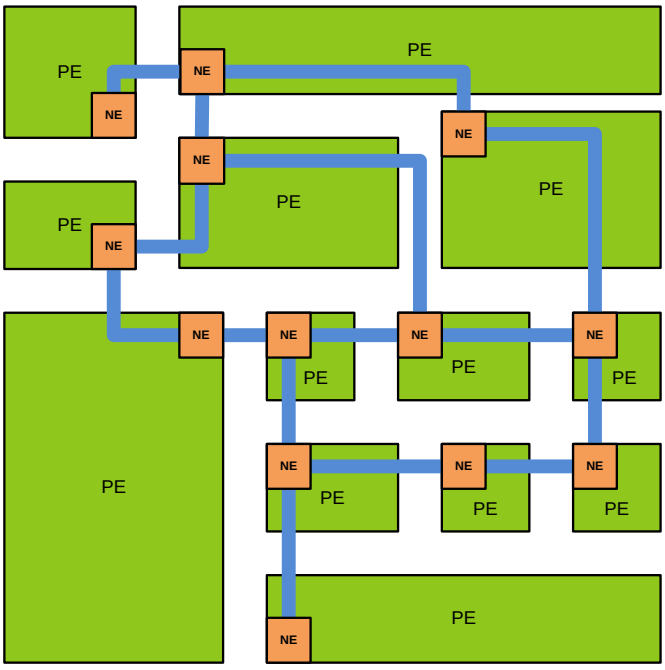


Figura. 2.35. Topología Independiente para un MPSoC Heterogéneo

2.3.3. Técnicas de Enrutamiento

Anteriormente se había dado un ejemplo de cómo se transmite un paquete de datos a través de una red, para con ello poder definir que es una situación de estancamiento o *deadlock*. El hecho de que la información debe llegar de un nodo a otro, implica la existencia de una infraestructura de comunicación entre todos los nodos del sistema y que estos puedan enviar mensajes entre sí por medio de la red. Para conseguir este objetivo se hace uso de técnicas de enrutamiento con el fin de especificar el camino que tendrá que seguir un paquete desde el nodo de origen hasta el nodo de destino (Rivero, 2005). Así tenemos:

- Técnica de Enrutamiento Basada en Fuentes.
- Técnica de Enrutamiento Basada en Tablas.
- Técnica de Enrutamiento Basada en Algoritmos.

- **Técnica de Enrutamiento Basadas en Fuentes**

La técnica de enrutamiento basada en fuentes hace uso de las tablas de ruteo indexadas según el destino. En las tablas de ruteo se escoge la ruta más apropiada para el envío de un mensaje y cuando existe más de una ruta disponible se la escoge aleatoriamente. (Rivero, 2005)

En este tipo de técnica todas las decisiones se toman en el nodo de origen, brindando de este modo mayor velocidad en la transmisión de datos. Tras seleccionar la ruta, se pierde la necesidad de realizar un cómputo a medida que el paquete avanza hacia su destino. Sin embargo su desventaja es que no puede adaptarse a las situaciones de congestión que pudieran presentarse en la topología de la red. (Rivero, 2005)

- **Técnica de Enrutamiento Basadas en Tablas**

En esta técnica las tablas de enrutamiento contienen la información del próximo nodo al cual viajará el paquete. Estas tablas son a su vez, almacenadas en todos los elementos de la red con lo que se ocupa un menor espacio, pues solo se almacena la información correspondiente al siguiente nodo de destino en lugar de la ruta completa. La técnica basada en tablas puede adaptarse a la congestión que se presenta en la red al no seleccionar una salida adecuada. A pesar de sus ventajas la distribución de las tablas de enrutamiento genera un aumento en latencia. (Rivero, 2005)

- **Técnica de Enrutamiento Basadas en Algoritmos**

En las técnicas mencionadas anteriormente se hace uso de tablas de enrutamiento para el envío de información entre los nodos de la red. Estas técnicas presentan como principal ventaja la adaptabilidad a cualquier topología. Por el contrario, los algoritmos de enrutamiento son desarrollados para una determinada topología y no usan tablas de enrutamiento, sino que se hace uso de un algoritmo para poder calcular la ruta que después será implementada por medio de lógica combinacional. En la actualidad existen muchos algoritmos de enrutamiento probados que funcionan eficientemente tales como XY, WestFirst, NorthLast, etc. (Rivero, 2005)

2.3.4. Técnicas de Conmutación

Las técnicas de conmutación empleadas para el desarrollo de una NoC se clasifican de acuerdo a las características de la red. Durante el proceso de síntesis de cada topología NoC se define el tamaño de los conmutadores y la conectividad de los componentes de tal manera que no exista interbloqueos en el enrutado para los distintos flujos de datos entre los componentes existentes de la NoC. Por lo tanto, es un método utilizado para la transmisión de información entre los diferentes nodos que pueden existir dentro del área del MPSoC.

Existen varias técnicas de conmutación que se pueden clasificar en dos grandes grupos: la conmutación de circuitos y la conmutación de paquetes.

Conmutación de Circuitos

La conmutación de circuitos se encarga de reservar la ruta física antes de comenzar la transmisión de la información. La transferencia de información desde el nodo de origen hacia el nodo de destino a través de un espacio físico proporciona garantías de Quality of Service (QoS) y un mínimo espacio en el búfer, a través de un ancho de banda que está determinado, presentando así una baja latencia. Sin embargo, presenta desventajas claras como tiempo elevado en el establecimiento de la conexión, la necesidad de un reloj global, desperdicio de ancho de banda cuando no se está enviando información y gran inflexibilidad. (Rivero, 2005)

Conmutación de Paquetes

La conmutación de paquetes, a diferencia de la conmutación de circuitos, transmite la información dividida en paquetes, es decir, el envío dependerá de la cantidad de información contenida en cada paquete, así se determinará que el tiempo de la transmisión sea prolongado o corto. Esta técnica de conmutación se utiliza en paquetes de tamaño variable, pues según el paquete se debe establecer el espacio del búfer, los retardos en función de la congestión y los algoritmos de ruteo, para su debida transmisión. (Bijlsma, 2005)

Dentro de la conmutación de paquetes se presentan otras clasificaciones, las cuales son: Store & Forward, Virtual Cut-Through y Wormhole.

- **Store & Forward (S&F)**

La conmutación Store & Forward envía el paquete entero al siguiente nodo, siempre y cuando, se haya garantizado espacio en el búfer del nodo de destino, el mismo que deberá mantener el paquete entero. De esta forma ya no es necesario dividir el paquete en flits (Mirza-Aghatabar, Koohi, Hessabi, & Pedram). El *flit* es la unidad de medida empleada para el control de flujo de mensajes, siendo los búferes de entrada y salida del router lo suficientemente grandes para almacenar algunos flits. La desventaja con esta técnica es que el Switch se puede demorar en recibir información o puede no tener suficiente espacio en el búfer. (Bijlsma, 2005)

- **Wormhole**

Esta técnica alcanza mínima latencia en la transmisión de paquetes, para lo cual los paquetes de datos son cada vez más segmentados en flits. Una vez hecha la segmentación se envía un primer flit de cabecera el cual almacenará la información necesaria del nodo de destino así como cada Switch de paso, para el ruteo de los demás paquetes, con lo que se establecerá el canal para que todos los paquetes crucen como un “gusano”, al final, el canal se libera por medio del último paquete o cola del flit. La desventaja de usar la técnica de conmutación por wormhole es que un paquete puede ocupar varios switches al mismo tiempo. (Bijlsma, 2005)

- **Virtual Cut-Through (VCT)**

Virtual Cut-Through es una técnica similar al Wormhole, pero en este caso la cabecera del primer flit enviado es examinada inmediatamente por el switch de paso y transmitido al siguiente switch sin la necesidad de tener que almacenar todo el paquete en el mismo switch. Así, el proceso continúa hasta llegar al nodo de destino. Cuando se presenta un bloqueo en la salida debido a que el puerto no está disponible, la técnica por VCT pasa a comportarse de forma similar a la técnica por S&F. (Rivero, 2005)

2.3.5. Control de flujo

El control de flujo se encarga de administrar los recursos de la red, tales como el espacio del búfer y el ancho de banda que se asignan a cada paquete de datos

transmitido por la red (Agarwal, Iskander, & Shankar, 2009). El control de flujo sincroniza a los elementos de la red (NE) y a las interfaces de red (NI), empleando un sistema compatible *Globally Asynchronous, Locally Synchronous* (GALS), con el fin de crear regiones aisladas en el chip que se comuniquen de forma sincrónica, y una área global con comunicación asincrónica. (Bijlsma, 2005)

El control de flujo se puede clasificar en: Credit-Based y Handshake.

- **Credit-Based**

Credit-Based emplea contadores para poder ejecutar el envío y la recepción de un flit a través de la red. La función del contador es la de incrementarse cuando transmite un flit y decrementarse cuando recibe dicho flit. Para el caso en que el contador tenga valor de cero ya no se receptan más flits en el Switch, debido a que el búfer se encuentra lleno, hasta que este se vuelva a habilitar. (Bijlsma, 2005)

- **Handshake**

Con Handshake los flits se envían una vez que se encuentran disponibles en el siguiente nodo, el cual envía un reconocimiento o acknowledgement (ACK) al nodo de origen para verificar la entrega. Caso contrario se tiene dos alternativas, retardar o descartar el flit y transmitir una señal negativa de reconocimiento o negative acknowledgement (NACK) al nodo de origen. (Bijlsma, 2005)

El retardo de flits significa que nunca se descartan, sino que estos son almacenados hasta su reconocimiento, haciendo que la entrega de los flits se retrase. Por otro lado, el descarte de flits significa que son retransmitidos hasta su reconocimiento. (Bijlsma, 2005)

De los dos métodos para el control de flujo mencionados anteriormente, el más viable para su implementación es el que emplea Handshake, ya que este método es más económico que Credit-Based, y aunque existe un modelo basado en este método denominado stall-and-go más económico que el Handshake, se desconoce su uso en tecnologías actuales para la implementación de NoC, esto debido a que no ofrece tolerancia a fallos.

2.3.6. Estado del Arte de NoC

A continuación se presenta el estado del arte de NoC con el objetivo de proporcionar una idea general de las propuestas de investigación sobre Networks-On-Chip, así como su desarrollo en la actualidad. Cada propuesta contiene un listado de datos considerados relevantes desde el punto de vista cualitativo y cuantitativo para su implementación. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

NoC SPIN

- Implementada en el año 2000 sobre un ASIC, cuyo layout ocupa un área de 4.6mm^2 utilizando tecnología de fabricación CMOS de 130nm.
- Topología Fat-tree.
- Posee una estrategia de conmutación determinística y adaptativa.
- El tamaño del Flit es de 32 bits para datos y 4 bits para control

- Tiene como almacenamiento temporal una cola de entrada y dos colas compartidas de salida.
- Utiliza como interface de conmutación el bloque IP VCI (Identificador de canal virtual)
- El área del switch es de 0.24mm^2 CMOS 130nm
- El desempeño pico estimado es de 2 Gbits/s por cada switch. (Guerrier & Greiner, 2000) (Andriahantenaina & Greiner, Micro-network for SoC: Implementation of a 32-port SPIN network, 2003) (Andriahantenaina & Charlery, SPIN: a Scalable, Packet Switched, On-chip Micro –network, 2003)

NoC aSoC

- Implementada en el año 2000 sobre un ASIC, utilizando tecnología de fabricación CMOS de 350nm.
- Topología 2D-Mesh escalable.
- Posee una estrategia de conmutación determinada por la aplicación.
- El tamaño del Flit es de 32 bits.
- No tiene almacenamiento.
- No utiliza una interface de conmutación que use un módulo IP.
- El área del switch está definida por 50000 transistores.
- El desempeño pico estimado es de 2 Gbits/s por cada switch.
- Soporta QoS mediante conmutación de circuitos. (Jian, Swaminathan, & Tessier, 2000)

NoC Dally

- Implementada en el año 2001.
- Topología Folded 2D-Torus 4x4.
- Posee una estrategia de conmutación XY – Source.
- El tamaño del Flit es de 256 bits para datos y 38 bits para control.
- Tiene como almacenamiento temporal una cola de entrada.
- No utiliza una interface de conmutación que use un módulo IP.

- El área del switch es de 0.59mm^2 CMOS 100nm (6.6% corresponde a la región síncrona).
- El desempeño pico estimado es de 4 Gbits/s por cable.
- Soporta QoS mediante Throughput garantizado por canales virtuales. (Dally & Towles, 2001)

NoCArc

- Implementada en el año.
- Topología 2D-Mesh escalable.
- El tamaño del Flit es de 290 bits para datos y 10 bits para control.
- Tiene como almacenamiento temporal una cola de entrada y una cola de salida. (Kumar, 2002)

NoC Sgroi

- Implementada en el año 2001.
- Topología 2D-Mesh.
- El tamaño del Flit es de 18 bits para datos y 2 bits para control.
- Utiliza como interface de conmutación el bloque IP OCP (Open Core Protocol). (Sgroi, et al., 2001)

NoC Octagon

- Implementada en el año 2001.
- Topología Chordal Ring de 8 nodos, tipo anillo.
- Posee una estrategia de conmutación distribuida y adaptativa.
- El tamaño del Flit es variable para datos y 3 bits para control
- El desempeño pico estimado es de 40 Gbits/s.

- Soporta QoS mediante conmutación de circuitos. (Karim, Nguyen, & Dey, An interconnect architecture for network systems on chips, 2000) (Karim, Nguyen, Dey, & Rao, On-chip communication architecture for OC-768 network processors, 2001)

NoC MARESCAUX

- Implementada en el año 2002 sobre un FPGA Virtex/Virtex II.
- Topología 2D-Torus escalable.
- Posee una estrategia de conmutación XY Blocking, basada en saltos determinísticos.
- El tamaño del Flit es de 16 bits para datos y 3 bits para control.
- Tiene como almacenamiento temporal una cola virtual de salida.
- Utiliza como interface de conmutación el bloque IP-Switch personalizable.
- El área del switch ocupa 446 porciones para Virtex/Virtex II (4.8% de área de overhead para XCV800)
- El desempeño pico estimado es de 320 Mbits/s por cada canal virtual a 40 MHz.
- Soporta QoS mediante dos canales virtuales multiplexados. (Marescaux, Bartic, Verkest, Vernalde, & Lauwereins, 2002)

NoC RIJPKEMA

- Implementada en el año 2002 sobre un ASIC.
- Topología 2D-Mesh.
- Posee una estrategia de conmutación determinística y adaptativa.
- El tamaño del Flit es de 32 bits.
- Tiene como almacenamiento temporal una cola de entrada.
- El área del switch es de 0.26mm^2 CMOS 120nm
- El desempeño pico estimado es de 80 Gbits/s por cada switch.
- Soporta QoS mediante conmutación de circuitos y Throughput garantizado. (Rijpkema, Goossens, & Radulescu, Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip, 2003)

(Rijpkema, Goossens, & Wielage, A Router Architecture for Networks on Silicon, 2001)

NoC ECLIPSE

- Implementada en el año 2002.
- Topología 2D-Mesh dispersa jerárquicamente.
- El tamaño del Flit es de 68 bits.
- Tiene como almacenamiento temporal una cola de salida.
- No soporta QoS. (Forsell, 2002)

NoC PROTEO

- Implementada en el año 2002 sobre un ASIC, utilizando tecnología de fabricación CMOS de 350nm.
- Topología de anillo bidireccional.
- Posee una estrategia de conmutación determinística y adaptativa.
- El tamaño del Flit es variable, tanto para datos como para control.
- Utiliza como interface de conmutación el bloque IP VCI (Identificador de canal virtual). (Guerrier & Greiner, 2000) (Kumar, 2002) (Petrini & Vanneschi, 1997)

NoC QNoC-Technion

- Implementada en el año 2003 y desarrollada por el Instituto Tecnológico de Israel.
- Topología 2D-Mesh.
- Posee una estrategia de conmutación tipo wormhole y algoritmo de enrutamiento XY.

- Define cuatro clases de tráfico (Leroy, 2006-2007):
    - Señalización (señales de control en los bloques locales).
    - En tiempo real (retardos fuertes).
    - De lectura/escritura (corto acceso a los datos).
    - Transferencia en bloques (grandes ráfagas de datos).

NoC HERMES

- Implementada en el año 2003 sobre un FPGA Virtex II y desarrollada en la Facultad de Informática de la Pontifica Universidad Católica del Rio Grande del Sur do Brasil (PUCRS).
- Topología 2D-Mesh escalable.
- Posee una estrategia de conmutación XY.
- El tamaño del Flit es de 8 bits para datos y 2 bits para control
- Tiene como almacenamiento temporal una cola de entrada.
- Utiliza como interface de conmutación el bloque IP OCP (Open Core Protocol).
- El área del switch ocupa 631 tablas de búsquedas o Look-up Table (LUTs) y 316 porciones en Virtex II.
- El desempeño pico estimado es de 500 Mbits/s por cada switch a 25 MHz.
- No soporta QoS. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

Hermes es ampliamente tratada en el **Capítulo 5,** por ser la arquitectura de red sobre la cual se trabajó en el presente proyecto de grado.

NoC NOSTRUM

- Implementada en el año 2004 sobre un ASIC, utilizando tecnología de fabricación CMOS de 180nm.
- Topología 2D-Mesh escalable.
- Posee una estrategia de conmutación XY.
- El tamaño del Flit es de 128 bits para datos y 2 bits para control

- No tiene almacenamiento temporal.
- El desempeño pico estimado es de 4Gbits/s.
- Soporta QoS mediante circuitos virtuales. (Hemani, et al., 2000)

NoC Xpipes

- Implementada en el año 2004 sobre un FPGA Altera Cyclone II.
- Topología no específica con el fin de no restringir el diseño.
- Posee una estrategia de conmutación tipo wormhole.
- Utiliza como interface de conmutación el bloque IP OCP (Open Core Protocol).
- El área del switch ocupa 161 de 444 elementos lógicos del FPGA.
- El desempeño pico estimado es de 22,4 Gbits/s por cada switch a 350 MHz. (Dall'Osso, Biccari, Giovannini, Bertozzi, & Benini)

NoC ANOC-CEA

- Implementada en el año 2005 sobre un STMicroelectronics de tecnología a 130nm.
- Topología 2D-Mesh irregular debido a que utiliza algoritmos de enrutamiento de origen.
- El tamaño del Flit es de 34 bits.
- El área del switch es de 0.21mm^2.
- El desempeño pico estimado es de 160Mflit/s en el peor de los casos y de 220Mflit/s en condiciones normales. (Edith, et al.) (Miro-Panades, Clermidy, Vivet, & Greiner)

NoC MANGO-DTU

- Implementada en el año 2005 y desarrollada por la Universidad Técnica de Dinamarca, cuyo pre-layout se estima en 0.277mm^2 utilizando tecnología de fabricación de 130 nm.
- Combina un switch best-effort y un switch garanty-services para la conmutación de información.
- El tamaño del Flit es de 33 bits.
- El desempeño pico estimado es de 420Mflits/s por cada puerto de sus switches, pudiendo mantener 7 conexiones independientes por puerto de la red. (Leroy, 2006-2007) (Marescaux, Mapping and Management of Communication Services on MP-SoC Platforms, 2007)

NoC AEthereal

- Implementada en el año 2005 y desarrollada por Philips, utilizando tecnología de fabricación CMOS de 130nm.
- Topología 2D-Mesh.
- Posee una estrategia de enrutamiento determinista y técnicas de conmutación wormhole.
- El tamaño del Flit es de 32 bits.
- Utiliza como interface de conmutación 4 protocolos estándar:
    - Maestro/Esclavo
    - OCP, Open Core Protocol
    - DTL, Diode-Transistor Logic
    - AXI, Advanced eXtensible Interface
- El desempeño pico estimado es de 16 Gbits/s por puerto.
- Soporta QoS en tiempo real tanto para Throughput, como para latencia delimitada, reduciendo considerablemente el overhead. (Goossens, Dielissen, & Radulescu)

NoC uSPIDER

- Implementada en el año 2007.
- Topología personalizable mediante código VHDL.
- Posee una estrategia de conmutación determinista para facilitar la reconfiguración en tiempo de ejecución y para optimizar el overhead.
- Tiene como almacenamiento temporal un buffer de tamaño configurable.
- Soporta varios niveles de QoS y alto Throughput. (Evain, Diguet, & Houzet)

ESTADO DEL ARTE DE NOC									
NoC	**Topología**	**Conmutación**	**Tamaño de Flit**	**Almacenamiento Temporal**	**Interfaz de conmutación**	**Área del Switch**	**Desempeño pico estimado**	**Soporta QoS**	**Implementación**
SPIN [Año 2000]	Fat-tree	Determinística y adaptativa	32 bits para datos y 4 bits para control	Una cola de entrada y dos colas compartidas de salida	Bloque IP VCI	0.24mm^2 CMOS 130nm	2 Gbits/s por cada switch		ASIC, layout de 4.6mm^2 CMOS 130nm
aSoC [Año 2000]	2D-Mesh escalable	Determinada por la aplicación	32 bits			50000 transistores	2 Gbits/s por cada switch	Conmutación de circuitos	ASIC, CMOS 350nm
Dally [Año 2001]	Folded 2D-Torus 4x4	XY – Source	256 bits para datos y 38 bits para control	Una cola de entrada		0.59mm^2 CMOS 100nm	4 Gbits/s por cable	Throughput garantizado por canales virtuales	
NoCArc [Año 2001]	2D-Mesh escalable		290 bits para datos y 10 bits para control	una cola de entrada y una cola de salida					
Sgroi [Año 2001]	2D-Mesh		18 bits para datos y 2 bits para control		Bloque IP OCP				
Octagon [Año 2001]	Chordal Ring de 8 nodos	Distribuida y adaptativa	variable para datos y 3 bits para control				40 Gbits/s	Conmutación de circuitos	
MARESCAUX [Año 2002]	2D-Torus escalable	XY Blocking, basada en saltos determinísticos	16 bits para datos y 3 bits para control	Una cola virtual de salida	Bloque IP-Switch personalizable	446 porciones para Virtex/Virtex II	320 Mbits/s por cada canal virtual a 40 MHz	Dos canales virtuales multiplexados	FPGA Virtex/Virtex II

ESTADO DEL ARTE DE NOC									
NoC	**Topología**	**Conmutación**	**Tamaño de Flit**	**Almacenamiento Temporal**	**Interfaz de conmutación**	**Área del Switch**	**Desempeño pico estimado**	**Soporta QoS**	**Implementación**
RIJPKEMA [Año 2002]	2D-Mesh	Determinística y adaptativa	32 bits	Una cola de entrada		$0.26mm^2$ CMOS 120nm	80 Gbits/s por cada switch	Conmutación de circuitos y Throughput garantizado	ASIC
ECLIPSE [Año 2002]	2D-Mesh dispersa jerárquicamente		68 bits	Una cola de salida					
PROTEO [Año 2002]	Anillo bidireccional	Determinística y adaptativa	variable, tanto para datos como para control		Bloque IP VCI				ASIC, CMOS de 350nm
QNoC [Año 2003]	2D-Mesh	Wormhole y algoritmo de enrutamiento XY							Instituto Tecnológico de Israel
HERMES [Año 2003]	2D-Mesh escalable	XY	8 bits para datos y 2 bits para control	Una cola de entrada	Bloque IP OCP	631 LUTs y 316 porciones en Virtex II	500 Mbits/s por cada switch a 25 MHz		FPGA Virtex II
NOSTRUM [Año 2004]	2D-Mesh escalable	XY	128 bits para datos y 2 bits para control				4Gbits/s	Circuitos virtuales	ASIC, CMOS de 180nm

ESTADO DEL ARTE DE NOC									
NoC	**Topología**	**Conmutación**	**Tamaño de Flit**	**Almacenamiento Temporal**	**Interfaz de conmutación**	**Área del Switch**	**Desempeño pico estimado**	**Soporta QoS**	**Implementación**
XPIPES [Año 2004]		Wormhole			Bloque IP OCP	161 de 444 elementos lógicos del FPGA	22,4 Gbits/s por cada switch a 350 MHz		FPGA Altera Cyclone II
ANOC [Año 2005]	2D-Mesh irregular	switch best-effort switch garanty-services	34 bits			0.21mm^2	160Mflit/s 220Mflit/s		STMicroelectronics de 130nm
MANGO [Año 2005]			33 bits				420Mflits/s por cada puerto de sus switches		pre-layout 0.277mm^2 130 nm
AETHEREAL [Año 2005]	2D-Mesh	Determinista Wormhole	32 bits		Maestro/Esclavo OCP, DTL, AXI		16 Gbits/s por puerto	Tiempo real Throughput, latencia delimitada	Philips, CMOS de 130nm
uSpider [Año 2007]	personalizable VHDL	Determinista		Buffer de tamaño configurable				Varios niveles de QoS y alto Throughput	

Tabla. 2.2. Estado del Arte NoC

CAPÍTULO 3

TÉCNICAS DE DISEÑO

3.1. CO-DISEÑO DE HARDWARE Y SOFTWARE

Gran parte de los sistemas electrónicos digitales actuales están constituidos por componentes de hardware y software. Los componentes de software son programas desarrollados en lenguajes de alto nivel que corren sobre uno o más procesadores programables, mientras que los componentes de hardware son la parte tangible del sistema. Los circuitos digitales constituyen una clase de componentes de hardware y se clasifican en dos tipos que son:

- Circuitos Estándar
- Circuitos ASIC (Application-Specific Integrated Circuit)

Los *Circuitos Estándar* son usados en diversas aplicaciones y se fabrican en serie, en donde es el fabricante quien proporciona las características técnicas del componente. Por otro lado los *Circuitos ASIC* son como su nombre lo indica, son creados para una aplicación específica. Sin embargo para ambos casos la combinación hardware/software busca satisfacer las especificaciones relacionadas con el desempeño, la confiabilidad, el costo y el tiempo. (Wolf, Hardware-Software Co-Design of Embedded Systems, 2004)

La implementación tradicional de este tipo de sistemas se basa en el desarrollo de los componentes de software y hardware de manera separada. Este enfoque restringe la posibilidad de realizar una exploración amplia del espacio de diseño para cada aplicación. Esta situación dificulta la correlación entre hardware y software, ocasionando la introducción de errores en el proceso de integración final de las componentes; lo cual incide en el aumento del costo y tiempo de desarrollo. (Chiodo, et al., 2003)

Estas dificultades han impulsado el desarrollo de nuevas herramientas de proyección Asistido por Computador (CAD, Computer-Aided Design), donde es posible la especificación, la simulación y la síntesis concurrente de los componentes de hardware y software. El objetivo de estas nuevas metodologías es cumplir con los requerimientos del diseño a nivel de sistema explotando la sinergia existente entre el hardware y el software mediante un planteamiento concurrente. (Chiodo, et al., 1994)

3.1.1. Evolución del Diseño Electrónico

En los inicios, se esbozaba todo el nuevo circuito integrado a mano, transistor a transistor, indicando sus dimensiones, su ubicación en el plano base y su conexionado con el resto de componentes del circuito. A esto se le conoce como Bottom-Up o Diseño Ascendente. Según este tipo de metodología, se comienza a partir de los elementos más pequeños para conseguir que la suma de sus efectos realice la aplicación deseada. (Ou & Prasanna)

La metodología Bottom-Up obvia una estructuración jerárquica de los elementos del sistema. Simplemente reúne componentes de bajo nivel para formar el diseño global, por lo que se trata de una técnica poco eficiente cuando

se pretende hacer funcionar un sistema compuesto por miles de componentes de bajo nivel.

En un diseño Bottom-Up se comienza realizando una descripción con esquemas de los componentes del circuito (Figura. 3.1). Estos componentes se construyen normalmente a partir de otros que pertenecen a una biblioteca que contiene componentes básicos, que representan unidades funcionales con significado propio dentro del diseño.

Figura. 3.1. Diseño Button-Up

En este caso, el sistema no puede comprobarse o simularse hasta que no está totalmente terminado, y un gran número de decisiones críticas a nivel de tecnología de fabricación deben tomarse al comienzo del diseño. Es por ello que, cuando los proyectos son complejos o de gran envergadura, esta forma de trabajo conlleva un alto costo en la producción, ya que los fallos y errores durante el diseño son frecuentes debido a la naturaleza, sensibilidad a los cambios y/o ajustes de los elementos más pequeños. (Antoni, 2007)

La consolidación en la década de los noventa de los lenguajes de descripción de hardware (HDL, Hardware Description Languages) ha permitido, la implantación progresiva de las denominadas metodologías Top-Down o Diseño Descendente que, en contraposición a la metodología Bottom-Up, permiten la descripción del sistema al más alto nivel. En el diseño Top-down (Figura. 3.2) la dependencia de la implementación final es prácticamente inexistente en las etapas de definición funcional, y se irá concretando en las sucesivas fases del proyecto, hasta llegar a la síntesis de un sistema sobre cualquiera de las tecnologías existentes: full-custom, ASICs, FPGAs, CPLDs. (Al-Hadithi & Muro, 2004)

El diseño Top-Down consiste en capturar una idea con un alto nivel de abstracción, implementarla partiendo de la misma, e incrementar el nivel de detalle según sea necesario. El sistema inicial se subdivide en módulos, estableciendo una jerarquía y cada módulo se subdivide cuantas veces sea necesario hasta llegar a los componentes primarios del diseño. La descripción del circuito a distintos niveles de detalle, así como la verificación y simulación del mismo, permiten reducir la posibilidad de incluir errores. (Antoni, 2007)

El diseño Top-Down ofrece como ventaja que la información se estructure de forma modular. El diseño se realiza a partir del sistema completo y se subdivide en módulos, las subdivisiones se realizan de forma que sean funcionalmente independientes en planeación, revisión y verificación. (Al-Hadithi & Muro, 2004)

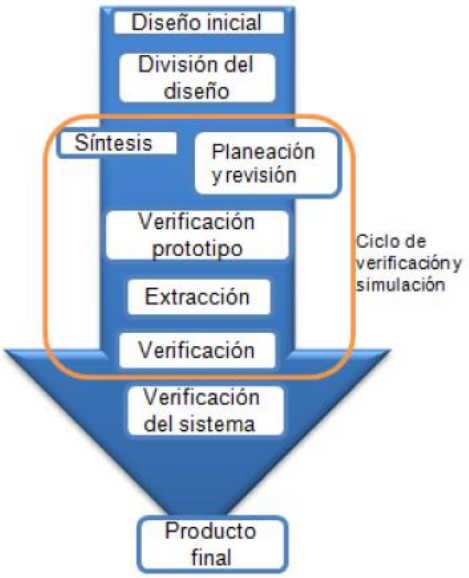


Figura. 3.2. Diseño Top-Down

La forma más común de describir un circuito es mediante la utilización de esquemas, que son una representación gráfica de lo que se pretende realizar. Con la aparición de herramientas EDA (Electronic Design Automation) cada vez más complejas, que integran en el mismo marco de trabajo las herramientas de descripción, síntesis, simulación y realización, se generó la necesidad de disponer de un método de descripción de circuitos, que permitiera el intercambio de información entre las diferentes herramientas que componen el ciclo de diseño.

3.1.2. Metodología de Diseño

La cada vez más imperiosa demanda del mercado, obligó a los fabricantes de herramientas de diseño electrónico EDA, a dotarlas de capacidades de desarrollo que posibiliten la transformación de una idea inicial en un diseño final, de forma rápida, eficiente, fiable y, por supuesto, con un bajo costo asociado. Habitualmente en una herramienta clásica EDA se creaba un ciclo de diseño basado en prototipos que se iban mejorando hasta conseguir un producto final. En la actualidad el esfuerzo se concentra en el nivel conceptual, verificando

previamente la funcionalidad del sistema sin necesidad de llegar a un nivel de detalle excesivo. (Al-Hadithi & Muro, 2004)

En principio se utiliza un lenguaje de descripción que permite, mediante sentencias simples, describir completamente un circuito. Este lenguaje de descripción toma el nombre de Netlist, y es un conjunto de instrucciones que indican las interconexiones entre los componentes de un diseño. Conforme las herramientas de diseño se vuelven más sofisticadas, y la posibilidad de desarrollar circuitos digitales mediante dispositivos programables es más viable, aparece la necesidad de poder describir los circuitos mediante un lenguaje de alto nivel de abstracción. No desde un punto de vista estructural, sino desde un punto de vista funcional. (van der Putten, Voeten, Geilen, & Stevens)

Por otra parte, en comparación con una descripción mediante el uso de esquemáticos, los lenguajes de descripción de hardware (HDL) permiten que los diseñadores puedan recuperar el control a lo largo del proceso de creación de sistemas digitales complejos. Este proceso es asistido por herramientas que ayudan en el desarrollo de un sistema organizado de forma jerárquica y modular, dando lugar a la posibilidad de reutilización del código. (van der Putten, Voeten, Geilen, & Stevens)

La mayor parte de los diseños que se llevan a cabo en la actualidad implican enfrentarse a un equivalente de millones de puertas. Esto hace que el lograr a la primera un sistema que cumpla las especificaciones, sea una tarea difícil y costosa si no se lleva a cabo una cuidadosa planificación de tareas. El principal factor a considerar en el diseño de todo producto suele ser su costo final. Sin embargo en muchas ocasiones, es igualmente importante la cuantía de diseño, entendiendo por tal, no solo el valor monetario, sino también en tiempo de diseño (time to market). (van der Putten, Voeten, Geilen, & Stevens)

Además se deben considerar factores del tipo técnico, como la fiabilidad del sistema obtenido en términos de control de calidad, así como la eficiencia obtenida para el objetivo que el sistema se encuentre diseñado. El hecho de utilizar una metodología de diseño que no optimice los factores mencionados, puede llevar al desarrollo de sistemas con un excesivo costo de diseño y/o implementación. Es por ello que la aplicación de metodologías de diseño a los procesos de desarrollo de sistemas supone una clara ventaja sobre los procesos de diseño tradicionales.

Los métodos de diseño primarios utilizados en a la actualidad se pueden dividir en tres categorías:

- Diseño Guiado por Tiempo (TDD)
- Diseño Basado en Bloques (BBD)
- Diseño Basado en Plataforma (PBD)

Estas metodologías varían en función de las tecnologías utilizadas, la capacidad de diseño, y el nivel de inversión en la reutilización. La transición de metodologías ha sido un proceso que empezó por TDD, dando importantes saltos hasta llegar a BBD, y finalmente llegando a lo que hoy es PBD. A continuación en la siguiente tabla se resume las principales características comparativas de cada una de las metodologías antes mencionadas. (Martin & Chang, 1999)

CARACTERISTICAS DE DISENO	TDD	BBD	PBD
Complejidad de Diseño	5k a 250k compuertas	150k a 1500k compuertas	300k compuertas en adelante
Nivel de Diseño	RTL	Funcional / RTL	Evaluación de Arquitectura y Componentes Virtuales

CARACTERISTICAS DE DISENO	TDD	BBD	PBD
Equipo de Diseño	Pequeño y Enfocado	Multidisciplinario	Multidisciplinario y Multi grupal
Diseño Primario	Lógica Personalizada	Bloques de Contexto e Interfaces Personalizadas	Interconexión con el Sistema y el Bus
Reutilización del Diseño	Ninguno	Oportunista, Firme y Persistente	Planificada y Persistente
Componente Base del Diseño	Compuertas y Memorias	Grupos Funcionales y Núcleos	Componentes Virtuales
Arquitectura de Prueba	Escaneo	Escaneo / JTAG/Personalizada	Jerárquica/ Escaneo/ Paralelo/ JTAG/ Personalizada
Nivel de Verificación	RTL / Compuerta	Bus Funcional / RTL / Compuerta	RTL con Hardware y Software
Análisis de Tiempo	Completamente	Completamente con limitaciones de jerarquía	Jerárquico
Calculo de Retardo	Completamente	Completamente	Jerárquico
Verificación Física	Completamente	Completamente con limitaciones de jerarquía	Jerárquico

Tabla. 3.1. Comparación entre Metodologías de Diseño

Actualmente, la mayoría de proyectos se basan en el Diseño Basado en Plataforma puesto que brinda ciertas ventajas como:

- Disminución del time to market expandiendo las oportunidades y la velocidad de distribución de sus productos derivados.
- Reducción de riesgos involucrados en el diseño.
- Reutilización de grupos de IP-Cores en una arquitectura.
- Simplificación del proceso de diseño.

3.1.3. Flujo de diseño

3.1.3.1. Flujo de Diseño Tradicional

Tradicionalmente en el diseño de sistemas en los que se mezclan elementos de hardware y software, su desarrollo se ha venido realizando de forma separada, utilizando herramientas para realizar la compilación del software y herramientas para realizar la síntesis del hardware. Es así que la interacción entre ambas partes y la determinación de funcionalidades específicas tanto para hardware, como para software, se torna una problemática en el proceso de verificación del sistema en su totalidad, ocasionando la aparición de incompatibilidades.

En la Figura. 3.3 se muestra el flujo de diseño tradicional, en la cual se parte de la especificación del sistema, para proceder a ingresar al bucle de diseño. Posiblemente el sub problema más significativo del problema de diseño tradicional sea el reparto de las tareas que debe realizar el sistema entre los módulos de hardware y software. Esta partición puede realizarse atendiendo a distintos factores, como pueden ser:

- Tamaño de área y código fuente.
- Velocidad de procesamiento.
- Consumo de potencia.

En la práctica es deseable establecer un equilibrio entre todos los factores, aunque se otorguen pesos relativos a cada uno de ellos. El particionamiento suele hacerse manualmente, evaluando distintas posibilidades y considerando que existen prioridades para ciertas tareas que deben implementarse en un dispositivo en concreto. (Guzmán, 2006)

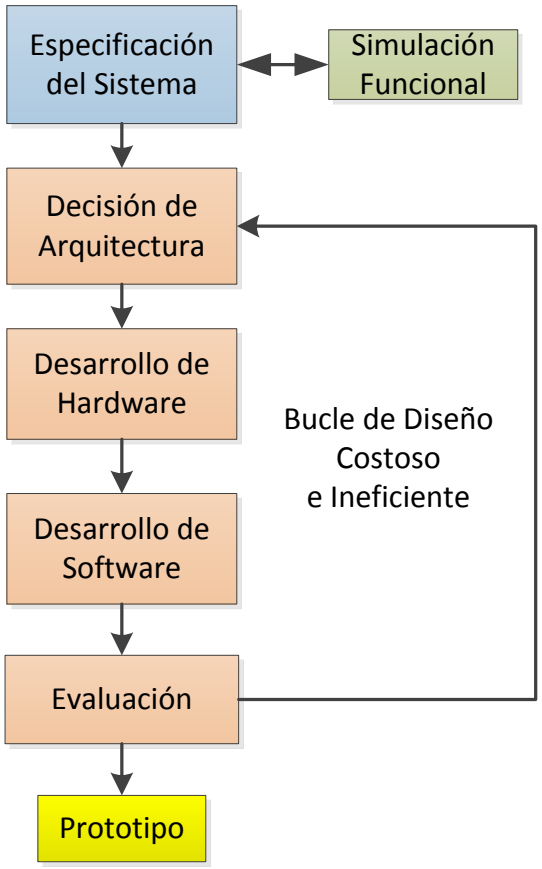


Figura. 3.3. Flujo de Diseño Tradicional

Para tratar de solventar estos inconvenientes, surge una nueva técnica de diseño: el Co-diseño. El objetivo principal del co-diseño es el desarrollo de sistemas que no sólo cumplan con las especificaciones iniciales de funcionamiento, sino que además, ayuden en el particionamietno de las tereas tanto de hardware como de software. Es así que las decisiones de lo que se hace en hardware o en software se toman utilizando criterios objetivos de optimización. (Guzmán, 2006)

3.1.3.2. Flujo de diseño con herramientas de co-diseño

A lo largo de la última década, los diseñadores han tomado conciencia de la potencia que ofrecen las técnicas de co-diseño, al posibilitar trabajar en un nivel de abstracción que se puede considerar independiente de la plataforma que sustentará el diseño final. Esto agilita la creación de módulos de diseño que no presentan las restricciones asociadas a la elección de una determinada tecnología, sistema o plataforma para su implementación. (Martin & Chang, 1999)

En la actualidad, el co-diseño es un tema en auge que ha motivado la aparición de distintas herramientas destinadas a facilitar la tarea de los diseñadores, se hallan tanto proyectos de investigación, como soluciones comerciales (ver Tabla. 3.2 y Tabla. 3.3). (Guzmán, 2006)

Herramientas Académicas de Co-diseño		
Herramienta	Desarrollador	Descripción
Polis	Univ. Berkeley	Herramienta para el desarrollo a nivel de sistema, que permite trabajar con modelos mixtos hardware / software. Centrada en una representación basada en máquinas de estado finitas. (CFSM, Co-design Finite State Machine)
Cosyma	Univ. Braunshweig	Herramienta que particiona automáticamente un conjunto de tareas sobre hardware y software, en base a una arquitectura consistente en un procesador, una memoria y componentes hardware.
Cosmos	Univ. Grenoble	Herramienta para la síntesis a nivel de sistema, orientada al diseño y síntesis de sistemas mixtos hardware / software complejos.
ASP	Univ. Queensland	Herramienta automática de co-diseño, orientada para la implementación de algoritmos con fuertes restricciones temporales.

Tabla. 3.2. Herramientas Académicas de Co-diseño

Herramientas Comerciales de Co-diseño		
Herramienta	Desarrollador	Descripción
VIOOL	ASC	Esta herramienta permite convertir VHDL en código C++, creando un hardware virtual que es posible comunicar con los posibles componentes software del diseño.
Cosmos-Galaxy	Omniview	Sistema de desarrollo que permite una especificación del hardware y del software en VHDL. Una vez decidida una estrategia de diseño, Galaxy es capaz de sintetizar el sistema en componentes y/o esquemáticos.
System Generator	Xilinx & MathWorks	Herramienta para el diseño de sistemas que usen FPGAs de Xilinx, basada en Mathlab y Simulink.
System-C	Synopsys	Lenguaje basado en C++ que permite describir los diseños a nivel de sistema.

Tabla. 3.3. Herramientas Comerciales de Co-diseño

De una forma general, al utilizar herramientas de software específicas para realizar co-diseño Hardware/Software, se espera un diagrama de flujo de diseño como el de la Figura. 3.4. Normalmente cada uno de los pasos se realiza con una herramienta diferente, por lo que la integración de las distintas herramientas en un único flujo de diseño es una tarea compleja. (Guzmán, 2006)

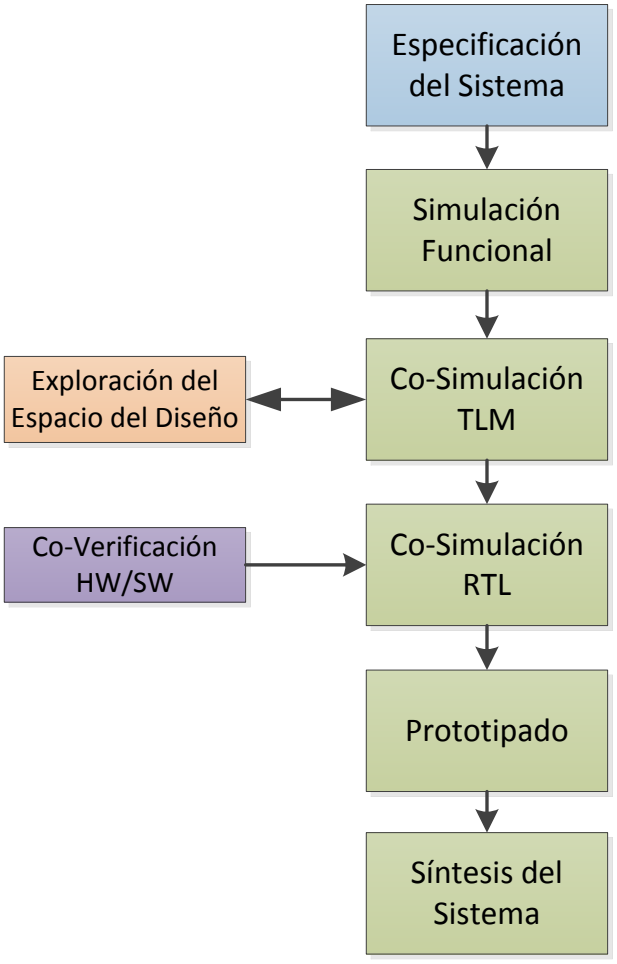


Figura. 3.4. Flujo de diseño con herramientas de co-diseño

- **Especificación del Sistema**

El primer paso para atacar la problemática de co-diseño Hardware/Software de un sistema específico, es realizar una descripción del sistema que se pretende implementar en un lenguaje de abstracción que esté por encima de los lenguajes concretos. Por ejemplo para describir el software se utiliza C y para describir el hardware se utiliza VHDL o Verilog. (Guzmán, 2006)

- **Simulación Funcional**

El siguiente paso consiste en realizar una simulación funcional del sistema descrito, para comprobar si realmente la aplicación descrita coincide con las especificaciones deseadas. Sólo cuando la descripción a alto nivel es completamente funcional tiene sentido seguir adelante con el flujo de diseño para buscar una arquitectura óptima. (Guzmán, 2006)

- **Exploración del Espacio de Diseño**

La exploración del espacio de diseño, se hace manualmente utilizando una herramienta de simulación a nivel transaccional (TLM, Transactional Level Modelling). El objetivo es evaluar las distintas alternativas de particionado y catalogarlas según rendimiento, para tomar una decisión respecto al particionado Hardware/Software que cumpla con los requerimientos del sistema. (Guzmán, 2006)

- **Co-Simulación TLM**

Se trata de una simulación a nivel transaccional. En un modelo transaccional, los detalles de la comunicación entre bloques computacionales están separados de las descripciones de dichos bloques. Los bloques computacionales y de comunicación se conectan a través de datos abstractos. La comunicación se modela a través de canales, de forma que las peticiones de transacción se realizan a través de funciones de interfaz correspondientes a los modelos de los canales de comunicación. TLM acelera la simulación y permite explorar y validar

diferentes alternativas de diseño en un nivel superior de abstracción. (Guzmán, 2006)

- **Co-Verificación Hardware/Software**

El objetivo de este paso es comprobar que, tras el particionado de Hardware/Software, nuestro sistema tiene las mismas funcionalidades que en la descripción funcional y cumple con los requisitos adicionales que se le ha impuesto. Esta co-verificación es importante ya que si bien la simulación TLM realizada anteriormente es muy eficiente en tiempo, existen aspectos prácticos de los bloques computacionales del diseño que no se han considerado aún. (Guzmán, 2006)

- **Co-Simulación RTL**

La co-verificación Hardware/Software se hace normalmente a través de una co-simulación a nivel RTL (Register Transfer Level). Para realizar esta co-simulación se necesita:

- Un programa que simule adecuadamente y de forma precisa el Hardware, por ejemplo un simulador de VHDL o Verilog.

- Un programa que simule adecuadamente y de forma precisa el comportamiento del Software, es decir, un ISS (Instruction Set Simulator) del procesador utilizado.

- Integración fluida entre ambos programas. Esto se puede conseguir modificando los simuladores para añadirles interfaces a medida, consiguiendo una sincronización en tiempo de los simuladores. (Guzmán, 2006)

- **Prototipado**

Una vez que se tienen las descripciones de las partes hardware, software, e interfaces, se pueden utilizar herramientas que existen desde hace muchos años para realizar la síntesis del hardware o Placement and Routing si el hardware se va a sintetizar sobre un circuito programable, como un FPGA, y del código objeto para el microprocesador o más conocido como compiladores. Posterior a este proceso, se demostrará de forma práctica el sistema hardware/software funcionando. (Guzmán, 2006)

- **Síntesis del Sistema**

Tras comprobar que el prototipo funciona correctamente y es apropiado para la aplicación que se quiere resolver, se procede a realizar la síntesis del sistema final. (Guzmán, 2006)

3.1.3.3. Flujo de diseño en FPGA

Uno de los sectores de mayor crecimiento dentro de la industria de los semiconductores es el de las FPGA o Field Programmable Gate Array. Teniendo un vertiginoso desarrollo a partir de su aparición en 1985.

Históricamente las FPGA surgen como una evolución de los conceptos desarrollados en un PAL (Programmable Array Logic) o un CPLD (Complex Programmable Logic Device) [69]. Un FPGA es un dispositivo semiconductor que contiene componentes lógicos cuya interconexión y funcionalidad puede ser

programable. Además incluye elementos de memoria, flip-flops, aunque los sistemas más modernos incorporan recursos especiales como bloques de RAM, elementos de multiplicación y acumulación, módulos de comunicación y procesadores empotrados. (Peña, 2008)

Los sistemas electrónicos actuales demandan cada día mayor capacidad de cómputo, por lo que se necesita de dispositivos capaces de ofrecer un buen rendimiento a velocidades de transferencia en el orden de los gigabits por segundo. Para satisfacer este incremento de potencia de cálculo se requiere de dispositivos complejos y de nuevos algoritmos de programación.

La disponibilidad de las herramientas EDA han facilitado el desarrollo de sistemas digitales con lógica programable. En el caso de usar FPGA, como dispositivo programable, su flujo de diseño incluye las siguientes etapas (ver Figura. 3.5) (Mora, 2008):

- Especificación
- Verificación
- Realización
- Depuración

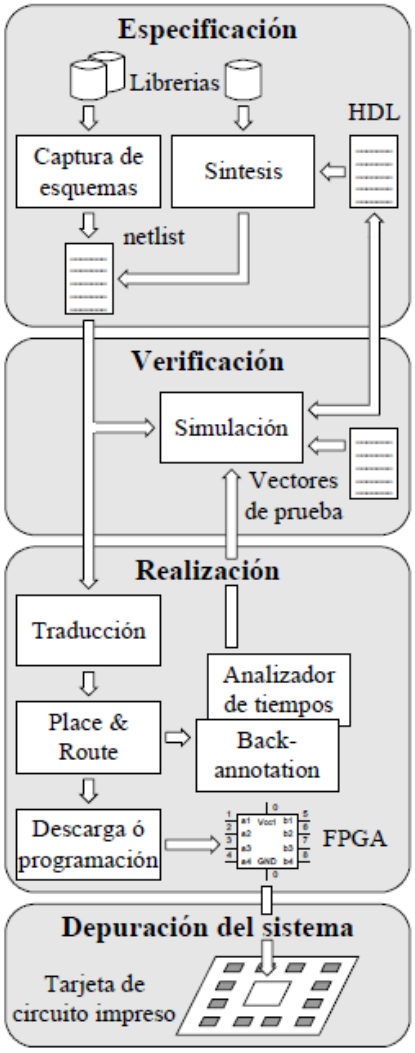


Figura. 3.5. Flujo de Diseño FPGA

Durante la etapa de especificación se realiza el diseño del sistema. Dada la complejidad de los sistemas actuales, se prefiere utilizar la descripción esquemática para representar módulos de jerarquías superiores, en tanto que para describir los módulos de jerarquías inferiores se utiliza lenguajes HDL como VHDL y Verilog. Generalmente, la descripción de los módulos se almacena en librerías para facilitar su reutilización en diseños posteriores. De manera que, cuando se diseña un sistema se utiliza dichas librerías y se describen sólo los nuevos módulos, lo que facilita el proceso.

La descripción es procesada por la herramienta de síntesis para obtener la representación del comportamiento al nivel de componentes electrónicos, mientras que la síntesis es el proceso de adaptación de la lógica de diseño a los recursos lógicos disponibles en el chip. En la síntesis se parte de la especificación de entrada con alto grado de abstracción y de las restricciones de área y tiempo definidas por el diseñador para obtener una descripción menos abstracta, más enfocada puntualmente al dispositivo programable. En el proceso de síntesis se comprueba la sintaxis del código y se analiza la jerarquía del diseño para asegurar una implementación sobre la arquitectura seleccionada. (Mora, 2008)

El proceso básico de verificación consiste en simular el sistema mediante el uso de un programa de software que auxilia al diseñador a comprobar el cumplimiento de las especificaciones del diseño. El simulador recibe un Netlist (archivo de texto equivalente al circuito) y un banco de vectores de prueba y genera las salidas que el diseño produce bajo estos patrones de estímulo. En este punto del flujo de diseño, sólo se puede realizar una simulación funcional en la cual se verifica la operación correcta de la descripción HDL. Si los resultados de salida no son correctos, se debe modificar la descripción del diseño y repetir el ciclo hasta que la funcionalidad sea satisfactoria.

La etapa de realización tiene como propósito obtener un dispositivo FPGA programado con la operación lógica que describe el Netlist. Para lograrlo se requiere de una secuencia de procesos detallados a continuación:

- Traducción (Translate)
- Mapeo (Map)
- Emplazamiento (Place)
- Ruteado (Route)
- Programación del dispositivo (Programming)

En la *Traducción* se integran los programas para importar el Netlist y fusionarlo con las restricciones del diseño definidas por el usuario, para obtener un archivo que describe el diseño lógico a partir de primitivas. En el *Mapeo*, el diseño lógico se ajusta a los recursos disponibles del dispositivo, relacionando los elementos lógicos del diseño con los recursos físicos del chip. El *Emplazamiento* es el proceso de asignar los módulos o bloques lógicos del diseño a elementos específicos de la FPGA. En el *Rutado* se interconectan los elementos lógicos seleccionados para que cumplan con la función lógica del diseño. La *Programación* consiste en descargar a la FPGA la información de configuración que define la funcionalidad diseñada.

En este punto del ciclo de diseño, el dispositivo puede estar ya trabajando, pero todavía es necesario depurar su funcionamiento dentro del sistema que lo contiene y bajo las condiciones del entorno donde trabajará. Este es el momento de resolver situaciones como funcionalidades no consideradas, operaciones incorrectas o requerimientos de Entrada/Salida fuera de norma. La ventaja de la tecnología FPGA es que se puede regresar a las etapas iniciales del ciclo de desarrollo para resolver este tipo de anomalías, sin perjudicar significativamente los planes de tiempos y costos del proyecto. (Mora, 2008)

3.1.3.4. Etapas de co-diseño en FPGA

El flujo de diseño utilizado para el desarrollo del sistema basado en co-diseño en un FPGA, consta de cinco etapas clásicas (Ver Figura. 3.6):

- Partición hardware / software.
- Descripción hardware y Programación de software.
- Cosimulación.
- Síntesis e Implementación.
- Programación de la FPGA.

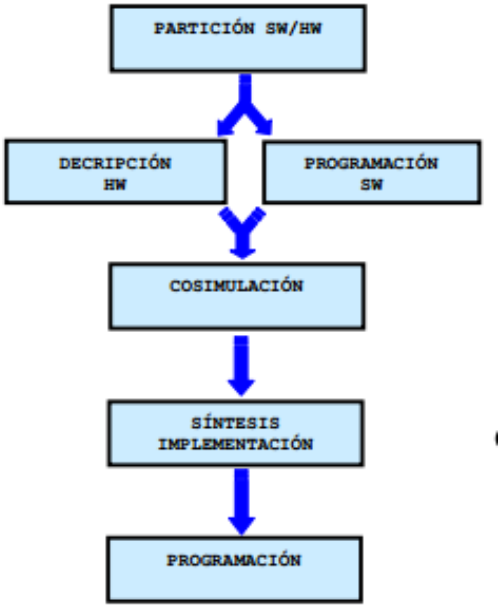


Figura. 3.6. Etapas de Co-diseño en FPGA

La primera etapa es la partición hardware/software en donde se decide qué tareas van a ser ejecutadas por el o los microprocesadores de propósito general y cuáles van a ser implementadas en el hardware específico. Los principales parámetros a considerar en la etapa de partición hardware/software son la velocidad, el área y el costo de desarrollo. Por lo general, las tareas más críticas son implementadas en el hardware específico, mientras que el resto de tareas de menor esfuerzo de procesamiento, son programadas para ser ejecutadas por el microprocesador. De tal manera que el hardware diseñado actúa como coprocesador.

Una vez finalizada la partición hardware/software comienza la etapa de descripción de hardware y programación de software. En esta etapa se describe el hardware, se parametriza si fuese necesario el microprocesador y se programan las tareas que éste va a ejecutar. Los diferentes módulos que integran el diseño, incluido el microprocesador, se describen en un lenguaje de descripción de hardware (HDL), mientras que la programación del microprocesador se realiza utilizando lenguaje de alto nivel (ANSI C).

En la etapa de cosimulación se simula el sistema completo, tanto el hardware como el software. Para ello, se necesita una descripción HDL del sistema que incluya el código máquina que debe ejecutar el procesador. Dicha descripción es procesada por las herramientas de co-diseño que se especificarán en la continuación del presente documento.

Finalmente, en las dos últimas etapas del flujo de diseño se realizan los procesos de síntesis e implementación y de programación del FPGA. En el proceso de síntesis lógica e implementación del diseño se genera un archivo de configuración necesario para programar el FPGA, el mismo que toma el nombre de bitstream. El último paso del flujo de diseño consiste en descargar el bitstream en el FPGA.

3.1.4. Herramientas de Co-diseño Embebido

En el mercado se encuentran gran cantidad de herramientas EDA para el diseño con FPGA y VHDL. Algunas de ellas son de uso público o de código abierto, por ejemplo CooCox que utiliza compiladores GNU, otras cubren casi todos los dispositivos del mercado, como las de Mentor Graphics, Exemplar, Synopsys y Synplicity. La mayoría son específicas de las compañías de dispositivos, como las de Actel, Altera, Cypress y Xilinx. Por lo general, las herramientas funcionan sobre computadores personales y algunas están disponibles para estaciones de trabajo.

Cada compañía ha optimizado las herramientas de co-diseño embebido para las arquitecturas de sus dispositivos cubriendo toda la gama de posibilidades de diseño que sus FPGA ofrecen. Por tanto incluyen herramientas específicas para todas las etapas del flujo de diseño. El presente proyecto se ha desarrollado

sobre una plataforma de la empresa Xilinx. Por tanto esta sección se orienta a describir la herramienta de co-diseño de sistemas embebidos de Xilinx, EDK Embedded Development Kit.

3.1.4.1. Embedded Development Kit, EDK

XILINX es una empresa que se ha dedicado a la creación de herramientas EDA para facilitar el diseño, implementación y programación de sus FPGA. El EDK, Embedded Development Kit es una herramienta de diseño compuesta por una serie de aplicaciones que permiten la configuración de la plataforma del hardware mediante el XPS, Xilinx Platform Studio y la creación de un diseño software que incluya las librerías adecuadas y los controladores de dispositivos para construir y cargar una arquitectura en un FPGA de Xilinx, a través del SDK, Software Development Kit.

Figura. 3.7. Embedded Development Kit de Xilinx

Si bien el entorno EDK admite la creación e implementación de diseños, EDK es un componente del ISE, Integrated Software Environment System Editions, el cual admite sintetizar el diseño del hardware del microprocesador para una arquitectura determinada hacia una aplicación específica sobre un FPGA. Las herramientas proporcionadas por el EDK están planteadas para asistir en todas las fases del proceso de co-diseño embebido, tal como se muestra en la siguiente Figura. 3.8.

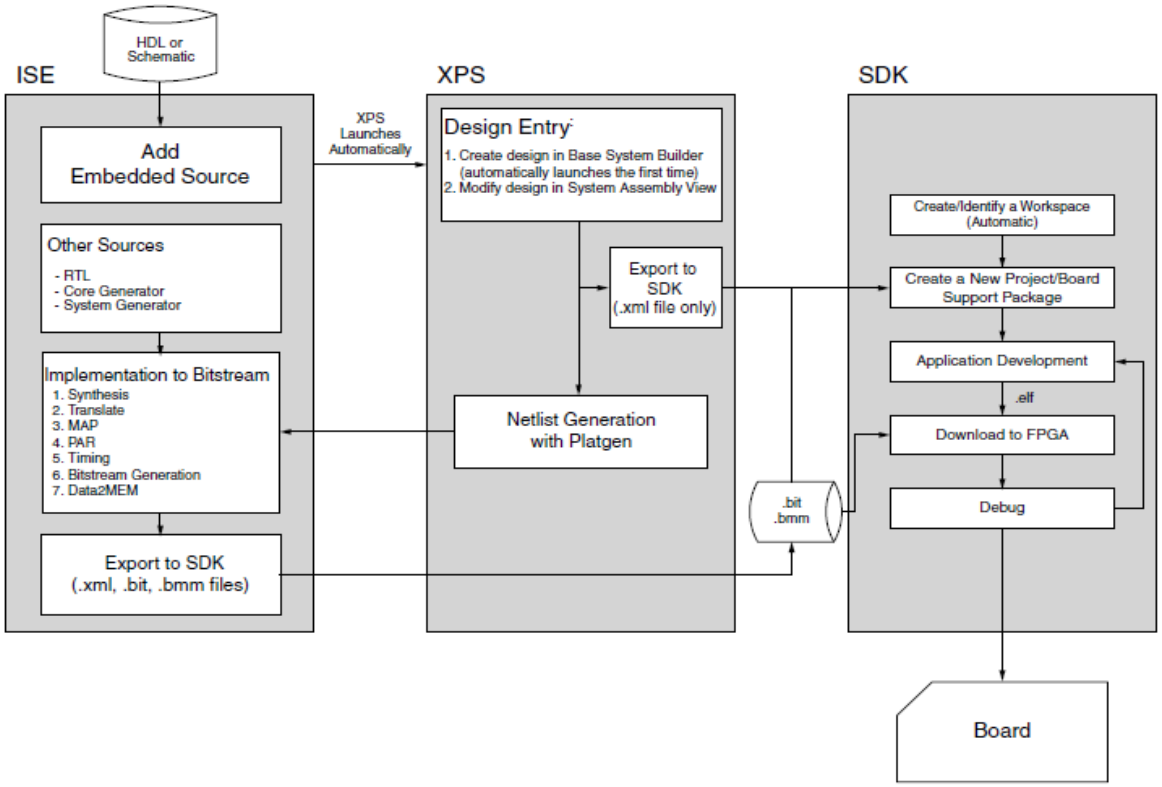


Figura. 3.8. Descripción del proceso de Co-diseño EDK

- **Xilinx Platform Studio, XPS**

El *Xilinx Platform Studio* es una interfaz gráfica a nivel de usuario que posee herramientas de diseño para crear el hardware de sistemas embebidos altamente personalizables. Consta de un amplio catálogo de Bloques de Propiedad Intelectual (IP Cores). Adicionalmente es flexible con la importación y creación de

IP Cores personalizados o desarrollados por el usuario. Por otra parte, permite la creación automática de reportes de diseño.

El resultado de diseño del hardware embebido realizado a través de XPS se almacena en un archivo de descripción de periféricos en Lenguaje de Marcas Extensible, XML eXtended Markup Language que es exportado hacia el SDK para la correspondiente programación de uno o más procesadores. Adicionalmente provee el archivo de flujo de datos o Bitstream File que sirve para programar el hardware sobre el FPGA.

A continuación se detallan cada uno de los componentes de la interfaz gráfica del XPS, expresado en la Figura. 3.9 que corresponden a la versión 13.2. (Xilinx, EDK Concepts, Tools, and Techniques, 2011)

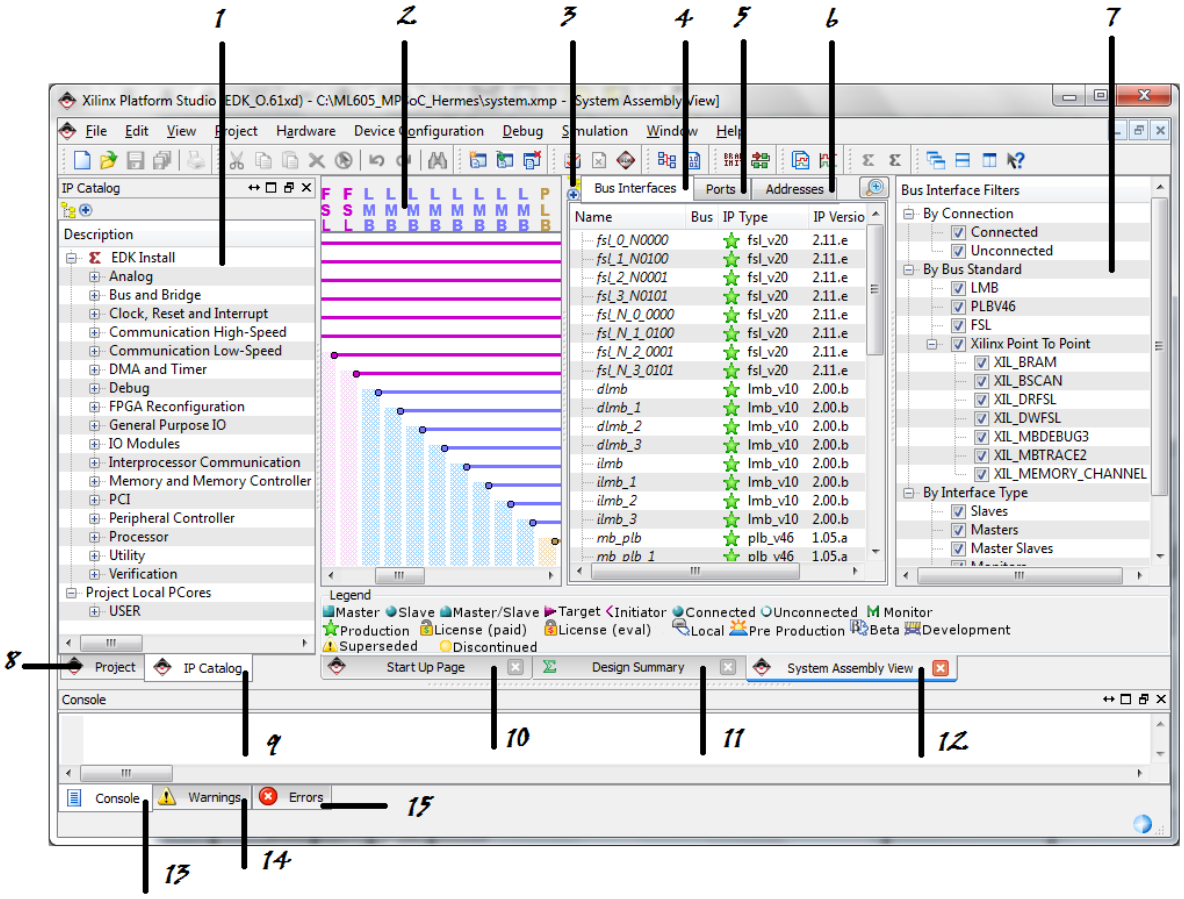


Figura. 3.9. Interfaz gráfica Xilinx Platform Studio

1. Área de Información del Proyecto (Project Information Area): Contiene la información y personalización de los componentes del proyecto. Consta de la pestaña Project e IP Catalog.

2. Panel de Conectividad (Conectivity Panel): Contiene la representación gráfica de las conexiones de hardware de la plataforma. Cabe señalar que una línea vertical representa un bus, y una línea horizontal representa una interfaz de bus a un IP Core.

3. Expansión de Información de Buses e IP-Cores (View Buttons): Permite desplegar toda la información referente a la conectividad entre buses e IP-Cores.

4. Pestaña Interfaz de Buses (Bus Interfaces): Es al Área de Desarrollo del Proyecto, despliega información detallada de los distintos buses presentes en el diseño.

5. Pestaña Puertos (Ports): Muestra información detallada de los distintos puertos presentes en el diseño.

6. Pestaña de Direccionamiento (Adresses): Presenta información detallada de las direcciones de origen y destino de los distintos elementos del sistema.

7. Panel de Filtros (Filters Pane): XPS proporciona filtros que se pueden utilizar para cambiar la forma de ver las Interfaces y Puertos de los buses en la vista del Sistema de Ensamblaje o System Assembly View.

8. Pestaña de Proyecto (Project): en esta sección se encuentra información específica del proyecto, tanto archivos principales, como características propias de configuración (ver Figura. 3.10). (Xilinx, Platform Specification Format Reference Manual, 2011) (Cadena & Mollocana, 2012)

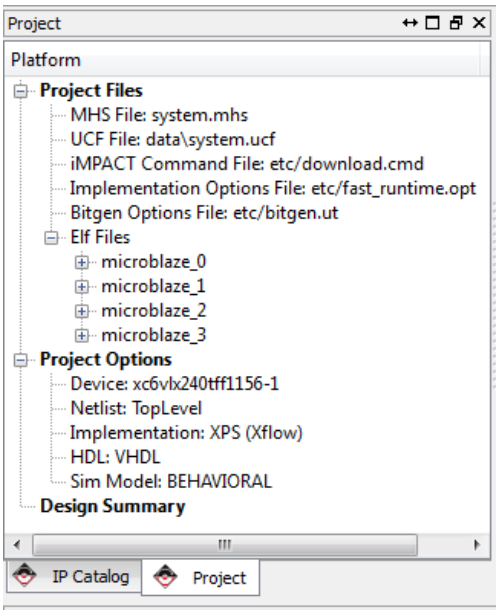


Figura. 3.10. Pestaña Project

A continuación se describe los tipos de archivos del Project (Ver Tabla. 3.4):

NOMBRE	DESCRIPCIÓN
Archivo MHS**, *Multiprocessor Hardware Specification*:** system.mhs	Define los componentes del Hardware y la configuración del Procesador Embebido del Sistema, así como la arquitectura de buses, los periféricos, la conectividad y los espacios de los bancos de direcciones.
Archivo MSS**, *Multiprocessor Software Specification*:** system.mss	Contiene descripciones para la personalización del sistema operativo, librerías y controladores de hardware. El archivo MSS posee una dependencia del archivo MHS.
Archivo UCF**, *User Constraints File*:** system.ucf	Contiene la configuración de pines del FPGA, asociados a las conexiones externas del diseño desarrollado en el XPS.
Archivo XMP**, *Xilinx Microprocessor Project*:** system.xmp	Contiene toda la información del proyecto y permite ser leído por el XPS para presentarlo de forma gráfica en la interfaz de usuario.

Tabla. 3.4. Descripción de Archivos del Project File

9. Pestaña Catalogo de Bloques de Propiedad Intelectual (IP Catalog): Contiene un listado de IP Cores, contenidos en el XPS o importados hacia este, en donde se detalla su nombre, licenciamiento, versión de lanzamiento y estado de funcionamiento. Cada IP Core contiene una hoja técnica y el archivo Microprocessor Peripheral Description, MPD, con información detallada de su uso y programación.

10. Pestaña de la Página de Inicio (Start Up Page): Presenta la página de inicio de XPS configurada por el usuario, sirve de ayuda para consultas web de hojas técnicas, soporte y corrección de errores.

11. Resumen de Diseño (Design Summary): Despliega un resumen detallado del desarrollo y diseño del proyecto, reporte de pines de conexión, reporte de utilización de relojes del FPGA, errores y alertas de configuración.

12. Vista del Sistema de Ensamblaje (System Assembly View): Permite visualizar las pestañas de configuración de buses, puertos y direccionamiento, así como el panel de conectividad.

13. Consola (Console): Permite ingresar comandos de configuración, en modo texto, tanto para configuración de buses, pines, procesadores y puertos. Además muestra reportes específicos de los archivos de información del sistema.

14. Alertas (Warnings): Despliega información sobre posibles alertas o fallas de configuración en alguno de los periféricos o elementos de hardware del sistema.

15. Errores (Errors): Despliega información sobre errores y fallas de configuración de hardware, sean estos correspondientes a la asignación de espacios de memoria, configuración de puertos y conectividad entre buses y periféricos.

- **Software Development Kit, SDK**

Xilinx Software Development Kit, SDK está basado en el estándar de código abierto Eclipse. SDK se incluye en el ISE Suite Design Embedded Edition como programa complementario del XPS, pero también está disponible como producto independiente, proporcionando un entorno para la creación de plataformas de software y aplicaciones específicas para los procesadores embebidos Xilinx.

SDK trabaja con diseños de hardware creados en XPS incorporando herramientas de desarrollo de software. Para el manejo de IP Cores, el SDK contiene un BSP, Board Support Package, que es una colección de bibliotecas, controladores de dispositivos e interrupciones y sistemas de archivos que definen, para cada procesador, la relación de los dispositivos de entrada y salida con el software. Así, de acuerdo a los periféricos y procesadores usados en el diseño de hardware se utilizarán sus correspondientes archivos en el SDK. Cabe señalar, que si fue necesario crear un módulo de hardware propietario en el XPS, será por tanto necesario crear su correspondiente driver en el SDK. (Xilinx, EDK Concepts, Tools, and Techniques, 2011) (Cadena & Mollocana, 2012)

SDK permite utilizar el BSP en dos ambientes de ejecución:

***Standalone*:** ambiente sencillo, en el cual se ejecuta un solo hilo que ofrece funciones básicas entre los dispositivos de entrada y salida, así como el acceso a las características de hardware del procesador.

***Xilkernel*:** ambiente de ejecución que proporciona servicios tales como programación, ejecución de hilos, sincronización, transmisión de mensajes y temporizadores mediante procesamiento ligero.

Por tanto, en el SDK se crea un proyecto que hace referencia al archivo XML realizado en XPS para conocer toda la información de hardware del proyecto sobre el que se va a trabajar. Luego se determina un ambiente BSP y posteriormente se programa el código en lenguaje C o C++ para las actividades que realizarán los distintos procesadores y periféricos del FPGA en una aplicación específica. Finalmente, se compila las aplicaciones, incluidas las bibliotecas y controladores, en un archivo en Formato Ejecutable Vinculado (ELF, Executable Linked Format) para ser ejecutado en cada procesador de la plataforma de hardware.

Así para probar el co-diseño en el FPGA se requiere enviar los archivos bitstream (*.bit) que contiene la arquitectura de hardware junto con el archivo *.elf que contiene el programa de la aplicación.

3.1.4.2. Simulador ISim

El ISim o Simulador del ISE, es un programa para Lenguaje Descriptivo de Hardware (HDL) que se puede ejecutar directamente dentro de la plataforma del ISE. La función del ISim es la de evaluar el comportamiento de los diseños realizados en VHDL, Verilog o de lenguajes mixtos. (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

- **Modo de Inicio**

El ISim se lo llama y ejecuta en un segundo plano dentro del mismo proyecto, luego de la creación de un archivo VHDL Test Bench. Este archivo se crea con todos los parámetros necesarios para una simulación de forma automática, el diseñador también puede manipular dichos parámetros para adecuarlos según le convenga.

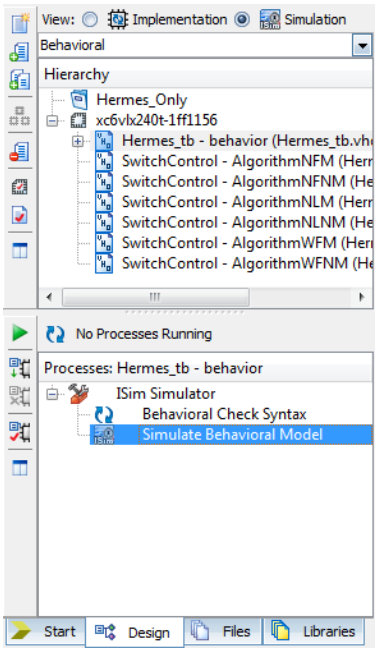


Figura. 3.11. Pestaña de Diseño.

La Figura. 3.11, muestra un archivo VHDL Test Bench creado para la simulación de comportamiento del dispositivo. Esta figura además muestra que en la lista de despliegue debe estar seleccionada la opción Behavioral. Tanto el archivo de simulación como la opción Behavioral, permiten la apertura del simulador siempre y cuando no existan errores en el mencionado archivo.

- **Entorno del ISim**

Siguiendo el ejemplo mostrado en la Figura. 3.11, para poder abrir el simulador se debe dar doble clic en Simulate Behavioral Model en el panel del proceso. Está acción abre la interfaz gráfica que se muestra en la Figura. 3.12. El entorno del ISim contiene una barra de estado, una ventana para graficar señales de onda entre otros paneles que serán descritos a continuación.

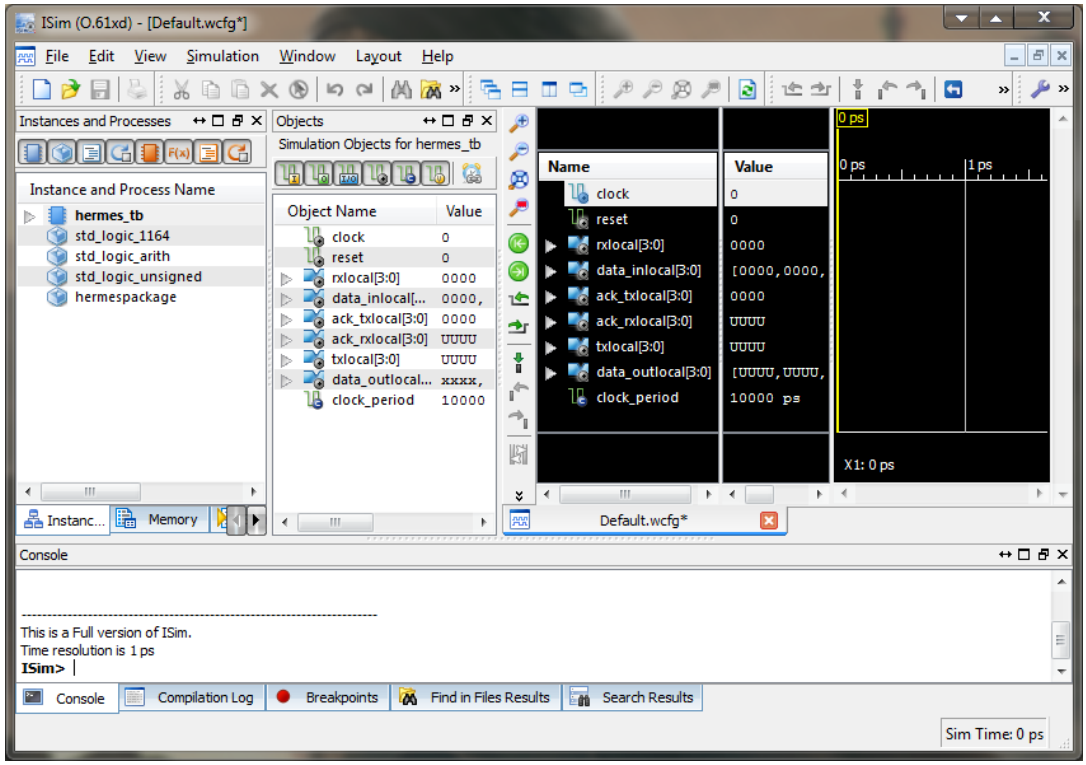


Figura. 3.12. Interfaz Gráfica del Simulador ISim.

Barra de Herramientas, se encuentran todas las opciones necesarias para poner en marcha al simulador. La Figura. 3.13 muestra la barra de herramientas que se encuentra en la parte superior del ISim.

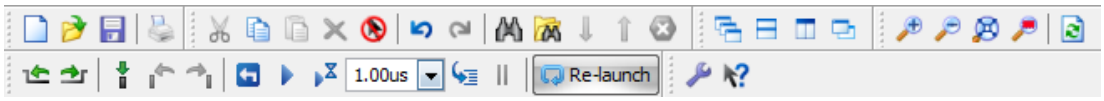


Figura. 3.13. Barra de Herramientas

Con la barra de herramientas se obtiene gran versatilidad para la ejecución de varias instrucciones al dar un solo clic. De esta forma se evita el uso de comandos por el panel de control o el acceso frecuente a una opción por medio del menú. (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

Panel de Procesos, por medio de este panel se puede seleccionar los diferentes módulos que componen la entidad principal del proyecto a través del

archivo VHDL Test Bench. De esta forma se puede seleccionar y crear diferentes grupos de señales que serán graficados en la ventana de señales. (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

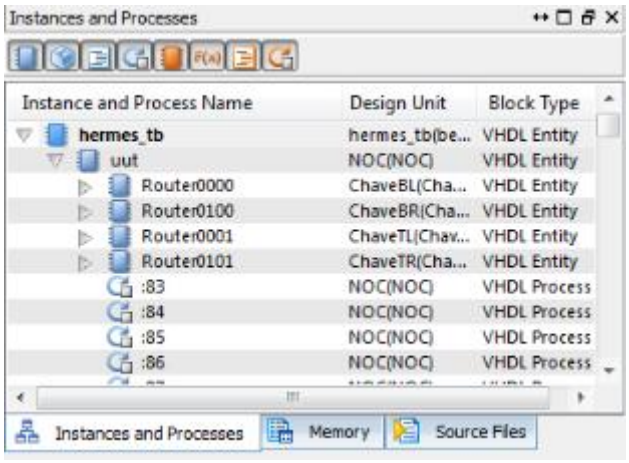


Figura. 3.14. Panel de Procesos

Panel de Archivos Fuente, el panel de la Figura. 3.15, muestra todos los archivos que están vinculados al proyecto. Al tener estos archivos en el panel de archivos fuente, se facilita el acceso inmediato a cualquiera de ellos para poder hacer correcciones en el diseño sin tener que regresar al ISE. (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

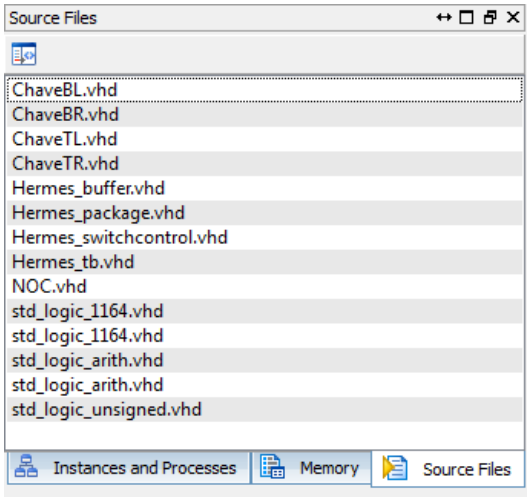


Figura. 3.15. Panel de Archivos Fuente

Panel de Objetos, asocia todos los puertos o pines involucrados con el proyecto en general o con un módulo solamente, dependiendo de lo que se haya seleccionado en el panel de proceso (ver Figura. 3.16). (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

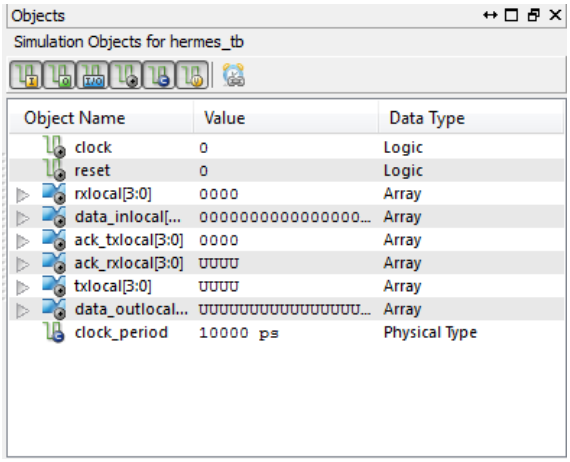


Figura. 3.16. Panel de Objetos.

En el panel de objetos se define los siguientes parámetros:

- *Object Name*: Contiene los nombres de los puertos del proyecto global o de un solo módulo en específico.

- *Value*: Muestra los valores que están presentes en los puertos, o se identifican con una U si no existe valor vinculado. Estos valores pueden ser alterados por el diseñador al hacer clic derecho sobre uno de los puertos.

- *Data Type*: Despliega el tipo de dato correspondiente a cada puerto que bien puede ser de tipo arreglo o lógico.

Ventana de Señales, en la ventana de señales se grafican las formas de onda de los puertos, lo que permite una apreciación del comportamiento del diseño al cual se lo está evaluando (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011). La Figura. 3.17 se abre con un grupo de señales que pertenecen a la entidad. Sin

embargo, dicha ventana puede ser personalizada de acuerdo a las necesidades del diseñador, dando clic derecho sobre la columna Name se puede agregar divisiones, editar los colores de las señales o crear grupos para así priorizar la señal o señales del componente que se está probando.

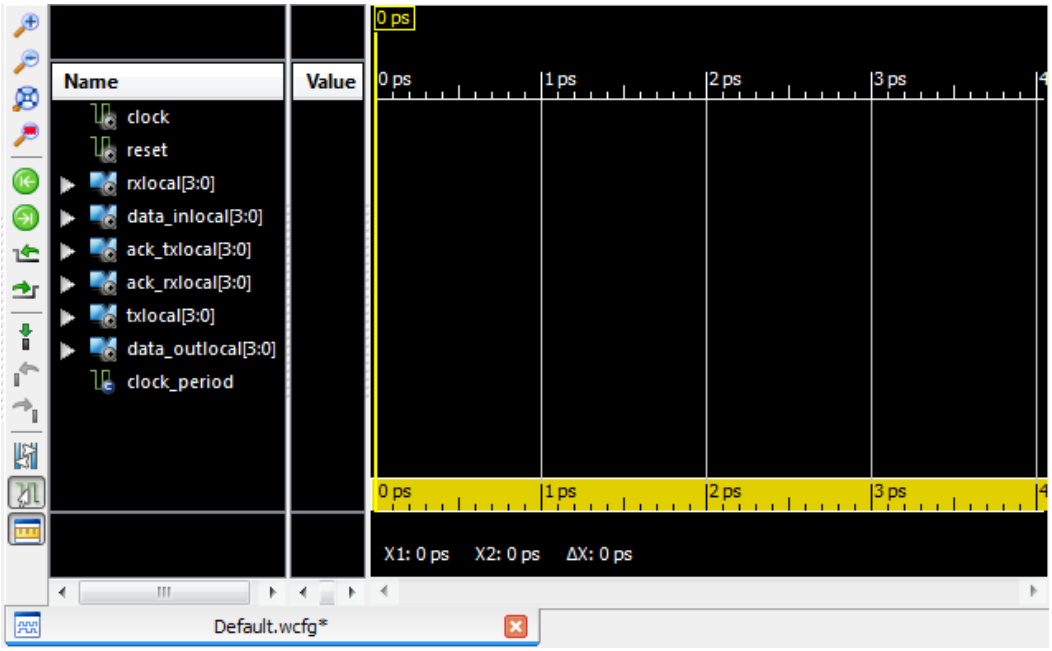


Figura. 3.17. Ventana de Señales

La barra localizada en la parte izquierda de la Figura. 3.17 ofrece otras herramientas que permite incrementar o decrementar la escala de tiempo, para una mejor apreciación de resultados según convenga. También ofrece otra serie de herramientas que ayudan en la evaluación del componente como marcas (X2-X1=ΔX), muy útiles para medir tiempo entre una transición a otra.

Editor de Texto, permite establecer puntos de ruptura sobre la descripción de hardware que se está simulando (Ver Figura. 3.18). Hacer esto permite depurar el diseño si tiene alguna falla, pues la corrección es inmediatamente realizada sin tener que volver a la plataforma del ISE. (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

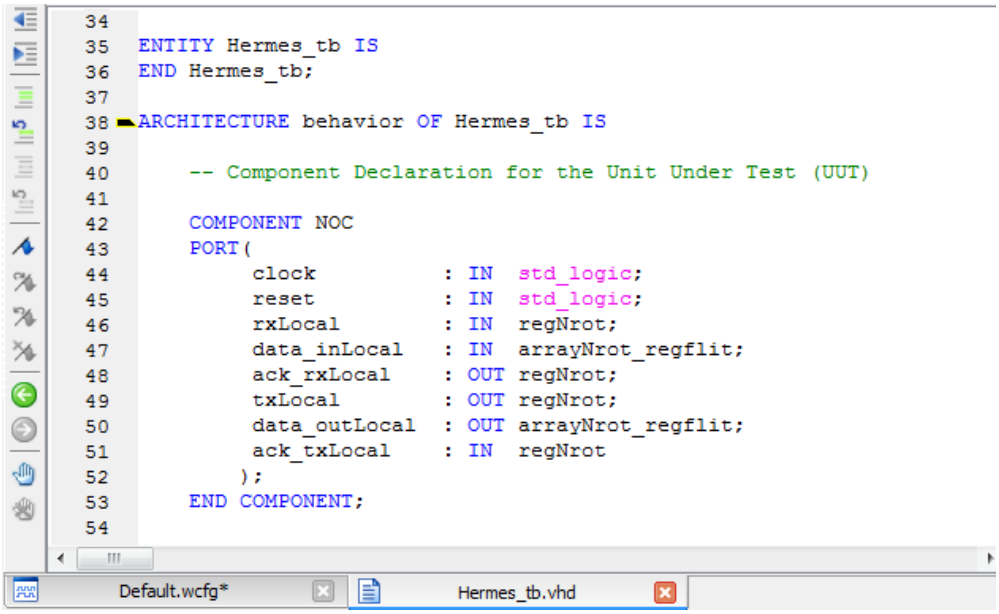


Figura. 3.18. Editor de Texto.

Panel de Punto de Ruptura, La Figura. 3.19 muestra el panel de puntos de ruptura, con el cual se ejemplifica que el mencionado panel despliega todos los puntos de ruptura que hayan sido ubicados en el proyecto. De esta forma se identifica el archivo y la línea de código en donde se encuentra para una mejor navegación dentro del diseño. (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

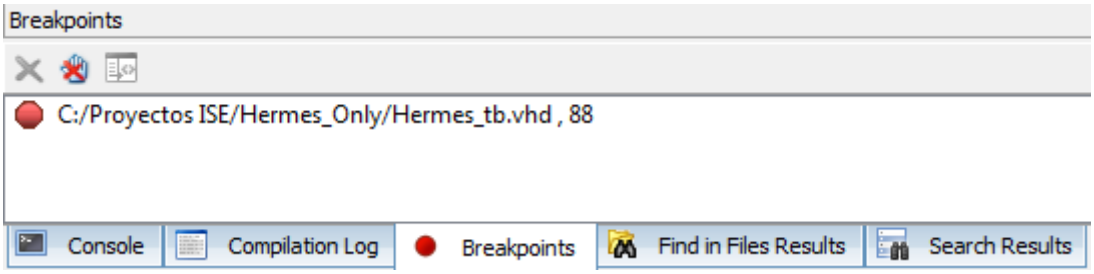


Figura. 3.19. Panel de Puntos de Ruptura

Panel de Control, permite ejecutar comandos, que bien se los puede ejecutar directamente desde la barra de herramientas, así como también despliega mensajes informativos del estado actual de la simulación (Ver Figura. 3.20). (Xilinx, ISE Simulator (ISim), In-Depht Tutorial, 2011)

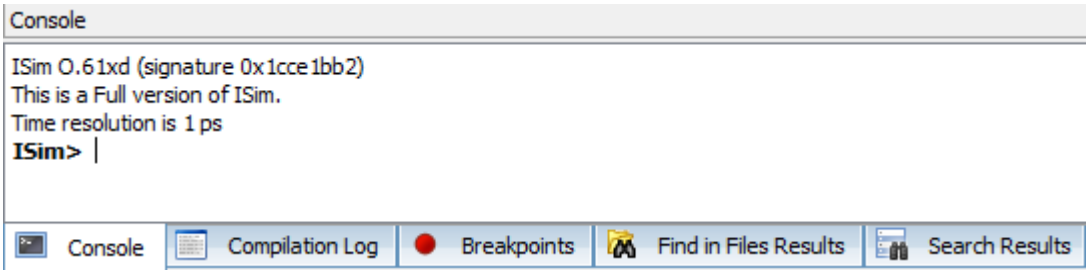


Figura. 3.20. Panel de Control

3.1.4.3. Xilinx Power Analyzer

El XPower Analyzer es una herramienta Xilinx que realiza un análisis de estimación de potencia posterior a la implementación de una aplicación. Es la herramienta más precisa puesto que puede leer desde la base de datos del diseño, los recursos de acceso utilizados. XPA también permite ajustar la configuración del entorno y la tasa de conmutación del diseño para que se pueda evaluar el consumo de energía térmica. Ver Figura. 3.21.

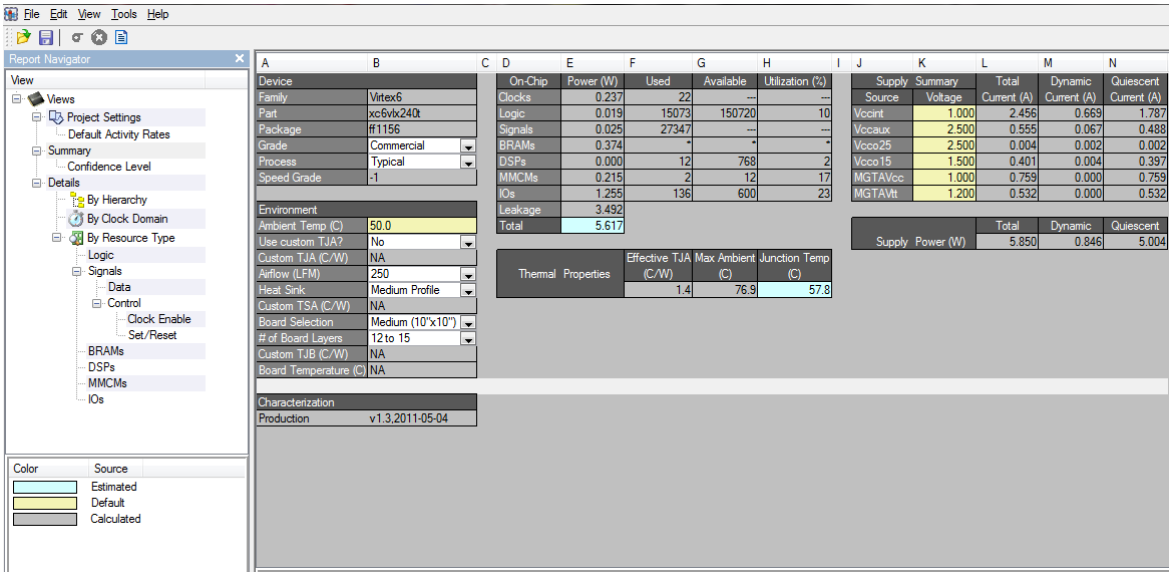


Figura. 3.21. Plataforma Xilinx Power Analizer

En los últimos años, los retos de implementación de nuevas tecnologías embebidas en un chip han perfeccionado las capacidades de los desarrolladores, enfocándose en la reducción del time-to-market, disminución de costos y mejoras

significativas de desempeño. La cobertura de dichas necesidades, da como resultado un incremento en el número de transistores para la implementación de diferentes circuitos lógicos, lo cual ha conducido a un aumento dramático en el consumo de energía de las nuevas tendencias de arquitecturas implementadas en FPGA. (Anderson)

En consecuencia, la disipación de potencia en un FPGA se está convirtiendo en una prioridad de consideración en la etapa de diseño, junto con los objetivos tradicionales de funcionamiento óptimo del circuito, velocidad de respuesta y área del chip utilizada. De hecho, la potencia se ha citado como un factor limitante en la capacidad de los FPGAs para reemplazar a los ASICs. (Anderson)

- **Fuentes de Alimentación Dentro de un FPGA**

Las FPGA modernas integran una amplia gama de lógica personalizada, capacidades de memoria, procesadores, bloques de procesamiento de datos y comunicación. Cada uno de estos componentes necesita fuentes de alimentación y requisitos específicos de disipación de calor, esto origina una necesidad de trabajo a diferentes niveles de tensión para un mayor rendimiento, preservando una alta inmunidad a efectos de ruido y señales parasitas. (Xilinx, Power Methodology, 2011) Ver Tabla. 3.5.

Fuente de Alimentación	Recursos a Energizar
Vccint	Todos los Bloques de Lógica Configurable CLB Recursos de Enrutamiento Bloques de RAM/FIFO Bloques DSP Elementos lógicos de entrada y salida Elementos Ethernet MAC Administradores de Reloj DCM, PLL
Vccaux	Administradores de Reloj MMCM IODELAY Todos los Buffers de salida Buffers de entrada diferenciales Voltajes de Referencia para entradas estándar
Vcco	Algunos Buffers de entrada Controladores Digitales de Impedancia
MGT	Circuitos Transceptores PMA

Tabla. 3.5. Fuentes en una FPGA

- **Potencia Disipada en la FPGA**

La potencia total disipada en un FPGA está dividida en tres tipos de potencia para cada fuente de alimentación:

- Potencia Estática de Dispositivos

La potencia estática de dispositivos, representa la energía necesaria para que el dispositivo funcione y esté disponible para su programación, también es conocida como potencia de fuga del transistor, cuando este está energizado, pero no está configurado. (Xilinx, Power Methodology, 2011)

- Potencia Estática de Diseño

La potencia estática de diseño, representa el consumo adicional de energía cuando el dispositivo está configurado, pero no existe actividad de conmutación. Incluye energía estática en bloques de entrada y salida y gestores de señal de reloj. (Xilinx, Power Methodology, 2011)

- Potencia Dinámica de Diseño

La potencia dinámica de diseño, representa el consumo de energía adicional resultante de la actividad de conmutación de los elementos de la aplicación. Esta energía varía en el tiempo con la actividad del diseño, también depende de los niveles de voltaje utilizados, recursos lógicos y del enrutamiento. (Xilinx, Power Methodology, 2011)

- **Consumo de Potencia en la FPGA**

La potencia total suministrada al FPGA fluye hacia el exterior, por medio de dos rutas específicas:

- Potencia Térmica

Representa la energía consumida internamente dentro del FPGA. Permite la habilitación de elementos del dispositivo, así como su conmutación, estas

acciones generan calor, lo que contribuye a elevar la temperatura de juntura y se requiere proporcionar rutas de disipación hacia el medio ambiente para mantener a los dispositivos en rangos tolerables de funcionamiento.

- Potencia off-chip

Representa la corriente que fluye desde la fuente de energía a través de los pines de alimentación de entrada y salida del FPGA, hacia los distintos elementos de consumo, por lo tanto no contribuyen a elevar la temperatura de juntura. Generalmente es consumida por LEDs, terminales de entrada y salida y buffers. (Xilinx, Power Methodology, 2011)

Las arquitecturas FPGA actuales implementan sistemas complejos con millones de transistores que consumen varios vatios de potencia, por lo cual el diseñador esta consiente que el desarrollo de su sistema requiere de herramientas de estimación en la disipación de energía con respecto al flujo de diseño y evaluación de requisitos térmicos. Estas herramientas permiten equilibrio en las diversas etapas de diseño, reduciendo el esfuerzo de desarrollo, evaluación y costo. Ver Figura. 3.22.

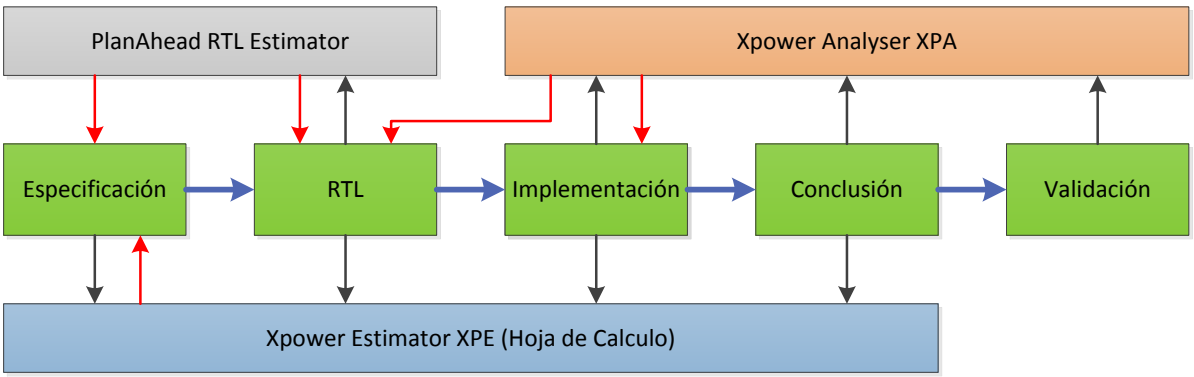
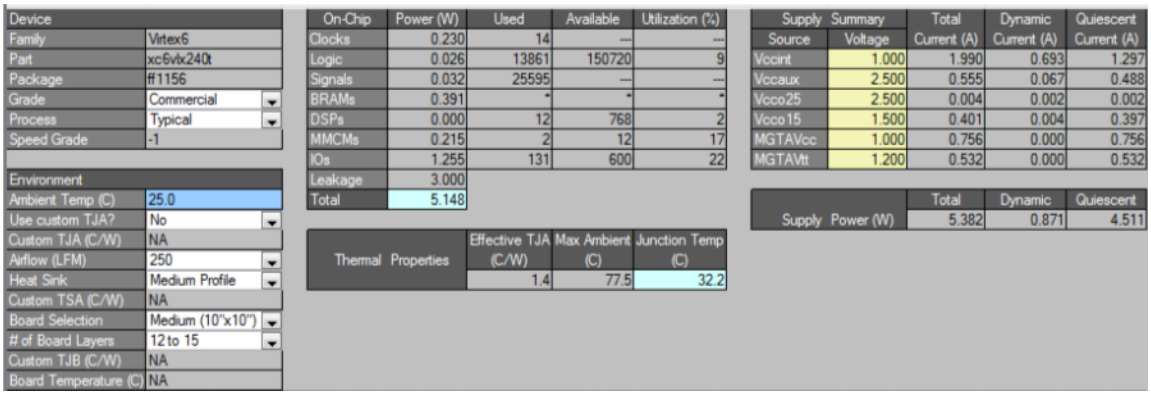


Figura. 3.22. Herramientas de Análisis y Estimación de Potencia en el proceso de diseño

Uno de los factores fundamentales para desarrollar una estimación de potencia recomendable es considerar de forma aproximada los valores de temperatura ambiente y la presencia de flujo de aire en contacto con el FPGA. Ver Figura. 3.23.



Device	
Family	Virtex6
Part	xc6vlx240t
Package	ff1156
Grade	Commercial
Process	Typical
Speed Grade	-1

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250
Heat Sink	Medium Profile
Custom TSA (C/W)	NA
Board Selection	Medium (10"x10")
# of Board Layers	12 to 15
Custom TJB (C/W)	NA
Board Temperature (C)	NA

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.230	14	---	---
Logic	0.026	13861	150720	9
Signals	0.032	25595	---	---
BRAMs	0.391	*	*	*
DSPs	0.000	12	768	2
MMCMs	0.215	2	12	17
IOs	1.255	131	600	22
Leakage	3.000			
Total	5.148			

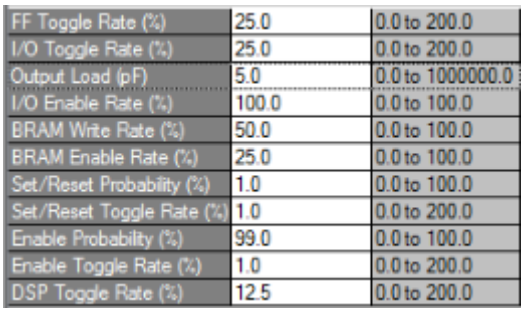
Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	1.4	77.5	32.2

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.000	1.990	0.693	1.297
Vccaux	2.500	0.555	0.067	0.488
Vcco25	2.500	0.004	0.002	0.002
Vcco15	1.500	0.401	0.004	0.397
MGTAVcc	1.000	0.756	0.000	0.756
MGTAVtt	1.200	0.532	0.000	0.532

	Total	Dynamic	Quiescent
Supply Power (W)	5.382	0.871	4.511

Figura. 3.23. Configuración de Condiciones Ambientales del FPGA

La tasa de conmutación es otro de los factores a considerar por el diseñador para determinar con mayor precisión la estimación de potencia de la arquitectura a ser implementada. Ver Figura. 3.24.



FF Toggle Rate (%)	25.0	0.0 to 200.0
I/O Toggle Rate (%)	25.0	0.0 to 200.0
Output Load (pF)	5.0	0.0 to 1000000.0
I/O Enable Rate (%)	100.0	0.0 to 100.0
BRAM Write Rate (%)	50.0	0.0 to 100.0
BRAM Enable Rate (%)	25.0	0.0 to 100.0
Set/Reset Probability (%)	1.0	0.0 to 100.0
Set/Reset Toggle Rate (%)	1.0	0.0 to 200.0
Enable Probability (%)	99.0	0.0 to 100.0
Enable Toggle Rate (%)	1.0	0.0 to 200.0
DSP Toggle Rate (%)	12.5	0.0 to 200.0

Figura. 3.24. Configuración de Estimación de Tasa de Conmutación en XPA

Los valores de configuración de reloj se los debe realizar manualmente en algunos de los casos, dependiendo del IP Core o elemento del FPGA a ser configurado. Ver Figura. 3.25.

ame	Power (W)	Frequency (MHz)	Fanout	Slice Fanout
Clocks				
CLK_S	0.20952	200.0	14698	5357
DDR3_SDRAM/DDR3_SDRAM/mpmc_core_0/gen_v6_ddr3_phy.mpmc_phy_if_0/u_phy_read/u_phy_r...	0.01170	150.0	1004	231
DDR3_SDRAM/DDR3_SDRAM/mpmc_core_0/gen_v6_ddr3_phy.mpmc_phy_if_0/u_phy_read/u_phy_r...	0.00623	150.0	332	80
SysACE_CompactFlash/SysACE_CLK_BUFGP/IBUFG	0.00279	33.3	61	39
microblaze_0_mdm_bus_Dbg_Update	0.00012	0.0	83	43
MMCM_PHASE_CALIBRATION_ML_LUT2_236_ML_NEW_CLK	0.00001	0.0	3	3
DDR3_SDRAM/DDR3_SDRAM/mpmc_core_0/gen_v6_ddr3_phy.mpmc_phy_if_0/u_phy_read/u_phy_r...	0.00067	150.0	20	20
DDR3_SDRAM/DDR3_SDRAM/mpmc_core_0/gen_v6_ddr3_phy.mpmc_phy_if_0/u_phy_read/u_phy_r...	0.00067	150.0	18	18
DDR3_SDRAM/DDR3_SDRAM/mpmc_core_0/gen_v6_ddr3_phy.mpmc_phy_if_0/u_phy_read/u_phy_r...	0.00067	150.0	18	18
DDR3_SDRAM/DDR3_SDRAM/mpmc_core_0/gen_v6_ddr3_phy.mpmc_phy_if_0/u_phy_read/u_phy_r...	0.00067	150.0	18	18
MMCM_PHASE_CALIBRATION_ML_LUT2_228_ML_NEW_CLK	0.00000	0.0	3	3
clock_generator_0/clock_generator_0/MMCM0_INST/MMCM_ADV_inst_ML_NEW_I1	0.00000	0.0	3	3
clock_generator_0/clock_generator_0/MMCM2_INST/MMCM_ADV_inst_ML_NEW_I1	0.00000	0.0	3	3
mdm_0/mdm_0/drck_i	0.00000	0.0	633	207
Total	0.23305			

Figura. 3.25. Configuración de Frecuencias de Operación en XPA

3.1.4.4. Xilinx Power Estimator

XPower Estimator, XPE, es una herramienta propia de Xilinx, desarrollada en una hoja de cálculo en formato Excel, ver Figura. 3.26, que suele utilizarse en diferentes etapas del diseño y la ejecución de un proyecto. Esta herramienta proporciona asistencia en la evaluación de la arquitectura a implementarse, selección de los componentes idóneos de alimentación y componentes térmicos de disipación de energía para cada aplicación específica. (Xilinx, Power Methodology, 2011)

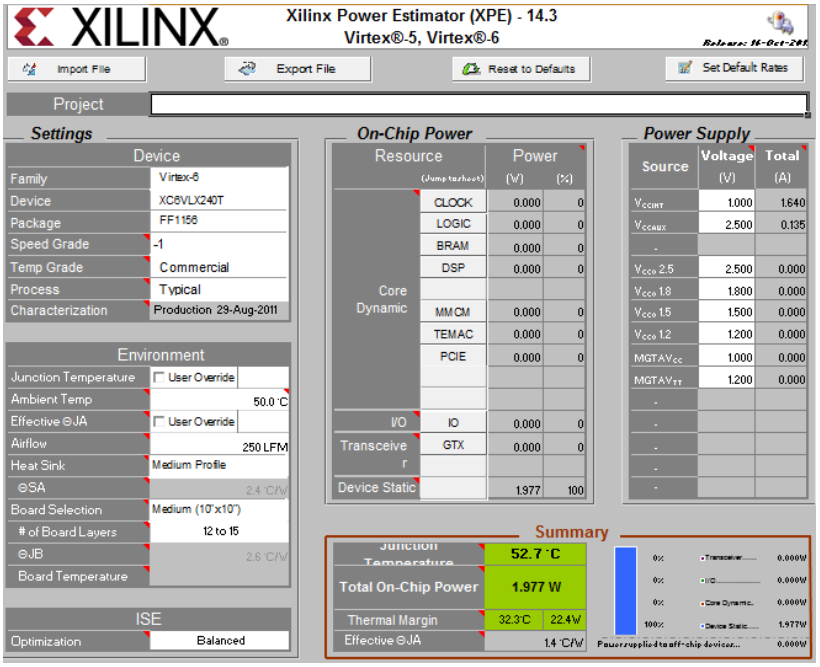


Figura. 3.26. Hoja de Excel, Xilinx Power Analizer

El XPE considera para su análisis, diversos recursos como:

- Tasa de Conmutación o Toggle Rate
- Carga en entradas y salidas
- Factores propios de cada FPGA, siendo estos extraídos por varios métodos de medición, simulación y extrapolación, proporcionados por el fabricante en cada encapsulado.

La precisión del XPE en su análisis de estimación de potencia, depende de dos factores fundamentales:

1. Utilización de los elementos del FPGA, configuración de relojes, habilitadores y Tasa de conmutación.
2. Características específicas de cada empaquetado del FPGA.

La estimación de potencia para los dispositivos programables FPGA es un proceso complejo, ya que es altamente dependiente de la cantidad de lógica en el diseño y la configuración de esta lógica. Para obtener estimaciones exactas, el proceso de estimación de potencia requiere valores precisos del funcionamiento de la aplicación, tales como la utilización de recursos, velocidades de reloj, y un análisis lo más aproximado posible de las tasas de conmutación de los diversos componentes a implementarse.

El estudio de estimación de potencia que realiza el XPE, esta modelado en una investigación de sensibilidad de consumo de energía, frente a las variaciones de temperatura y tensión de cada uno de los componentes del FPGA, además de un factor externo, como es la relación de temperatura del medio ambiente y la corriente de aire a la que estará expuesta el encapsulado. (Xilinx, XPower Estimator User Guide, 2012)

XPE presenta varios escenarios para analizar la distribución de energía en un FPGA.

- **Energía por fuente de voltaje**

Representa la energía requerida por cada fuente de tensión, esta información es necesaria para el dimensionamiento de fuentes de suministro de potencia y reguladores de poder.

- **Energía por recursos lógicos del usuario**

Para cada tipo de lógica utilizada en una aplicación específica, XPE analiza la cantidad de potencia esperada. Permitiendo reestructurar la arquitectura y los recursos utilizados, con la finalidad de mantenerse dentro del presupuesto de potencia asignado. (Xilinx, XPower Estimator User Guide, 2012)

- **Energía térmica**

XPE permite introducir los ajustes del dispositivo al medio ambiente y las características térmicas para cada aplicación programada. Pudiendo evaluarse la necesidad de refrigeración activa o pasiva, según sea el caso de diseño. (Xilinx, XPower Estimator User Guide, 2012)

3.1.4.4.1. Tasa de Conmutación

La tasa de conmutación corresponde a la información de mayor relevancia en el análisis de estimación de potencia, puesto que depende específicamente de cada aplicación. Por lo cual se recomienda modelarla de acuerdo al criterio propio de la programación de la aplicación.

- Para una aplicación síncrona, la tasa de conmutación refleja cuan frecuente, una salida cambia en relación a una entrada en un ciclo de reloj, y puede ser modelada en una escala porcentual de 0% - 100%. En donde 100% determina que la salida cambia en cada flanco de reloj. Por ejemplo, en un contador binario, el bit menos significativo, deberá tener una tasa de conmutación de 100%, puesto que cambia en cada flanco de reloj, en cambio, en el siguiente bit, la tasa de conmutación, viene dada en un 50%, puesto que depende del cambio del bit más significativo. (Xilinx, Power Methodology, 2011)

- Para una aplicación asíncrona o una máquina de estado, la configuración de la tasa de conmutación no es fácil de predecir. Por lo cual un método recomendado, es la separación de las diferentes secciones del diseño en base a su funcionalidad y estimar las tasas de conmutación para cada uno de los sub bloques. La mayoría de los diseños de lógica intensiva, trabajan en 12,5% como configuración por defecto de XPE. Para obtener una estimación del peor escenario, la tasa de conmutación podría ser configurada en 20%. Aplicaciones que incluyen múltiples operaciones aritméticas, en la mayoría de los casos se configuran a 50%, representando el peor de los escenarios. (Xilinx, Power Methodology, 2011)

3.1.4.5. Planahead RTL Power Estimator

El software PlanAhead realiza un análisis de estimación de potencia desde el punto de vista de diseño a nivel RTL. Mediante esta herramienta se puede

especificar el entorno operativo del FPGA, las propiedades de entradas y salidas, y las tasas de actividad para el diseño desde la interacción de un entorno gráfico. PlanAhead lee el código HDL para estimar los recursos de diseño necesarios e informar la potencia estimada a partir de un análisis estadístico de la actividad de cada recurso (Xilinx, PlanAhead User Guide, 2011). Ver Figura. 3.27.

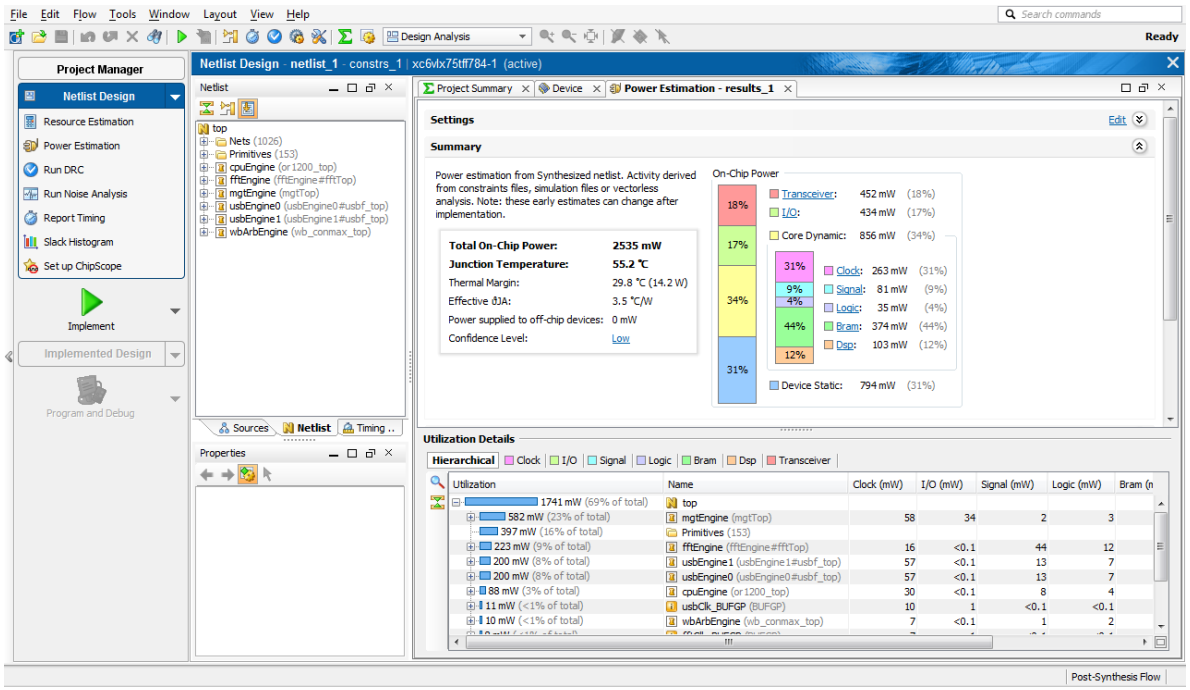


Figura. 3.27. Planahead RTL Power Estimator

3.2. LENGUAJES DE DESCRIPCIÓN DE HARDWARE

En la actualidad la mayoría de IP Cores son del tipo Soft Core, de allí la importancia de conocer los lenguajes de descripción de hardware (HDL, Hardware Description Language) Verilog y VHDL. El objetivo de un HDL es describir un circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere y ensamble un circuito que pueda ser implementado físicamente. El lenguaje HDL permite documentar las interconexiones y el comportamiento del circuito electrónico sin la necesidad de utilizar diagramas esquemáticos.

A continuación se especifican algunas de las características más relevantes de los lenguajes HDL:

- Permite definir un amplio rango de niveles de jerarquía.
- Permite combinar la descripción estructural con la descripción del comportamiento de un circuito electrónico.
- Define una sintaxis independiente del nivel de descripción.
- Permite verificar el funcionamiento del sistema dentro del proceso de diseño sin necesidad de implementar el circuito.
- Reduce el tiempo de diseño y la cantidad de errores producidos por el armado del circuito.
- Puede ser utilizados en cualquier tipo de dispositivo programable capaz de soportar la densidad del diseño.
- Posee compatibilidad con un gran número de herramientas de diseño electrónico, simuladores y generadores de vectores de prueba.
- Posee facilidad de comprensión y lectura que simplifica la documentación.

3.2.1. Lenguaje de Descripción Verilog

Verilog es un lenguaje de descripción de hardware utilizado para describir sistemas digitales, tales como procesadores, memorias o un simple flip-flop. Verilog existe desde el año 1984 y luego de varios años de uso se ha estandarizado y su última versión es del año 2001. (Silva, 2010)

Verilog permite modelar sistemas digitales reales que funcionan en forma paralela mediante la descripción de código concurrente. Diferenciándolo de la ejecución secuencial típica de un sistema computacional. En Verilog un sistema digital está compuesto por la interconexión de módulos. Cada módulo dispone de una serie de entradas y salidas a través de las que se interconecta con otros módulos.

```
module <nombre del modulo> ( <señales> );
<declaraciones de señales>
<funcionalidad del módulo>
endmodule
```

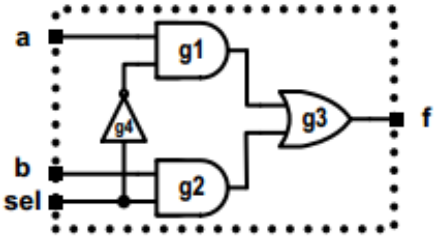
Verilog es un lenguaje similar a C y permite realizar descripciones abstractas de alto nivel, así como representaciones de sistemas digitales en base a compuertas e incluso en base a transistores. (Silva, 2010)

3.2.1.1. Niveles de Abstracción en Verilog

Verilog soporta el diseño de un circuito a diferentes niveles de abstracción, entre los que se destacan: (Nuñez)

- Nivel de Compuerta
- Nivel de Transferencia de Registros
- Nivel de Comportamiento

- **Nivel de Compuerta:** Corresponde a una descripción a bajo nivel de diseño, también denominada modelo estructural. En este nivel de abstracción, el diseñador describe el diseño mediante el uso de primitivas lógicas (AND, OR, NOT, etc.) y conexiones lógicas. Todas las señales son discretas, pudiendo tomar únicamente valores entre 0 y 1. En la Figura. 3.28 se describe un multiplexor a nivel de compuertas. (Nuñez)

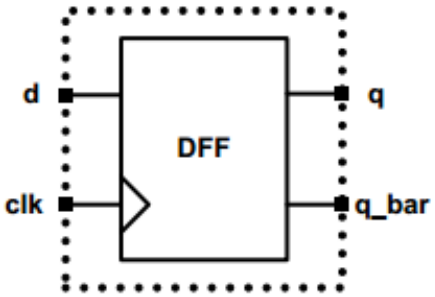



```
module mux(f, a, b, sel);
input           a, b, sel;
output          f;

and     #5      g1(f1, a, nsel),
                g2(f2, b, sel);
or      #5      g3(f, f1, f2);
not             g4(nsel, sel);
endmodule
```

Figura. 3.28. Multiplexor descrito a nivel de compuertas en Verilog

- **Nivel de Transferencia de Registros o Nivel RTL:** Los diseños descritos a nivel de RTL especifican las características de un circuito mediante operaciones y la transferencia de datos entre registros. Las operaciones se realizan en instantes determinados, mediante el uso de especificaciones de tiempo. La Figura. 3.29 corresponde a la descripción a nivel RTL de un flip-flop. (Nuñez)




```
module ff(d, clk, q, q_bar);
input           d, clk;
output          q, q_bar;
reg             q, q_bar;
always @(posedge clk)
begin
q       <= #1 d;        // Retardo de 1 unidad
q_bar   <= #1 !d;       // Retardo de 1 unidad
end
endmodule
```

Figura. 3.29. Flip-flop descrito en nivel RTL en Verilog

- **Nivel de Comportamiento o Behavioral Level:** La principal característica de este nivel de abstracción es su total independencia de la estructura de diseño. El diseñador, más que definir la estructura, define el comportamiento del diseño. En este nivel, el diseño se define mediante algoritmos en paralelo. Cada uno de estos algoritmos consiste en un conjunto de instrucciones que se ejecutan de forma secuencial. (Nuñez)

3.2.2. Lenguaje de descripción VHDL

VHDL es un acrónimo de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, es decir, es un lenguaje de programación utilizado para la descripción de hardware en circuitos integrados de alta velocidad como PLD, CLPD, PAL y FPGA entre otros. En otras palabras, VHDL es un lenguaje de programación que permite describir la estructura, flujo de datos, y comportamiento de dispositivos lógicos, tanto combinacionales como secuenciales. (Carpio, 1997)

VHDL es reconocido como un estándar de los lenguajes HDL por el Instituto de Ingenieros en Electricidad y Electrónica, IEEE en 1993 y en la actualidad es un estándar de la industria para la descripción, modelado y síntesis de circuitos digitales. (Carpio, 1997)

El lenguaje VHDL ha sido desarrollado con el propósito de especificar y documentar circuitos y sistemas digitales utilizando un lenguaje formal. En la práctica se ha convertido en el HDL de referencia para realizar modelos sintetizables automáticamente. El presente proyecto ha sido desarrollado en base a VHDL por lo cual a continuación se especifican las principales características de dicho lenguaje:

- Permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de compuertas.
- Permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.
- Permite que la descripción textual del hardware se materialice en una representación del mismo, siendo totalmente utilizable por herramientas auxiliares tales como simuladores y sintetizadores lógicos, compiladores de

silicio, simuladores de tiempo, simuladores de cobertura de fallos y herramientas de diseño físico.
- Permite la especificación de la funcionalidad de un circuito.
- Permite la simulación del circuito antes de ser implementado.
- Posee adaptabilidad a distintas metodologías de diseño.
- Es independiente de la tecnología por lo cual facilita su actualización y adaptación de los diseños a los avances de la tecnología en cada momento.
- Posee la propiedad de reutilización en otros sistemas. (M., 2009)

Estructura de un Diseño VHDL

Para la descripción de cualquier circuito (en código combinacional o secuencial) se hace necesario cumplir con dos partes que son: la declaración de la entidad y la declaración de la arquitectura. Cabe señalar, que una unidad de hardware se visualiza como una “caja negra”, dicha caja negra posee una interfaz completamente definida y su interior oculto. Para definir el funcionamiento de la caja negra se puede utilizar algoritmos, estructuras, tablas de verdad, diagramas de flujo o diagramas de estado.

En VHDL la caja negra se denomina entidad y describe las entradas y salidas del diseño, mientras que para describir su funcionamiento se asocia una implementación que se denomina arquitectura y describe el contenido del diseño. (M., 2009) (Figura. 3.30)

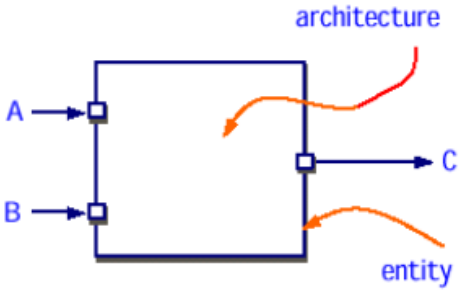


Figura. 3.30. Definición de las partes de una Caja Negra

En la *Entidad*, se declaran las entradas y salidas que tendrá el circuito. La declaración de la entidad se realiza mediante la palabra clave ENTITY y tiene la siguiente topología:

```
ENTITY nombre del dispositivo IS
PORT (
   Puerto de Entrada       : IN  tipo de dato;
   Puerto de Salida        : OUT tipo de dato);
END nombre del dispositivo;
```

Y en la *Arquitectura*, se describe el funcionamiento interno del circuito, es decir que en esta parte se declara el comportamiento de la salida en función de la entrada (Anderson). Su declaración es de la siguiente forma:

```
ARCHITECTURE nombre de arquitectura OF nombre del dispositivo IS
   {Área de declaración}
BEGIN
   {Área de descripción}
END nombre de arquitectura;
```

En la estructura anterior se observan dos áreas sobre las cuales se puede introducir código. En la primera área se crea un grupo de señales u objetos

internos para operaciones auxiliares, si es que el circuito lo necesita. Y en la segunda área se asocia lo realizado en el área de declaración junto con las entradas y salidas de la entidad, para obtener el comportamiento deseado del circuito. (M., 2009)

A fin de una mejor comprensión se sugiere revisar el Anexo 1 que contiene el desarrollo de la teoría de circuitos secuenciales síncronos y máquinas de estado finito.

Descripción de Máquinas de Estado Finito en VHDL

Para desarrollar la descripción de un Máquina de Estado Finito (FSM, Finite State Machine) en VHDL, es necesario definir las variables externas y las salidas de control, es decir declara la entidad propiamente dicha de la caja negra que se desea diseñar. A continuación se desarrollará un ejemplo de un diseño de FSM para dos entradas (variables externas) y dos salidas (salidas de control) para una mejor comprensión del método empleado para crear una FSM.

Figura. 3.31. Declaración de entidad (caja negra) para una FSM

La Figura. 3.31 muestra la caja negra del ejemplo que se pretende diseñar. Visto en VHDL el código para esta etapa de descripción se expresa a continuación:

```
ENTITY FMS IS
PORT (
   ent          : IN  std_logic_vector(1 downto 0);
   sal          : OUT std_logic_vector(1 downto 0);
   clk, rst     : IN  std_logic);
END FMS;
```

En la entidad se observa la declaración de la entidad como *ent*, para las entradas *x* e *y*, y sal para las salidas *w* y *z*, en forma de vector. Una vez definida la estructura de la caja negra lo siguiente que se debe realizar es la declaración de la función interna. En la Figura se puede aprecia las partes que componen la caja negra de la figura, en donde se puede apreciar los bloques que se tienen que describir en VHDL.

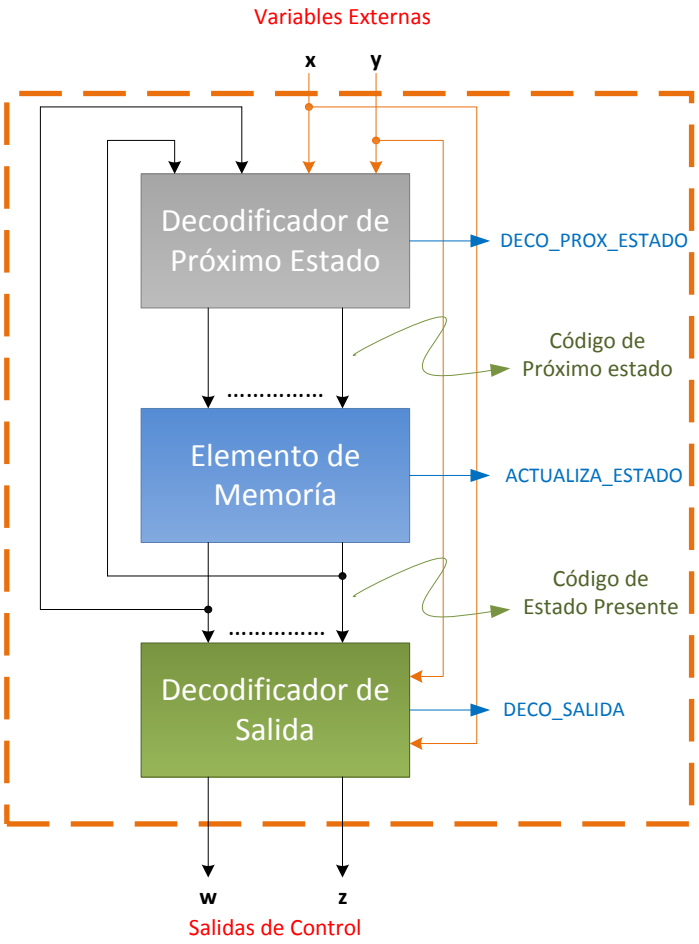


Figura. 3.32 Estructura interna de una FSM

Siguiendo el ejemplo de la FSM a continuación se muestra la descripción del hardware realizado en VHDL. En primera instancia se describen las señales y tipos de datos que asocian a las diferentes estructuras relacionadas que cumplirán una determinada funcionalidad.

```
ARCHITECTURE behavioral OF FSM IS
   TYPE estados IS (a, b, c, d, e, f, g);
   SIGNAL Est_actual, Prox_estado: estados;
END FSM;
BEGIN
```

Posterior a la descripción de señales, se continua con la descripción de las instrucciones concurrentes, sentencias condicionales y expresiones secuenciales que determinan la descripción del funcionamiento de la entidad. En el cuerpo de la arquitectura se modela el comportamiento del circuito con asignaciones, instanciaciones y la definición de un proceso, definido como *PROCESS*. Un Process define el comportamiento de un circuito cuyo estado puede variar cuando cambia alguna de las señales en su lista de sensibilidad, las instrucciones dentro del process se ejecutan secuencialmente, una detrás de otra, pero sin dar lugar a que avance el tiempo durante su ejecución. El tiempo solo avanza cuando se llega al final del proceso. Una arquitectura puede tener tantos procesos como se requiera, y todos se van a ejecutar en paralelo.

Retomando el ejemplo de diseño de FSM en VHDL a continuación se describe su funcionamiento interno mediante la descripción de tres bloques funcionales referentes a la figura 3.15, los mismos que son descritos a través de tres process. El primer process denominado ACTUALIZA_ESTADO, cumple con la sincronización entre el estado presente y el próximo estado en base al tipo de flanco de reloj (sea ascendente o descendente).

```
ACTUALIZA_ESTADO: PROCESS (clk, rst);
  BEGIN
        If rst = '1' then
              Est_actual <= 'a';
        Elsif clk'event and clk = '1' then
              Est_actual <= Prox_estado;
        End if;
  END PROCESS ACTUALIZA_ESTADO;
```

El DECO_PROX_ESTADO describe la acción que el dispositivo debe tomar frente a cambios que ocurran en las entradas (*x* e *y*), de acuerdo a dichos cambios se determina el valor que tomarán las salidas (*w* y *z*).

```
DECO_PROX_ESTADO: PROCESS (Est_actual, ent);
  BEGIN
        CASE Est_actual IS
              WHEN a =>
                    If    ent = "10" then
                          prox_estado <= b;
                    Elsif ent = "11" then
                          prox_estado <= c;
                    Else
                          prox_estado <= a;
                    End if;
              WHEN b =>
                    CASE (ent) IS
                          WHEN "00" => prox_estado <= d;
                          WHEN "10" => prox_estado <= e;
                          WHEN OTHERS => prox_estado <=b;
                    END CASE;
              WHEN c =>
                    If    ent = "1" then
                          prox_estado <= f;
                    Else
                          prox_estado <=c;
                    End If;
```

```
            WHEN d => prox_estado <= a;
            WHEN e =>
                  If    ent = "11" then
                        prox_estado <= g;
                  Else
                        prox_estado <=d;
                  End If;
            WHEN f => prox_estado <= g;
            WHEN g => prox_estado <= d;
            WHEN OTHERS => prox_estado <= a;
      END CASE;
END PROCESS DECO_PROX_ESTADO;
```

El process DECO_SALIDA describe la lógica combinacional para la definición de los valores que tomarán las salidas (*w* y *z*).

```
DECO_SALIDA: PROCESS (Est_actual, ent);
  BEGIN
      CASE Est_actual IS
            WHEN a =>
                  If    ent = "10" then
                        b <= "01";
                  Elsif ent = "11" then
                        c <= "00";
                  Else
                        a <= "00";
                  End If;
            WHEN b =>
                  CASE (ent)
                        WHEN "00" => d <= "01";
                        WHEN "10" => e <= "00";
                        WHEN OTHERS => b <= "00";
                  END CASE;
            WHEN c =>
                  If    ent = "1" then
                        f <= "00";
```

```
                Else  c <= "01";
                End If;
            WHEN d => a <= "00";
            WHEN e =>
                If    ent = "11" then
                      g <= "01";
                Else  d <= "00";
                End If;
            WHEN f => g <= "01";
            WHEN g => d <= "01";
            WHEN OTHERS => a <= "00";
        END CASE;
    END PROCESS DECO_SALIDA;
```

Como todo diseño en VHDL se requiere de una caja negra y la descripción de su funcionamiento, para este ejemplo la descripción se realiza a través de diagramas de estados.

Diagramas de estado

El diagrama de estados es para un circuito secuencial, lo que una tabla de verdad es para un circuito combinacional. El diagrama de estados cumple con la función de describir gráficamente las transiciones de estado a estado en función de los valores que se tengan presentes en las entradas o en las salidas. En la Figura. 3.33 se puede apreciar las partes que lo conforman.

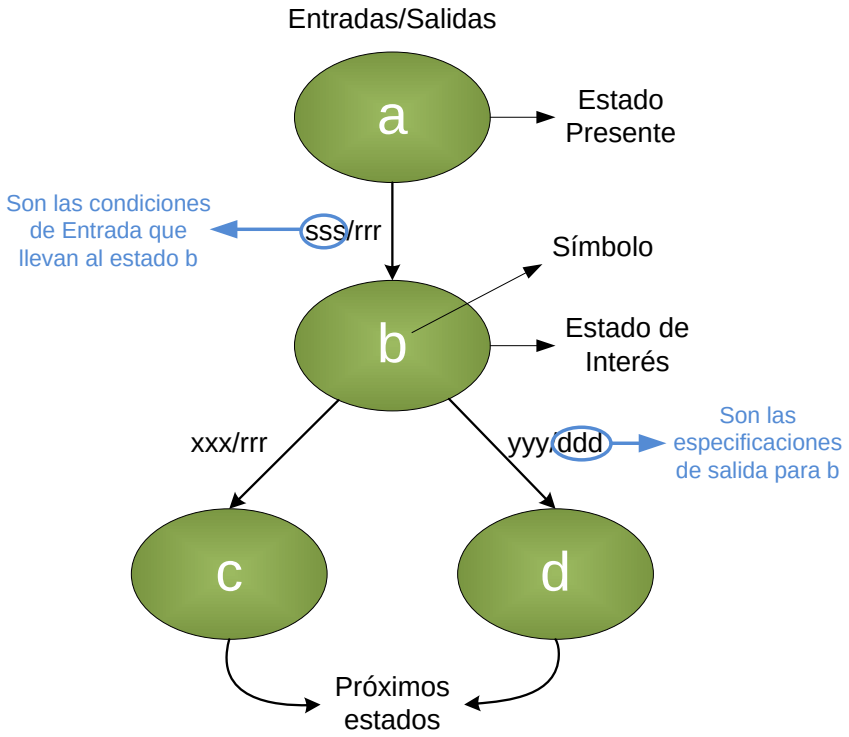


Figura. 3.33. Partes de un Diagrama de Estado

Para el ejemplo que se ha estado detallando, diseño de una FSM en VHDL, se presenta el diagrama de estados correspondiente:

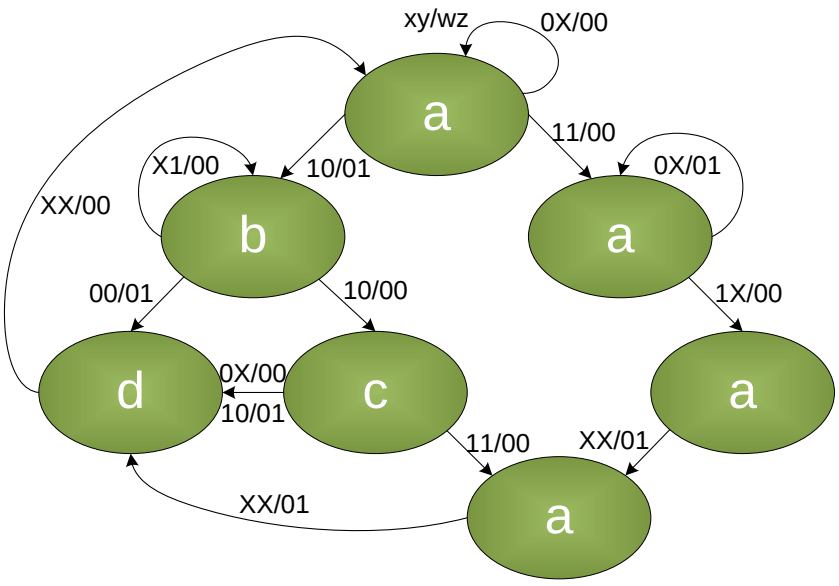


Figura. 3.34. Diagrama de Estados de una FSM

En función de esta estructura de código y representación funcional mediante diagramas de estados, en el presente proyecto se ha desarrollado un nuevo IP-Core (Interfaz de Red), tema que será tratado posteriormente en el **Capítulo 6**.

CAPÍTULO 4

DISEÑO E IMPLEMENTACIÓN DE UNA INFRAESTRUCTURA MPSOC SOBRE FPGA INTERCONECTADA POR BUSES

Para el desarrollo de un sistema de multiprocesamiento se debe tomar en cuenta la necesidad de la aplicación final. Dependiendo de la aplicación se agregan los periféricos y memorias que sean requeridos para la construcción del mismo. También hay que tomar en cuenta la capacidad de la FPGA y la cantidad de IP-Cores que ofrece la Plataforma de Desarrollo (EDK) antes de comenzar el flujo de diseño definido en el **Capítulo 3**.

En la Figura. 4.1 se muestra el diagrama de flujo para el diseño de un sistema de multiprocesamiento. Los diseñadores emplean este tipo de diagramas para la creación de un prototipo en la FPGA utilizando las herramientas de diseño de la plataforma de desarrollo. Previo a la obtención de un sistema con múltiples procesadores que satisfagan los requerimientos de la aplicación se realizan varias modificaciones para mejorar el hardware y se depura el software las veces que sean necesarias. (Asokan, 2007)

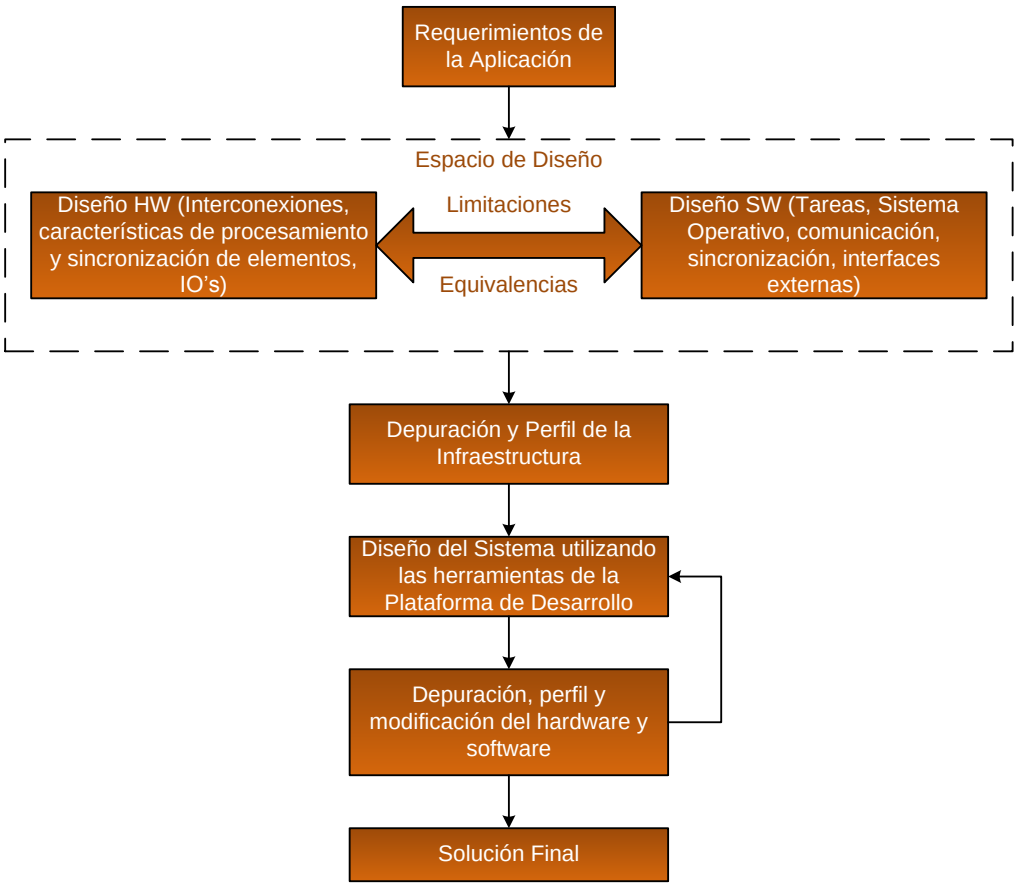


Figura. 4.1. Flujo de Diseño

4.1. APLICACIÓN DEL SISTEMA

La finalidad del presente proyecto es diseñar una arquitectura de multiprocesamiento y comparar su interconexión entre buses y una network-on-chip mediante la metodología de co-diseño explicada en el **Capítulo 3**. Para lo cual se escogió una aplicación de Esteganografía, la misma que trata sobre decodificar una cadena de caracteres que serán ocultos dentro de una imagen. La esteganografía se puede definir como la ocultación de información en un canal encubierto con el propósito de prevenir la detección de un mensaje oculto. La técnica se la tratará con más profundidad en el **Capítulo 8**.

4.2. DISEÑO BÁSICO DE MPSOC

Puesto que la aplicación consta de dos etapas, una de encriptación y otra de desencriptación de información, se ve la necesidad de emplear multiprocesamiento para dividir las tareas de forma independiente. Con lo cual se comparte distintos periféricos que permiten el manejo de la información entre cuatro procesadores interconectados como se puede apreciar en la Figura. 4.2

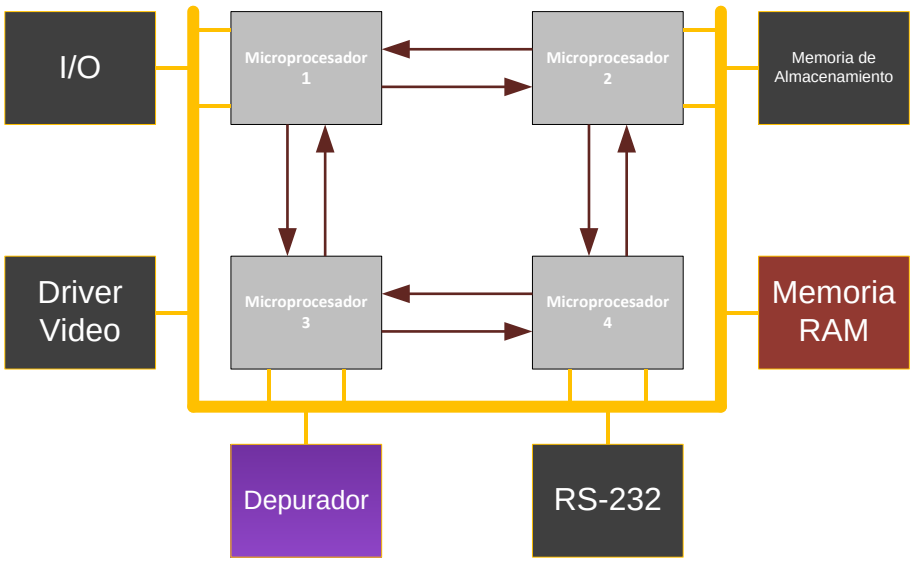


Figura. 4.2. Esquema general de una arquitectura de 4 microprocesadores

La Figura. 4.2 muestra en amarillo el bus de comunicaciones que es necesario para que los procesadores interactúen con los periféricos. Las flechas en color marrón representan el medio por el cual los procesadores intercambian información.

En otros aspectos generales los periféricos cumplen con las siguientes funciones:

IP-Core de I/O: Para el ingreso de instrucciones de diferentes funciones que debe cumplir los procesadores según la aplicación.

IP-Core para Driver Video: Da la capacidad de proyectar imágenes a través de un monitor con entrada VGA o DVI.

IP-Core de Depuración: Permite compilar y detectar posibles errores en la programación.

IP-Core de comunicación RS-232: Comunica la FPGA con el computador, para la descarga de los archivos *.bit y *.elf.

Memoria RAM: Cumple con el almacenamiento de toda la información que los procesadores deben procesar así como almacenar información para el despliegue de imágenes por medio del Driver de Video.

Memoria de Almacenamiento: Es la fuente de datos de la cual se extrae la información necesaria para realizar la aplicación.

4.3. PLATAFORMAS DE DESARROLLO DE HARDWARE BASADAS EN FPGA

Dada la característica programable de los FPGA y que existen FPGA muy potentes es posible implementar sobre ellos aplicaciones muy variadas que van desde un sencillo contador hasta complejos sistemas digitales para la industria, investigación, medicina, comunicaciones inalámbricas, procesamiento y almacenamiento de datos.

Con el aumento de las necesidades computacionales de los sistemas electrónicos actuales, los procesadores de propósito general desarrollados en un FPGA se ven desbordados y se hace prioritario buscar soluciones alternativas: Una posible solución es mejorar el procesador actual, mediante actualizaciones

estructurales que permitan obtener un mayor rendimiento, otra solución es el reemplazo por un procesador más moderno que ofrezca un rendimiento elevado o el desarrollo de nuevos coprocesadores, aunque esto involucre el incremento en el tiempo de elaboración del sistema y un aumento de su costo en el mercado. Una tercera solución consiste en utilizar múltiples procesadores en un solo chip (MPSoC), bien de propósito general o específico.

Las nuevas tecnologías FPGA ofrecen gran cantidad de recursos, millones de compuertas lógicas, bloques de memoria, así como uno o varios procesadores desarrollados por software (SCP, Soft Core Processors) que se pueden implementar utilizando recursos lógicos del FPGA. En la marca Xilinx se ofrece los Soft Cores; Power PC y MicroBlaze.

- **Power PC**

La arquitectura de un procesador Power PC, es una arquitectura de 64 bits con un subconjunto de 32 bits, definida en tres niveles o capas, los mismos que proporcionan flexibilidad y compatibilidad de software (Xilinx, PowerPC 405 Processor Block Reference Guide, 2010):

- User Instruction – Set Architecture (UISA)
- Virtual Environment Architecture (VEA)
- Operation Environment Architecture (OEA)

A continuación se detallan características fundamentales de la arquitectura embebida Power PC:

- Permite gestión de memoria para entornos de software embebido.
- Permite gestionar el set de instrucciones de memoria cache, para optimizar el rendimiento y el control de memoria en aplicaciones complejas.

- Posee registros de propósito especial para controlar el uso de los recursos de depuración.

Los procesadores Power PC son implementados en la gama Xilinx Virtex-II y Xilinx Virtex IV.

- **MicroBlaze**

Es una procesador de arquitectura RISC de 32 bits optimizado para trabajar en FPGA con conectividad para buses PLB (Processor Local Bus), AXI (Advanced High-performance Bus) o FSL (Fast Simplex Link). Para el caso de la aplicación de esteganografía se requiere del Soft Core MicroBlaze por su capacidad de soportar memoria cache mediante software, para acelerar el procesamiento de datos e instrucciones.

Para facilitar el diseño y reducir los costos, los fabricantes han creado varias plataformas de desarrollo de hardware. Una plataforma de desarrollo de hardware basada en FPGA es un entorno de aplicación donde se conectan módulos de procesamiento y periféricos que interactúan de forma flexible con el FPGA. En la marca Xilinx existe una amplia variedad de plataformas de desarrollo basadas en FPGA. Por sus capacidades de memoria, cantidad de unidades programables y adaptabilidad a la tecnología, se hace referencia a FPGA’s de gama alta: Spartan-6 y Virtex-6. Ambos modelos son comparados en la Tabla. 4.1.

Tabla de comparación FPGA		
FPGA	**Spartan-6**	**Virtex-6**
Celdas Lógicas	150.000	150.000
Recursos de Silicio para aplicaciones DSP	180	2016
Memoria RAM (DDR3)	256MB	512MB
Tecnología	45nm	40nm
Voltaje I/O	1.2V, 1.5V, 1.8V, 2.5V, 3.3V	1.5V, 2.5V

Tabla. 4.1. Comparación FPGA Xilinx de gama alta

De la Tabla. 4.1 se puede mencionar que ambos modelos poseen casi las mismas características. Sin embargo la aplicación de esteganografía necesita una capacidad de memoria de 512MB. Por lo tanto se ha seleccionado la plataforma de desarrollo con FPGA modelo Virtex-6 por tener una capacidad de memoria idónea para cumplir con los requerimientos de la aplicación anteriormente mencionada.

4.3.1. Especificaciones de Hardware Y Software

Virtex-6 FPGA ML605 Evaluation Kit

Virtex-6 ML605 Evaluation Kit es una plataforma base para el desarrollo de aplicaciones de alto rendimiento, conectividad e interfaces de memoria avanzada. Facilita el diseño en un entorno flexible y cumple estándares industriales de conectividad FPGA Mezzanine Connectors (FMC) que permiten la ampliación y personalización mediante tarjetas secundarias [72]. Virtex-6 ML605 FPGA (Figura. 4.3) posee un Encapsulado XC6VLX240T-1FFG1156, en la Tabla. 4.2 se detallan los componentes físicos de comunicación, configuración, memoria y dispositivos

de entrada y salida. (Xilinx, Getting Started with the Xilinx Virtex 6 FPGA ML605 Evaluation Kit, 2011)

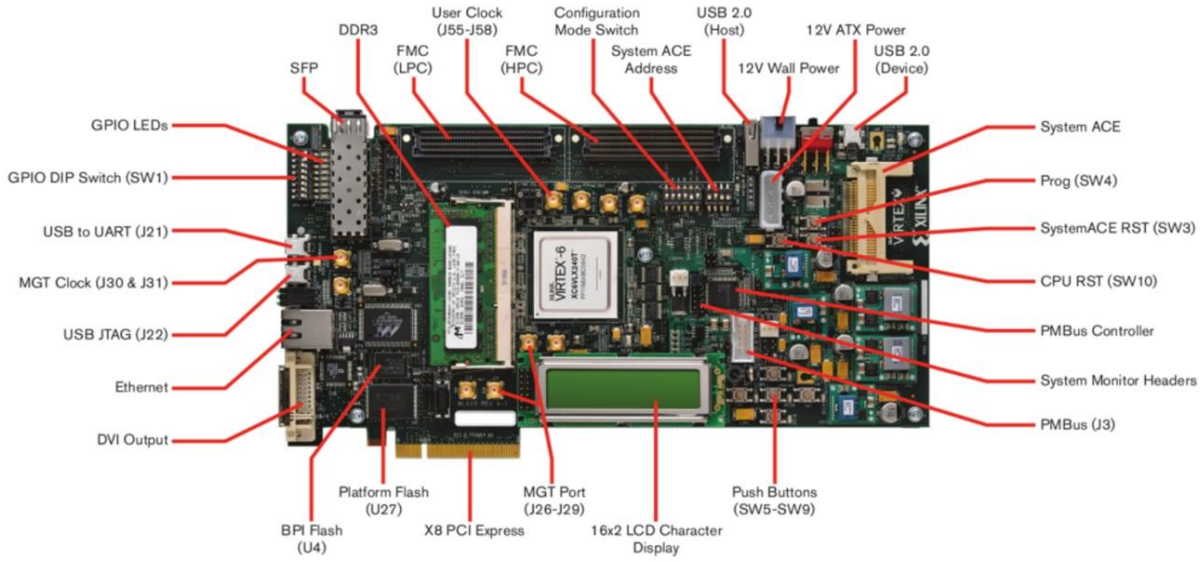


Figura. 4.3. FPGA ML605

XC6VLX240T-1FFG1156

Configuración	Comunicación	Memoria	Reloj	Entrada/Salida	Energía
Circuito Empotrado USB-JTAG	Tri-Speed Ethernet	DDR3 SODIMM 512MB	Oscilador Diferencial 200 MHz	LCD 16x12	12V Adaptador de Pared
16 MB Flash XL	Transductor SFP	BPI Flash 32 MB	Socket Oscilador 66 MHz	Puerto de Salida DVI	
32 MB Paralell Flash	USB-UART	IIC EEPROM 8 Kb	SMA Conector Reloj Externo	GPIOs 5 PushButtons 8 Dip Switches 13 Leds	
Controlador System ACE Compact Flash	USB Host Y Periféricos		GTX Conector Reloj Externo	Puerto FMC (HPC) 160 Entradas y Salidas Configurables	
	PCI Express			Puerto FMC (LPC)	

XC6VLX240T-1FFG1156					
Configuración	**Comunicación**	**Memoria**	**Reloj**	**Entrada/Salida**	**Energía**
				68 Entradas y Salidas Configurables	

Tabla. 4.2. Características FPGA ML605 (Virtex-6)

4.3.2. MicroBlaze

El Procesador MicroBlaze es un Soft Core optimizado para aplicaciones en FPGAs de la marca Xilinx. MicroBlaze es un procesador RISC de 32 bits diseñado para trabajar bajo la arquitectura Harvard y el estándar CoreConnect de IBM. Este procesador es altamente configurable permitiendo seleccionar un conjunto amplio de características específicas para cada diseño. (Xilinx, ML401/ ML402/ ML403 Evaluation Platform, 2010)

Características de MicroBlaze Soft Procesador:

- Posee 32 bits de Registros de Propósito General
- Posee soporte de Memoria
- Es un procesador de 32 bits con 8 KB de cache de Instrucciones y 8 KB de cache de Datos, que puede ser configurable desde 2 KB hasta 64KB (basado en el bloque RAM)
- Posee soporte para excepciones de hardware, especialmente para instrucciones ilegales, errores en el bus de datos, en el bus de instrucciones, para división, punto flotante, y MMU.
- Posee soporte para la lógica de depuración
- Soporta Unidad de Punto Flotante (FPU)
- Permite la aceleración de Hardware
- Posee Unidad de Gestión de Memoria (MMU)
- Admite arquitectura Segmentada, pipeline.

La arquitectura Harvard que rige en el MicroBlaze, separa la memoria de datos de la memoria de instrucciones. Además posee un bus de comunicaciones

para memorias y otro bus para periféricos, con lo cual permite separar las conexiones del procesador con sus memorias en un solo bus, mientras que el conexionado con periféricos lo realizan en otro diferente, disminuyendo la carga del sistema. (Xilinx, LogiCORE IP MicroBlaze, 2012)

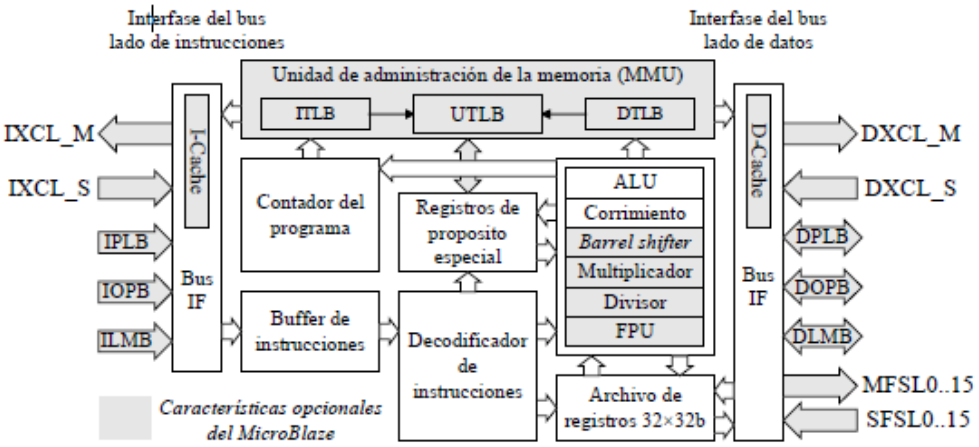


Figura. 4.4. Estructura Microprocesador MicroBlaze

4.3.2.1. Buses, Puertos e Interfaces del MicroBlaze

MicroBlaze posee buses separados para datos e instrucciones y genera interfaces de comunicación por cada bus. En la Figura. 4.5 se muestran los distintos buses que se utilizan en el MicroBlaze tal como:

- El bus de memoria local, Local Memory Bus (LMB)
- El bus local del procesador para control de periféricos, Processor Local Bus (PLB)
- El bus de periféricos en chip, On Chip Peripheral Bus (OPB)
- El bus de enlace, Fast Simple Link (FSL)
- Interfaz de memoria externa Xilinx Caché Link (XCL)

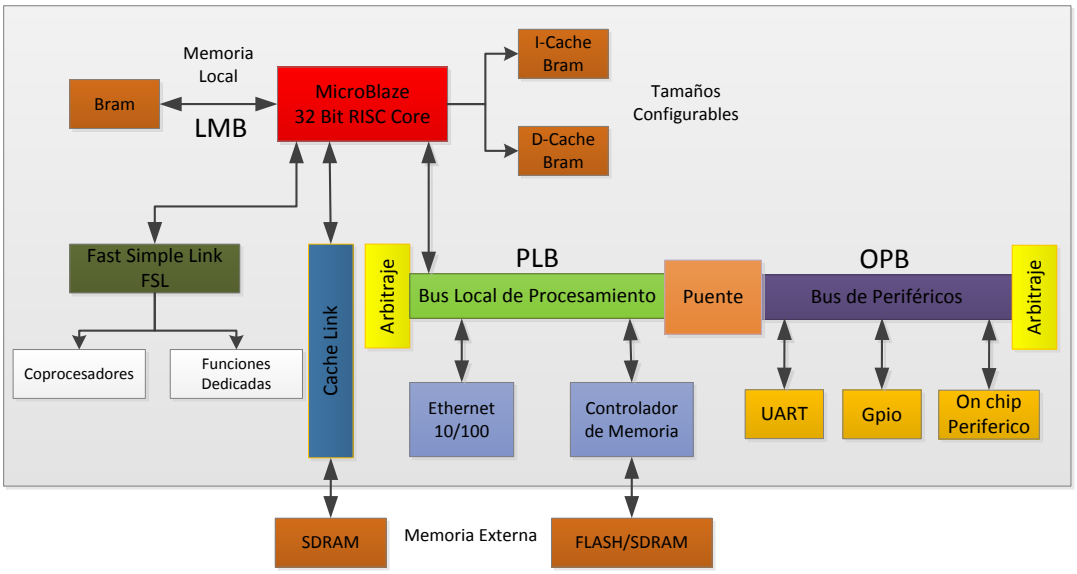


Figura. 4.5. Conexiones de Buses en el MicroBlaze

- **Local Memory Bus (LMB)**

EL bus LMB, Local Memory Bus, ver Figura. 4.6, es un bus síncrono, que permite el acceso a los bloques internos de memoria. El LMB proporciona un protocolo simple para una transferencia eficiente de bloques de RAM mediante la utilización de tamaños de palabras de 32 bits. Además sólo admite un maestro en su implementación y puede ser utilizado para interconectar los puertos de instrucciones y de datos del procesador sin la necesidad de un árbitro en la comunicación. El LMB posee un ancho de banda máximo de 500 Mb/s. (Xilinx, LogiCORE IP MicroBlaze, 2012)

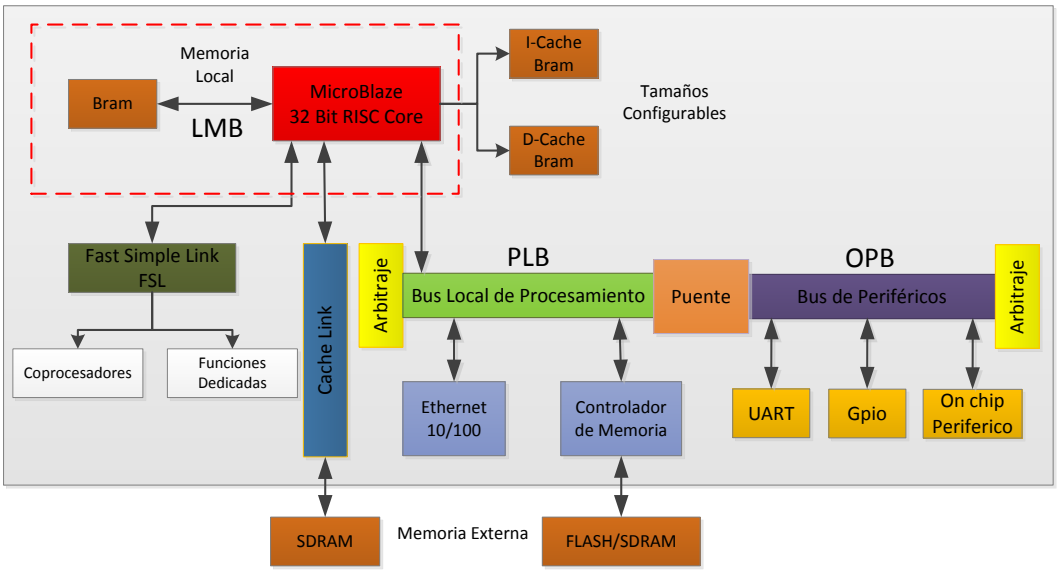


Figura. 4.6. Bus LMB para memoria local.

- **Processor Local Bus (PLB)**

El bus PLB, Processor Local Bus, ver Figura. 4.7, es un bus totalmente sincronizado con un solo reloj. El PLB proporciona una infraestructura de comunicación entre maestros y esclavos con un canal de datos de 32, 64 ó 128 bits. El PLB está compuesto principalmente por una unidad de control de bus, un watchdog timer y la unidad de lectura y escritura de trayectoria. Este bus es parte de la arquitectura CoreConnect de IBM que sirve como interface de datos de alta velocidad al IP Core MicroBlaze. Todos los periféricos y el sistema de memoria se comunican con el procesador utilizando este bus. Además posee una topología de interconexión de bus compartido o punto a punto, minimizando la latencia puesto que remueve el arbitraje de comunicación. (Xilinx, LogiCORE IP Processor Local Bus, 2010)

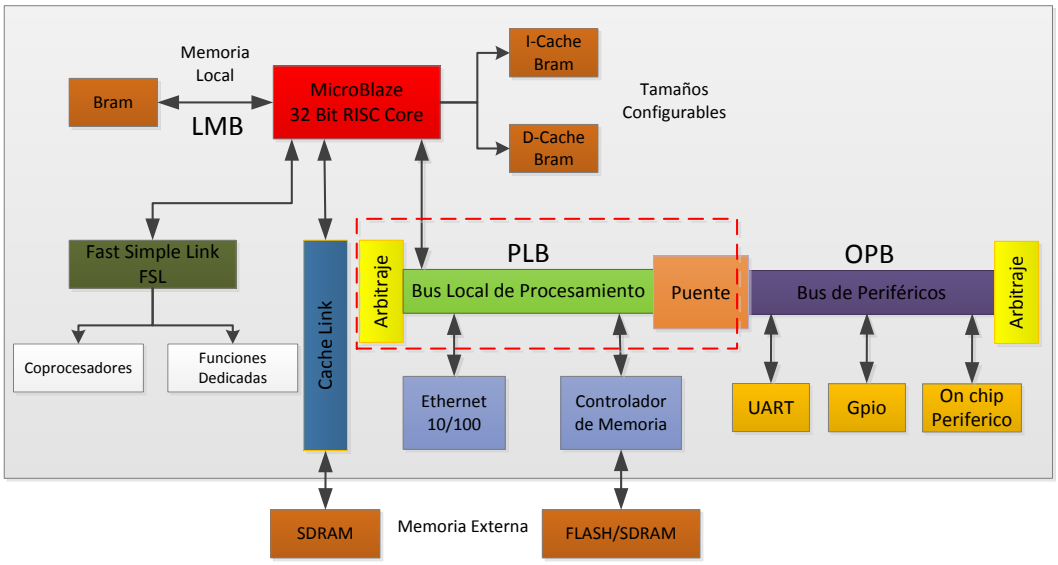


Figura. 4.7. Bus PLB (Processor Local Bus)

- **On Chip Peripheral Bus (OPB)**

El bus OPB, On chip Peripheral Bus, ver Figura. 4.8, desacopla dispositivos de bajo ancho de banda desde el PLB. El ancho de banda máximo del bus OPB es 167 Mb/s. El bus OPB utiliza arbitraje centralizado que reduce la carga sobre el PLB. Posee un reloj único de sincronismo y un bus compartido de direcciones de 32 bits. El OPB soporta 16 maestros y un número ilimitado de esclavos.

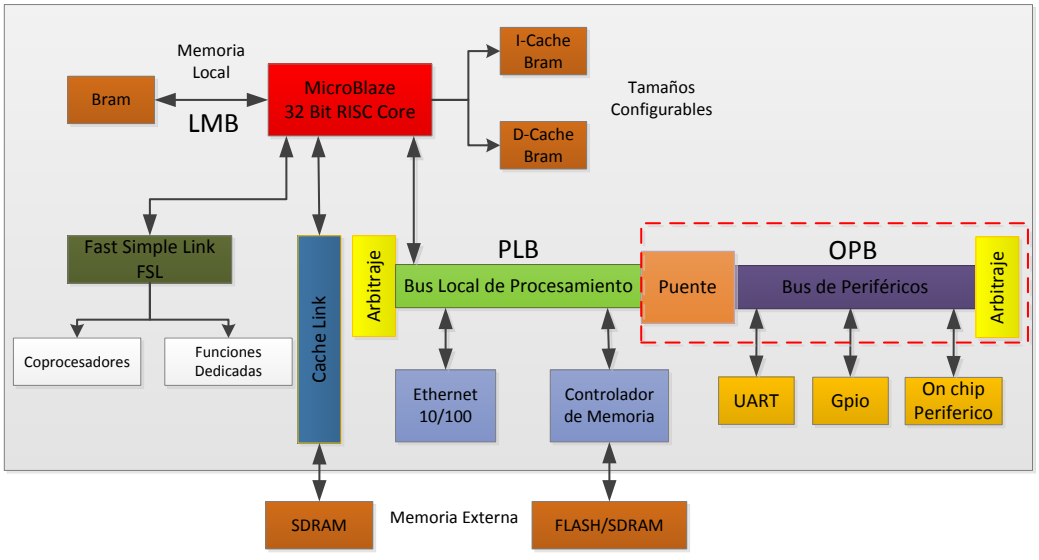


Figura. 4.8. Bus OPB (On Chip Peripheral)

- **Fast Simple Link (FSL)**

El FSL, Fast Simple Link, ver Figura. 4.9, provee un canal de comunicación unidireccional de alta velocidad en 2 ciclos de reloj a 600 Hz. El enlace FSL posee un ancho de banda máximo de 800 Mb/s. FSL proporciona conexión punto a punto no arbitrada de 32 bits entre las memorias internas FIFO (First in First out). FSL utiliza una FIFO de salida en el elemento maestro y una FIFO de entrada de en el elemento esclavo. El enlace FSL tiene un tamaño configurable entre 1 a 8192 palabras, ver Figura. 4.10. El enlace FSL es el medio ideal para extender la capacidad de conectividad del procesador utilizando aceleradores de hardware diseñados por el usuario.

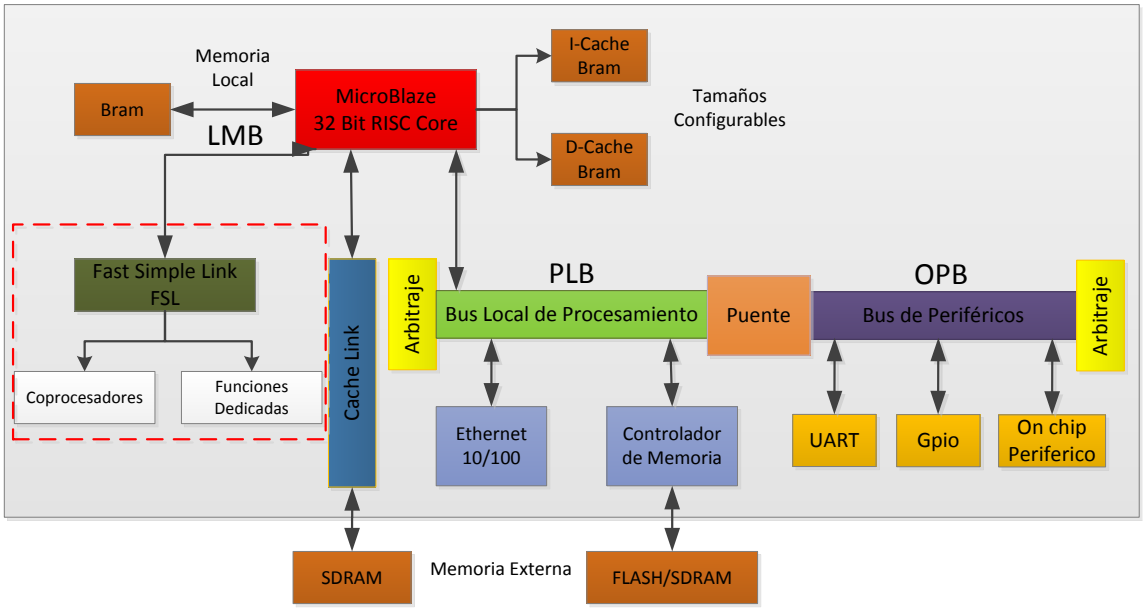


Figura. 4.9. Enlace FSL (Fast Simple Link)

La expansión de capacidad de conectividad del procesador se puede lograr también a través de la implementación de código en VHDL para su respectiva sintetización sobre el MicroBlaze. El MicroBlaze soporta hasta 16 dispositivos conectados mediante este protocolo de arquitectura dedicada. (de la Fuente, 2007)

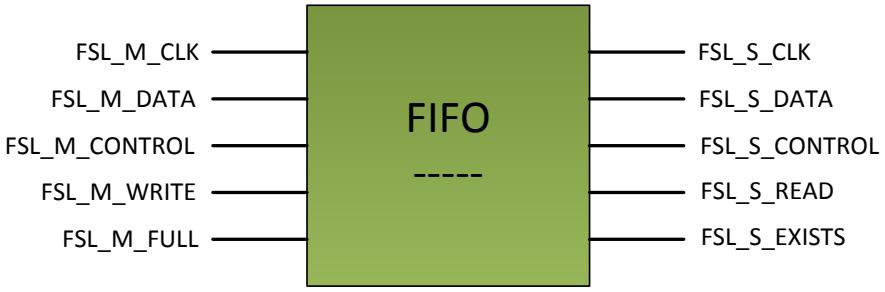


Figura. 4.10. Estructura del bus FSL (Fast Simple Link)

- **Xilinx Caché Link (XCL)**

La interfaz XCL, Xilinx Caché Link, ver Figura. 4.11, es una solución de alto rendimiento para accesos a memoria externa. XCL es un método de comunicación de baja latencia y maneja entre cuatro y ocho palabras de memoria caché por instrucción. El acceso a memoria caché solo está disponible cuando se habilita su interfaz por software en el MicroBlaze. (Xilinx, LogiCORE Multi-Channel OPB External Memory Controller, 2006)

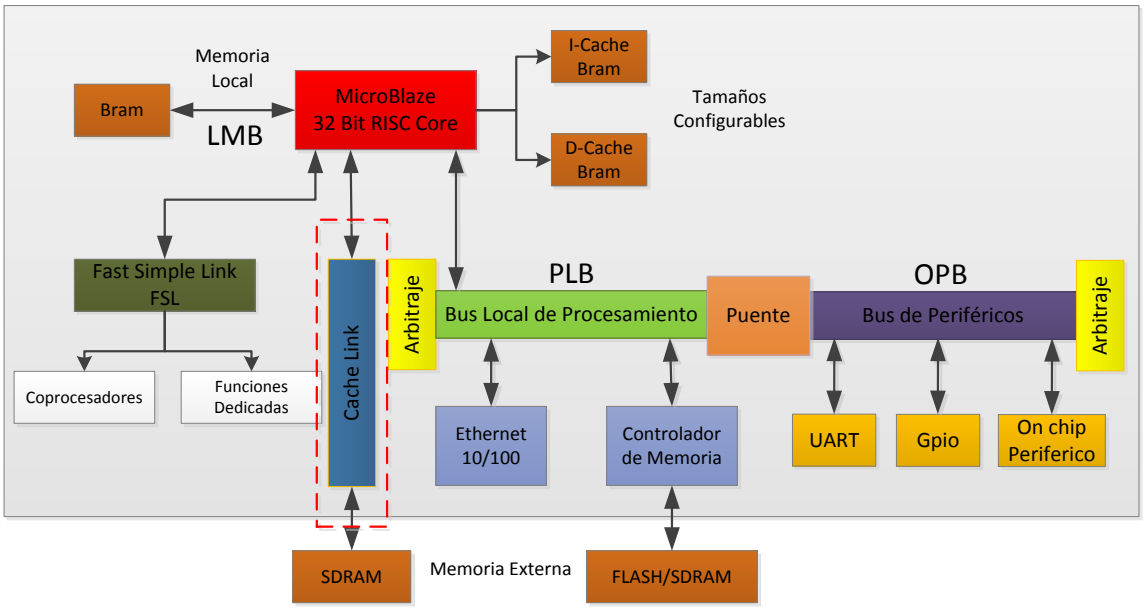


Figura. 4.11. Bus XCL (Xilinx Caché Link)

4.4. ARQUITECTURA DEL HARDWARE

Pueden desarrollarse una gran cantidad de arquitecturas de acuerdo a la necesidad de la aplicación como ya se ha mencionado anteriormente. La mayoría de ellas se interconectan mediante un bus PLB, ya sean memorias, periféricos, procesadores, etc. (Asokan, 2007). Por lo tanto antes de continuar con la arquitectura desarrollada para el presente proyecto, se hace mención a los medios para las comunicaciones entre procesadores que ofrece el XPS de Xilinx.

4.4.1. Medios de Comunicación Físicos entre Procesadores

***Procesadores comunicados por Mailbox*:** Mailbox es un IP-Core desarrollado por Xilinx como medio de comunicación entre procesadores. Mailbox permite el paso de mensajes simples de forma sincrónica y asincrónica. En el primer caso el procesador mantiene un sondeo contante en busca de nuevos datos, mientras que en el segundo caso se envía una interrupción hacia el procesador para informarle sobre la existencia de datos presentes. (Asokan, 2007). El IP-Core Mailbox se encuentra en Interprocessor Communication de la pestaña IP Catalog como se muestra en la Figura. 4.12.

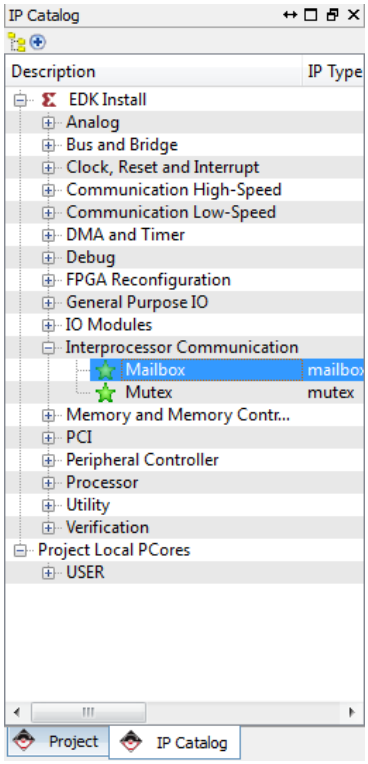


Figura. 4.12. Ubicación de Mailbox

Mailbox es un IP-Core que permite comunicación bidireccional entre dos procesadores, su conexión y programación se vuelve problemática al momento de agregar más procesadores. Los problemas de comunicación entre más de dos procesadores, utilizando Mailbox, se deben principalmente a que su conexión satura las direcciones de registro del Bus PLB por la creación de puentes que permitan el uso compartido de periféricos necesarios para el desarrollo de la aplicación de esteganografía. En la Figura. 4.13 se puede apreciar la conexión de 4 MicroBlaze a través de mailbox, en donde se imposibilita realizar una implementación de topología apropiada para interconectar los procesadores al mismo bus PLB.

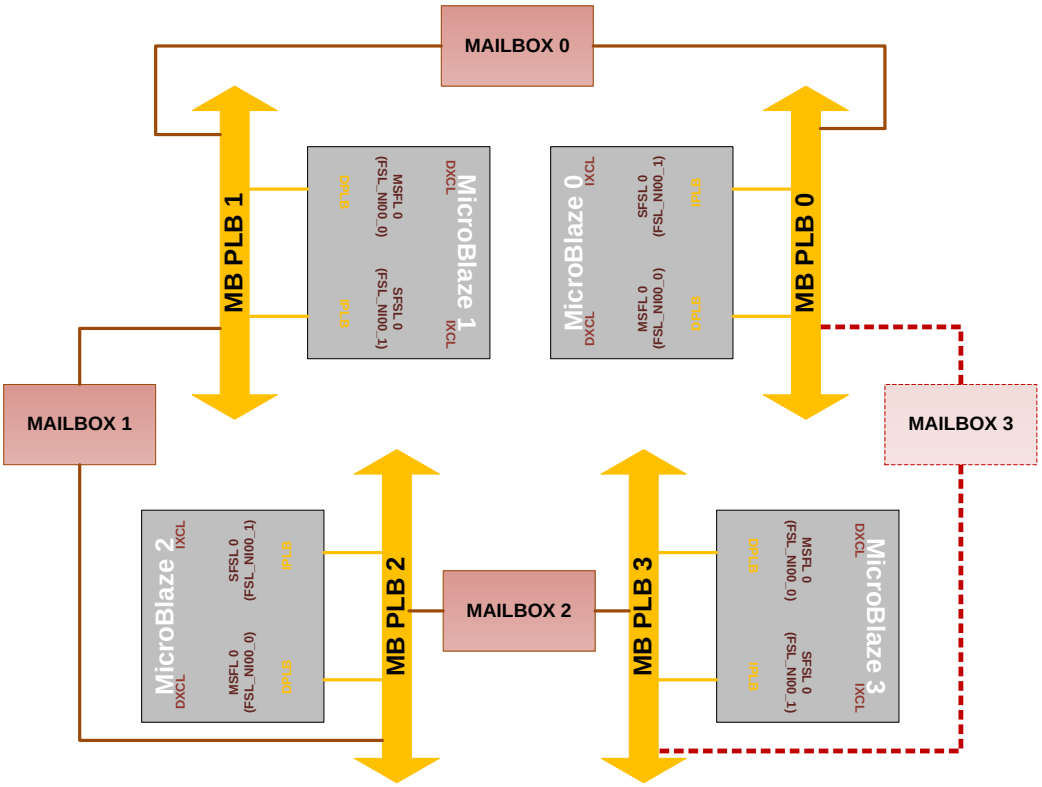


Figura. 4.13. Comunicación por Mailbox para 4 MicroBlaze

***Procesadores comunicados por FSL*:** FSL es el bus que se emplea en la arquitectura del hardware, debido a que puede soportar efectivamente topologías, a medida que se incremente el número de procesadores en la FPGA, como mesh, anillo y estrella. Además se menciona que este bus es utilizado para la comunicación de IP-Cores de diseño específico. (Pantalopoulos & Brokalakis, 2012) La Figura. 4.14, muestra el lugar en donde se encuentra el FSL dentro del IP Catalog.

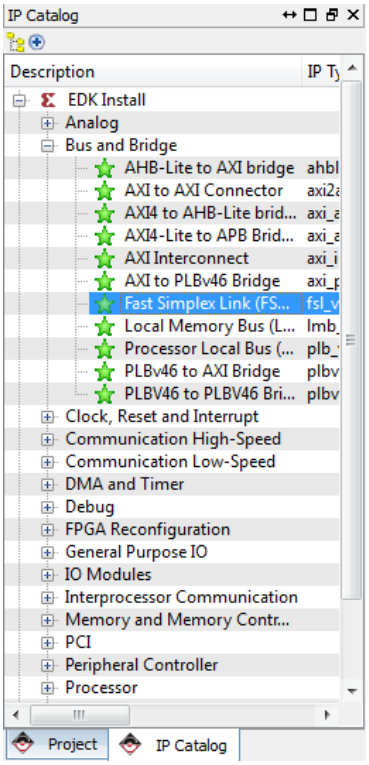


Figura. 4.14. Ubicación del FSL

En la Figura. 4.15 se muestra cuatro MicroBlaze interconectados en topología mesh empleando buses unidireccionales FLS de conexión punto a punto. Este modo de comunicación es el seleccionado para realizar la aplicación de esteganografía puesto que es de fácil implementación e independiente del bus PLB.

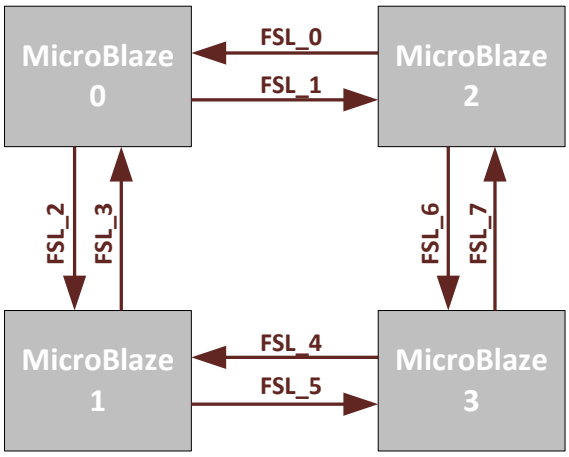


Figura. 4.15. Comunicación por FSL de 4 MicroBlaze

4.4.2. Diseño de la Arquitectura

Una vez definido el FPGA en donde desarrollar el proyecto se estable el diseño la arquitectura de hardware. La Figura. 4.16 muestra 4 MicroBlaze conectados por FSL para compartir información y un bus PLB para el uso de periféricos necesarios para cumplir con la aplicación de Esteganografía.

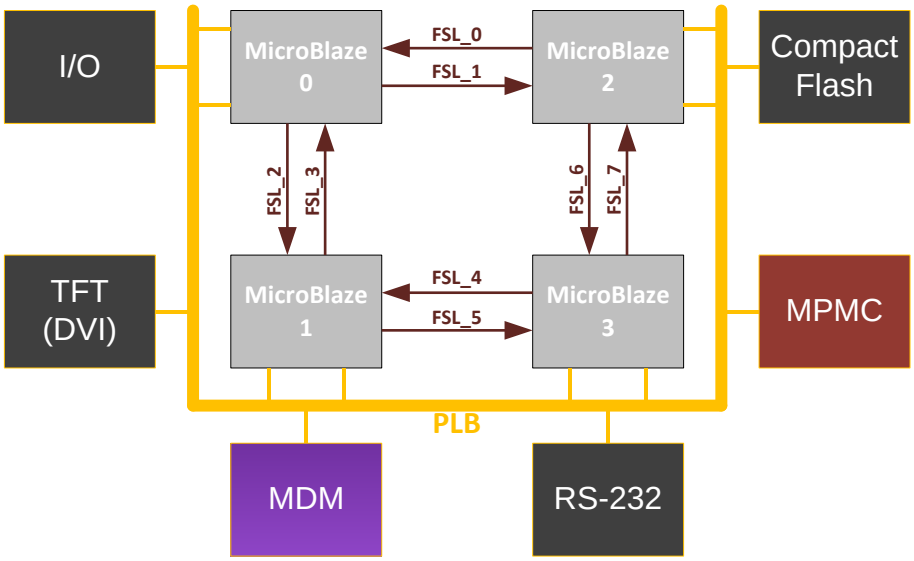


Figura. 4.16. Arquitectura de 4 MicroBlaze interconectados por buses

El diseño de la arquitectura consta de los siguientes IP-Cores:

I/O: están constituidos por dos periféricos que son los Dipswitchs y los LED para el ingreso de instrucciones y para la proyección de la instrucción ingresada respectivamente.

TFT: es el controlador de video por donde se proyectan imágenes en una pantalla con entrada de VGA o DVI.

MDM: que cumple la función de depurar el software que sea programado para los MicroBlaze.

RS-232: habilita la comunicación entre el FPGA y el computador.

MPMC: es un controlador de memoria externa, para el caso de la Virtex-6 el MPMC controla una memoria RAM de 512MB.

Compact Flash (CF): es la unidad de memoria de almacenamiento extraíble en donde se almacena la información necesaria para realizar la aplicación de esteganografía.

4.5. IMPLEMENTACIÓN DE LA ARQUITECTURA DE HARDWARE

Para mayor detalle de la implementación y configuración de la arquitectura desarrollada se debe revisar: Anexo 2, Anexo 3 y Anexo 6. Por lo tanto, a continuación se detalla los componentes que conforma la arquitectura de un MPSoC interconectado por buses diseñado para una aplicación de esteganografía.

En la Figura. 4.17 se destaca la Interfaz MPMC o Multi-Port Memory Controller, que es un IP-Core que permite la comunicación entre los procesadores con una memoria externa DDR3 de 512MB. El MPMC soporta hasta 8 conexiones (un par para cada MicroBlaze) de manera personaliza, haciendo uso de la interfaz XCL o bien por conexión directa al PLB. En la conexión del MPMC con el MicroBlaze mediante XCL se asigna un acceso de memoria directa para el procesador. Mientras que en la conexión del MPMC con el MicroBlaze mediante PLB la memoria se comparte para todos los procesadores. (Pantalopoulos & Brokalakis, 2012) (Xilinx, LogiCore IP Multi-Port Memory Controller (MPMC) (v6.04.a), 2011)

La arquitectura de la Figura. 4.17 muestra las conexiones del MPMC mediante XCL y PLB. El uso de ambas conexiones se hace necesario para que la memoria también funcione como Buffer de video, motivo por el cual el MicroBlaze 3 tiene acceso a la memoria DDR3 por medio del PLB.

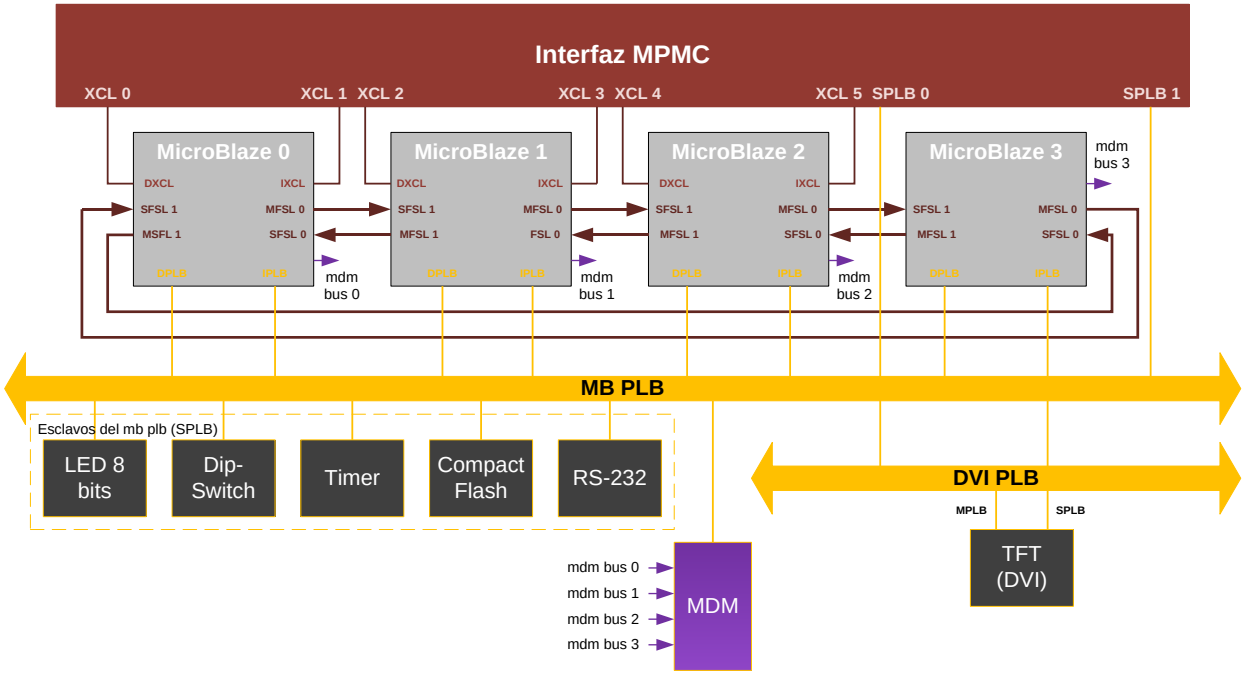


Figura. 4.17. Arquitectura de 4 procesadores embebidos para aplicación de Esteganografía

Los componentes más relevantes en esta arquitectura además del MPMC, son el TFT, y el lector de Compact Flash, los cuales se detalla a continuación:

El *IP-Core TFT o Thin Film Transistor* es el medio por el cual se controla una pantalla LCD o CTR. El TFT consta de una conexión maestra conectada de forma independiente del bus DVI PLB hacia la memoria DDR3 y una conexión esclava conectada al bus MB PLB para adquirir la información de los pixeles que serán proyectados. La conexión directa a memoria por medio del bus DVI PLB es necesario para que se pueda asignar 2MB de espacio requerido para mantener la imagen proyectada en la pantalla LCD o CTR. El TFT trabaja a una frecuencia de

25MHz con una resolución de 640x480 pixeles y tolera conexiones para DVI y VGA. La conexión VGA requiere de un adaptador de DVI a VGA. (Xilinx, XPS Thin Film Transistor (TFT) Controller (v2.01a), 2010)

El lector de *Compact Flash (CF) o System ACE Interface Controller (SYSACE)* es un IP-Core flexible que permite obtener información de una memoria CF (Xilinx, LogicCORE XPS SYSACE (System ACE) Interface Controller (v1.01a), 2010). El SYSACE usualmente es empleado para verificar que el FPGA está funcionando de manera correcta al tener acceso a archivos ejecutables ubicados dentro de la CF que permiten realizar un test de periféricos (Cadena & Mollocana, 2012). Para el caso de la arquitectura mostrada en la Figura. 4.17 el módulo SYSACE fue adaptado para poder adquirir información necesaria para la aplicación de Esteganografía.

Los demás IP-Cores que también forman parte de la arquitectura son: dos General Purpose IO (GPIO's) constituidos por los Led 8 Bits y Dip-Switchs, xps_timer, un RS232_Uart16550 y un MicroBlaze Debug Module (MDM). Los dos últimos IP-Cores son elementos indispensables en cualquier diseño, pues el RS232 permite una comunicación entre el computador y la FPGA (Xilinx, EDK Concepts, tools and techniques), y el MDM provee soporte para depuraciones en el software que se está desarrollando desde uno hasta ocho procesadores (Xilinx, MicroBlaze Debug Module (MDM) (v2.00b), 2011). En la Tabla. 4.3 se muestra los recursos utilizados en el FPGA Virtex-6.

Recursos utilizados en FPGA ML605 Virtex 6			
Lógica Utilizada	**Utilizado**	**Capacidad Total**	**Porcentaje**
Número de LUT's	13861	150720	9%
Número de Registros de Silicio	14868	301440	4%

Tabla. 4.3. Consumo FPGA ML605 Virtex-6

4.6. DISEÑO DEL SOFTWARE

Antes de realizar una programación de software para el diseño desarrollado en la FPGA hay que cerciorarse de que todos los procesadores estén conectados al módulo MDM, esto se hace por medio de la pestaña Debug en el XPS. La Figura. 4.18 se muestra la ventana Debug Configuration y el modo de acceder a ella. Dentro de esta opción se selecciona la opción JTAG UART en donde aparecerán todos los procesadores conectados al PLB.

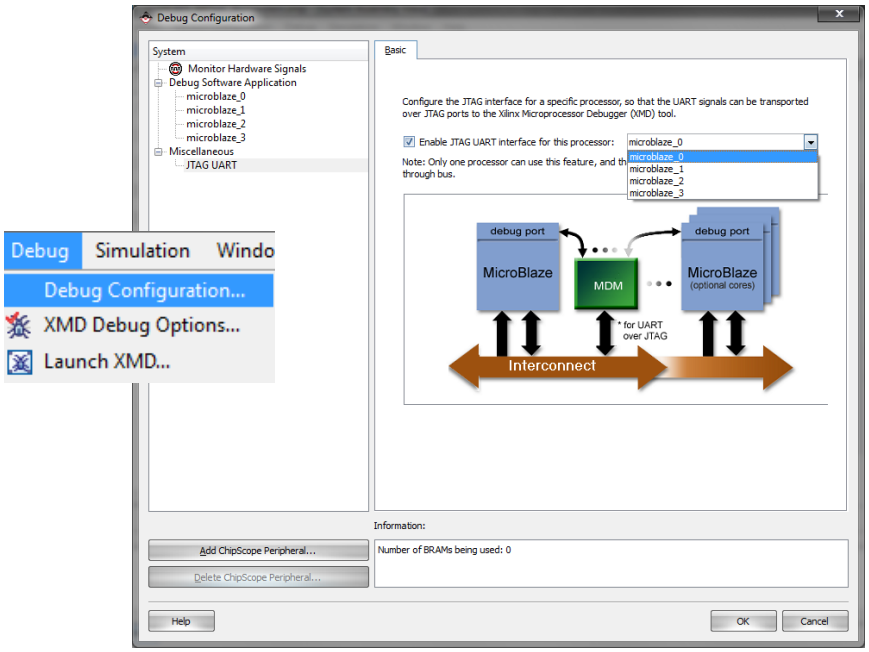


Figura. 4.18. Configuración del MDM

En la Figura. 4.18, se puede apreciar que existen cuatro procesadores, esto se debe principalmente a que todos están sobre un mismo bus PLB, esto facilita mucho la configuración del MDM pues para la arquitectura mostrada anteriormente (Figura. 4.17) tan solo basta con seleccionar uno de los procesadores, y el resto de todas formas compartirán el mismo módulo. (Nuñez)

4.6.1. Exportación del XPS al SDK

El hardware generado debe ser exportado a la plataforma de generación de software SDK, ver Figura. 4.19, una vez hecho esto se procede a ejecutar dicha plataforma en una nueva ventana de proyecto. En esta ventana se crearan las librerías y todas las configuraciones necesarias para la puesta en marcha de los cuatro MicroBlaze según la aplicación.

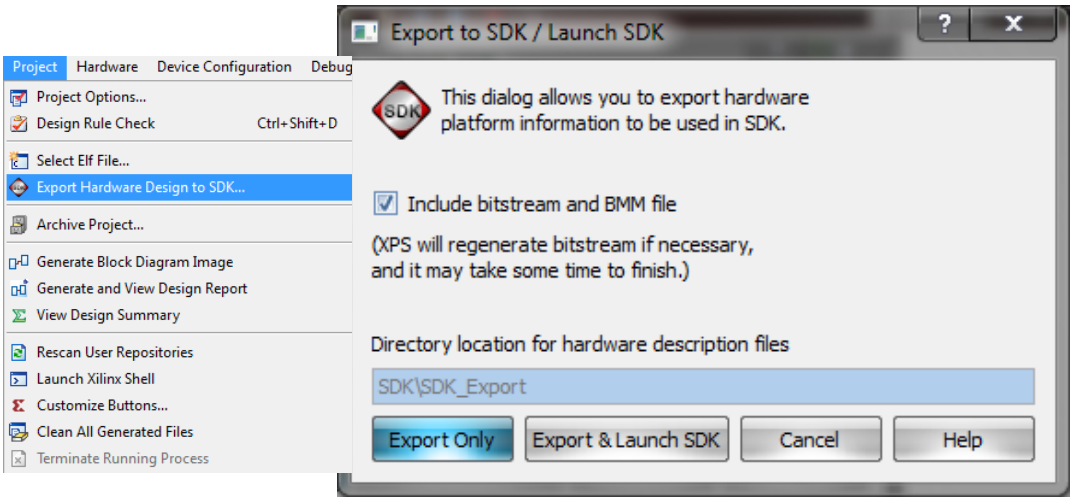


Figura. 4.19. Exportación XPS a SDK

- **Nueva Ventana de Proyecto**

En el SDK se crea el Sistema Operativo (OS) del hardware diseñado, y para este caso se crea un grupo de librerías en el ambiente de programación

Standalone. Previo a esta ejecución se debe crear antes una carpeta con la denominación SDK_Workspace (Xilinx, AXI Interface Based ML605/SP605 MicroBlaze Processor Subsystem, 2011) (Cadena & Mollocana, 2012), en el cual se almacenará el OS de cada procesador que se ha agregado en la Figura. 4.17.

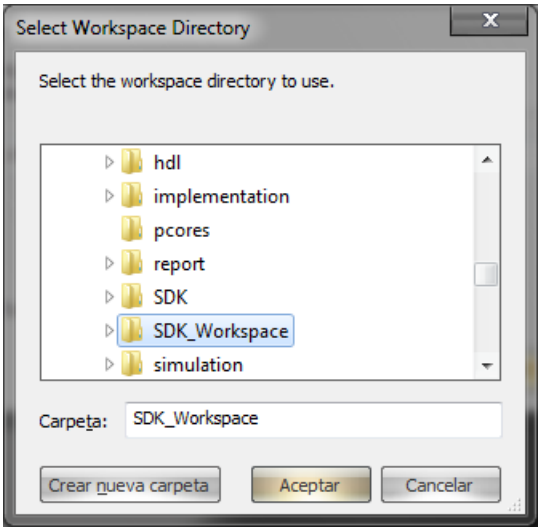


Figura. 4.20. Directorio de trabajo del SDK

La Figura. 4.20, muestra el directorio en donde se selecciona la carpeta SDK_Workspace. Dicha carpeta se la debe crear de forma manual dentro de la carpeta de proyecto (Xilinx, AXI Interface Based ML605/SP605 MicroBlaze Processor Subsystem, 2011) (Cadena & Mollocana, 2012). La ventana de directorio se despliega al dar clic en Browse (ver Figura. 4.21), cuando el proyecto es nuevo, en donde se hace la búsqueda de la carpeta creada.

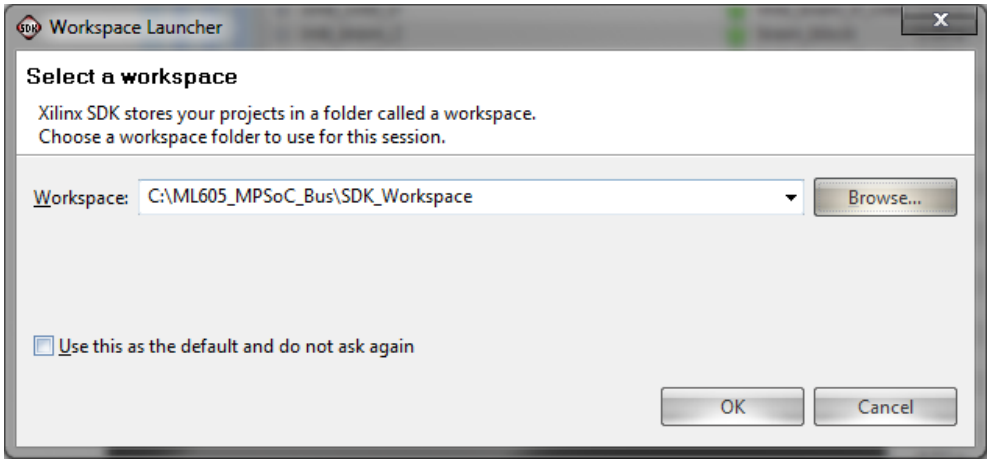


Figura. 4.21. Ventana Workspace Launcher

- **Especificaciones del Hardware**

Las especificaciones del hardware se encuentran en un archivo XML, dentro de la carpeta de proyecto en SDK/SDK_Export/hw. El archivo XML detalla todo el hardware que ha sido generado en el XPS y resulta de la exportación anteriormente mencionada.

Cuando la ventana de proyecto se abre, el archivo XML es el primero que debe ser llamado, pues a partir de este se pueden crear los grupos de librerías y controladores de los periféricos creados o agregados en el XPS.

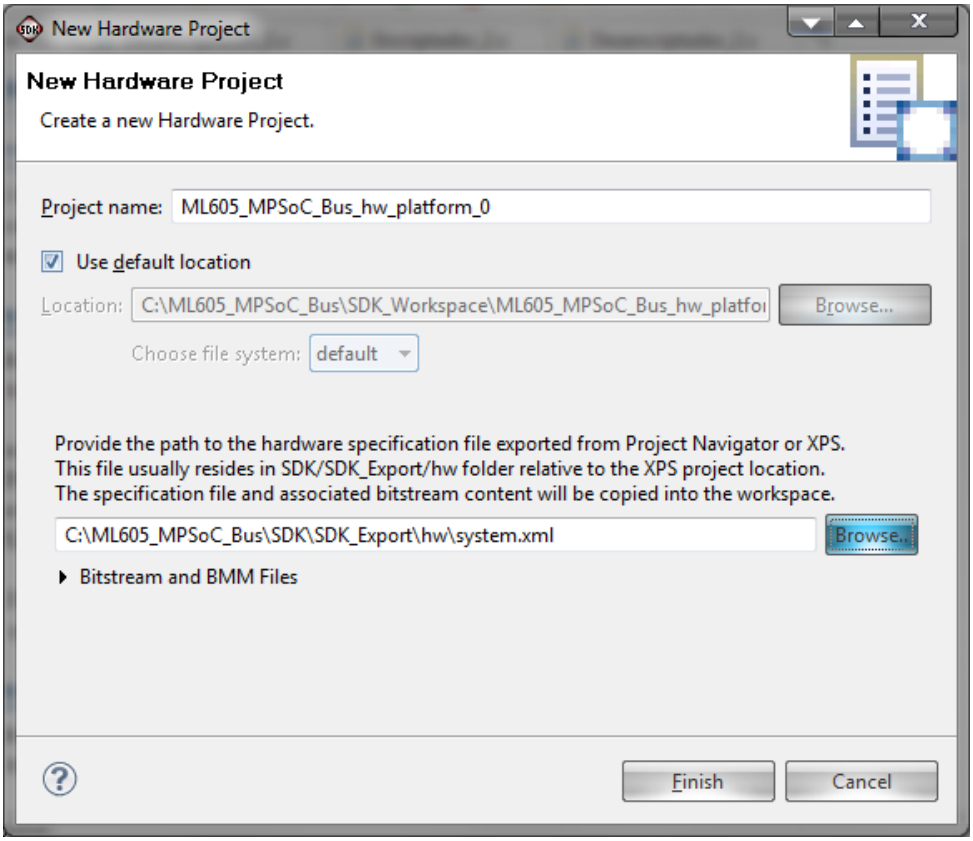


Figura. 4.22. Llamada del archivo XML

La Figura. 4.22 muestra la ventana en donde se carga el archivo XML. Esta ventana se encuentra al dar clic en la pestaña File/New/Xilnx Hardware Platform Specification, en la barra de herramientas.

- **Librerías y Controladores**

Las librerías y controladores se generan al crear un BSP, el cual permite el desarrollo de programación propia o bien hacer uso de pequeñas programaciones ya pre-establecidas (API) por el SDK. (Xilinx, AXI Interface Based ML605/SP605 MicroBlaze Processor Subsystem, 2011) (Cadena & Mollocana, 2012)

Para una arquitectura que contiene cuatro procesadores, se debe crear la misma cantidad de BSP equivalentes al número de procesadores contenidos en dicha arquitectura. La Figura. 4.23, muestra la ventana de generación del BSP correspondiente al MicroBlaze 0 mostrado en la Figura. 4.17.

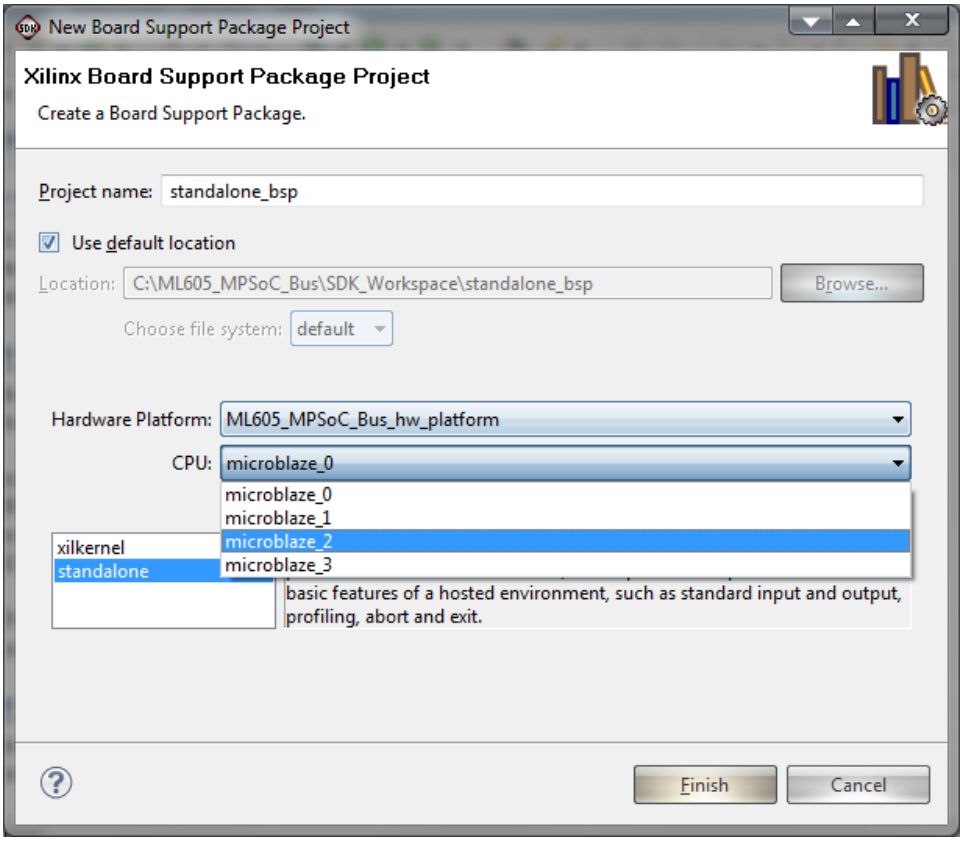


Figura. 4.23. Ventana Board Support Package (BSP)

- **Configuración RS-232 y MDM para Cuatro Procesadores**

El BSP debe ser configurado de acuerdo a las necesidades de la aplicación, y para el caso de tener más de dos procesadores embebidos dentro de la misma arquitectura, los controladores para los módulos RS-232 y MDM deben configurarse según lo expuesto en la Tabla. 4.4.

Configuración del BSP	
MicroBlaze 0 (BSP 0)	
Stdin	RS232_Uart_1
Stdout	RS232_Uart_1
MicroBlaze 1 (BSP 1)	
Stdin	mdm_0
Stdout	mdm_0
MicroBlaze 2 (BSP 2)	
Stdin	mdm_0
Stdout	mdm_0
MicroBlaze 3 (BSP 3)	
Stdin	mdm_0
Stdout	mdm_0

Tabla. 4.4. Configuración BSP para cuatro procesadores

Esta configuración es importante para poder implementar el hardware en el FPGA y a su vez poder realizar la descarga del respectivo software que contiene el set de instrucciones para implementar la aplicación de esteganografía. El controlador del módulo MDM hace uso del puerto JTAG para la depuración del software una vez que el hardware ha sido implementado en el FPGA. Por otro lado, al seleccionarse la opción RS232_Uart_1, el controlador del RS-232 habilita una comunicación para el computador y otra comunicación para el MDM (Figura. 4.24). Todo esto a partir de una interfaz interna del MDM conectada al PLB denominada UART. (Asokan, 2007)

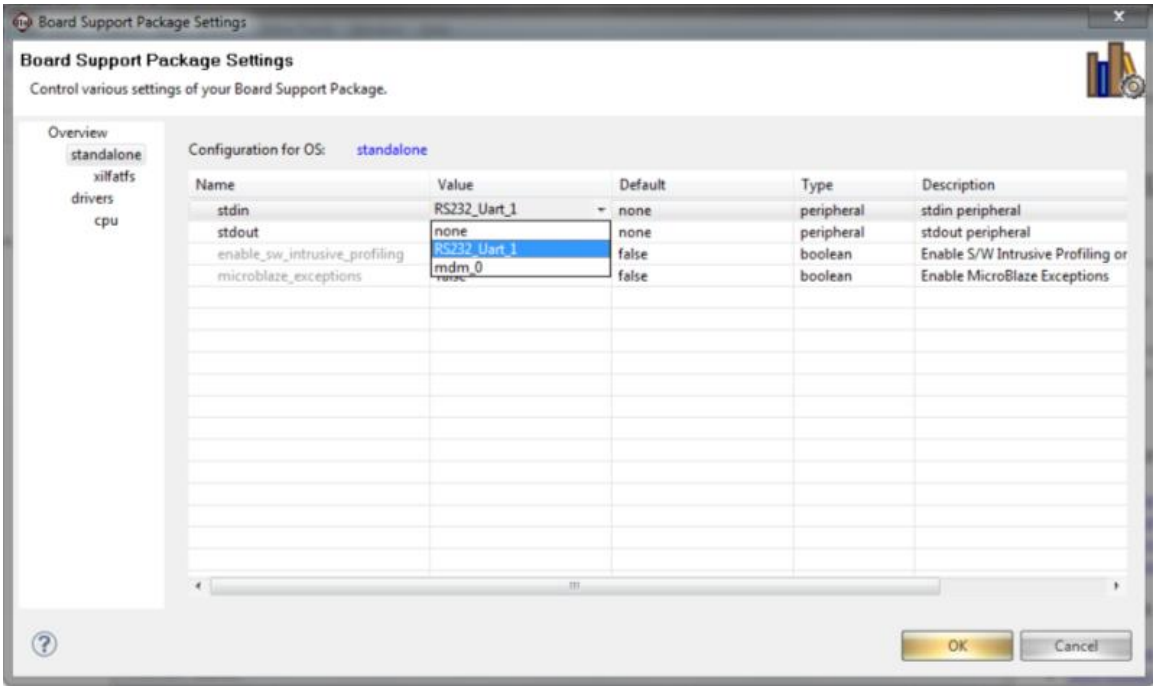


Figura. 4.24. Configuración RS-232 y MDM

- **Configuración SYSACE (XilFATFS)**

El controlador que habilita el uso del módulo SYSACE se llama XilFATFS y también se encuentra dentro de la configuración del BSP (Ver Figura. 4.25). La activación de este sistema permite manipular archivos contenidos dentro de una compact flash, ello implica la creación de carpetas, directorios y la lectura/escritura de archivos por medio de la librería *sysace_stdio.h* (Xilinx, OS and Libraries Document Collection, 2011).

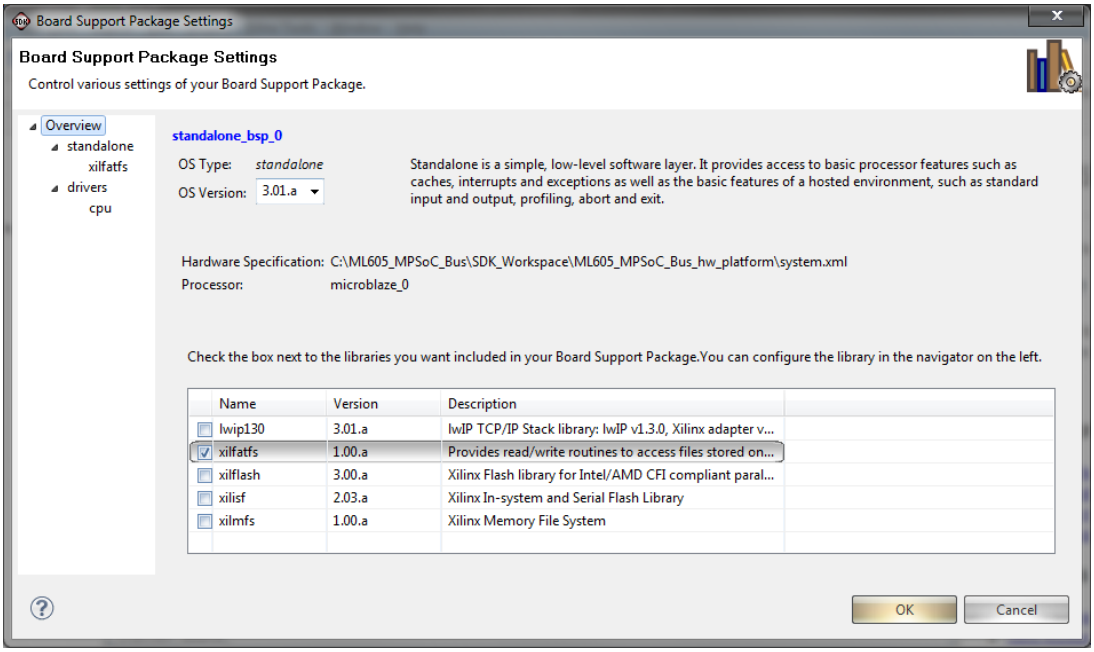


Figura. 4.25. Habilitación del XilFATFS

Después de habilitar el controlador XilFATFS, se realizan modificaciones para poder hacer uso de la librería *sysace_stdio.h* de acuerdo a la Figura. 4.26.

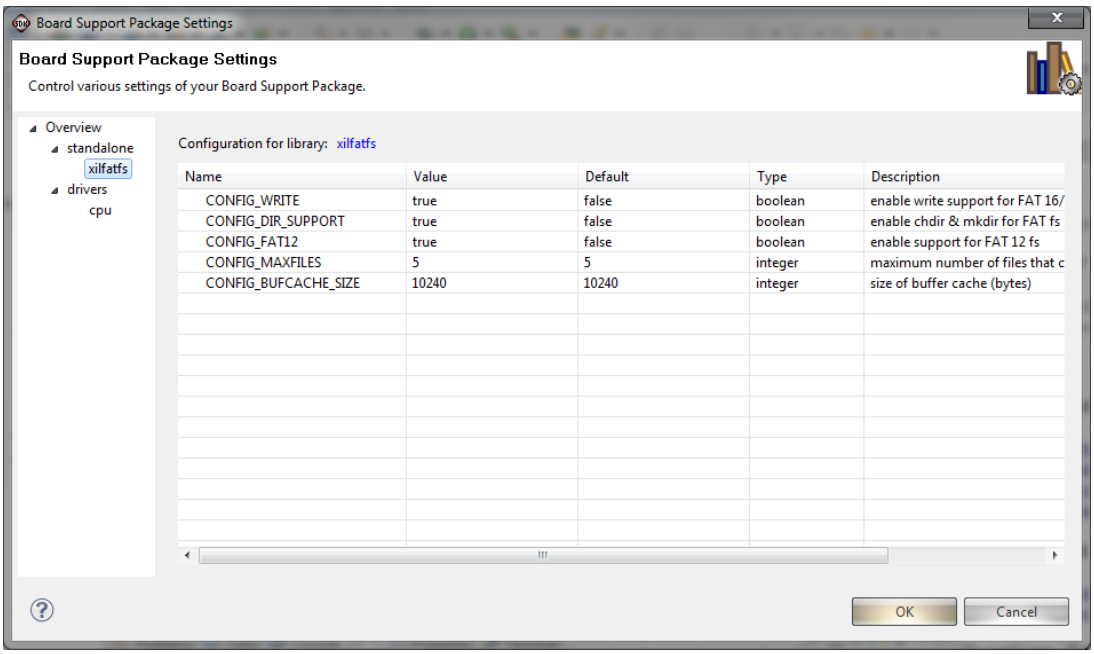


Figura. 4.26. Configuración del XilFATFS

Otra forma de configurar el XilFATFS es desde el archivo MSS introduciendo las siguientes líneas de código:

```
BEGIN LIBRARY
 parameter LIBRARY_NAME = xilfatfs
 parameter LIBRARY_VER = 1.00.a
 parameter CONFIG_WRITE = true
 parameter CONFIG_DIR_SUPPORT = true
 parameter CONFIG_FAT12 = true
 parameter CONFIG_MAXFILES = 5
 parameter CONFIG_BUFCACHE_SIZE = 10240
 parameter PROC_INSTANCE = microblaze_0
END LIBRARY
```

Como existen cuatro procesadores, este proceso deber repetirse en cada uno de los BSP generados, tomando en cuenta que el parámetro PRO_INSTANCE cambiará según el BSP en donde se encuentre el procesador MicroBlaze. (Xilinx, OS and Libraries Document Collection, 2011)

- **Carpeta y Archivos de Programación**

La carpeta de proyecto se crea haciendo clic en la pestaña File/New/Xilinx C Project, la Figura. 4.27 muestra la ventana que se despliega, en donde se puede escoger varios programas ya pre-establecidos. Para el caso de una nueva aplicación se selecciona la segunda opción *Empty Application* y el procesador con el cual se trabajará.

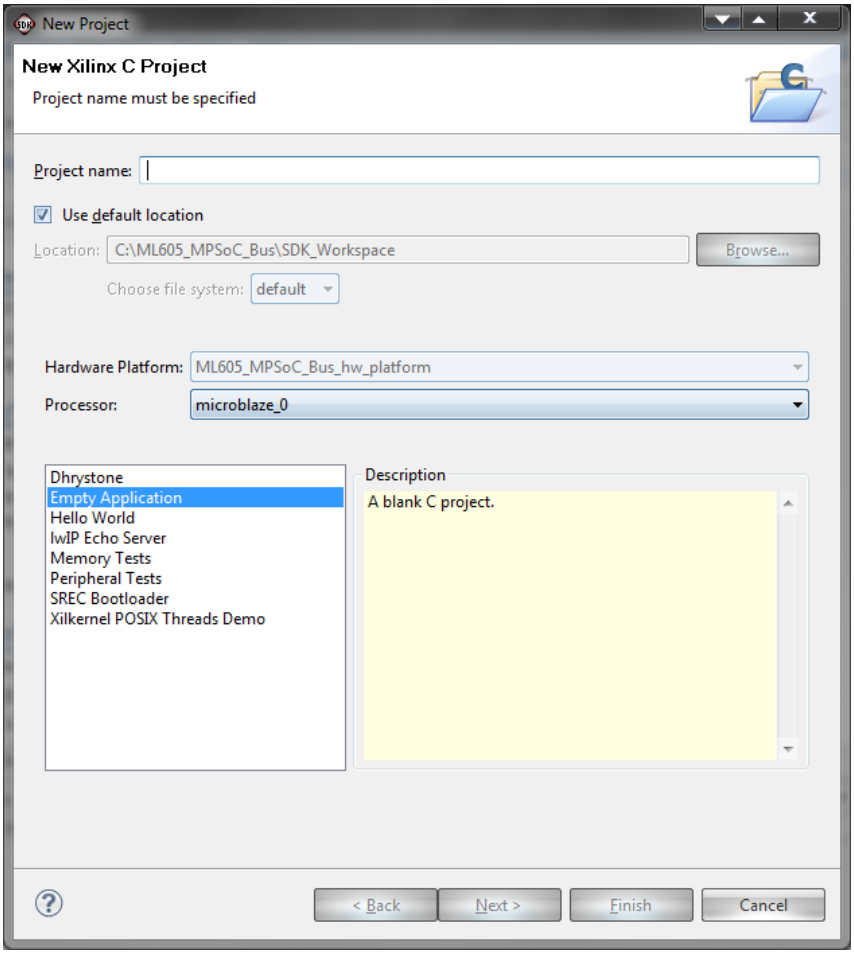


Figura. 4.27. Nueva Carpeta de Proyecto

Lo anteriormente mencionado genera una carpeta con algunos archivos a partir del BSP. Para crear un archivo nuevo de programación en C, debe darse clic derecho sobre la carpeta de proyecto y seleccionar New/Source File. El archivo que se crea está vacío por lo que se hace necesario hacer un llamado a las siguientes librerías:

```
#include "stdlib.h"
#include "math.h"
#include "xenv_standalone.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xil_cache.h"
#include "xuartns550_l.h"
#include "xsysace.h"
#include "sysace_stdio.h"
#include "fsl.h"
#include "delay.h"
```

```
#include "xtft.h"
#include "xtmrctr.h"
```

Estas librerías pueden aumentar según la necesidad de la aplicación y de los componentes creados o añadidos en el XPS. El grupo de librerías mostradas anteriormente son las que se emplean para ejecutar la aplicación de esteganografía. Para una aplicación de comunicación solo se necesitan las siguientes:

```
#include "xil_cache.h"
#include "xenv_standalone.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xuartns550_l.h"
#include "fsl.h"
```

La librería más relevante es la *xparameters.h*, la cual posee toda la información para la puesta en marcha de una aplicación. El resto de librerías habilitan el uso de periféricos, memorias y comunicación RS-232, este último para el caso del MicroBlaze 0 a través de la librería *xuartns550_l.h*. (Cadena & Mollocana, 2012)

4.6.2. Asignación de Memoria Utilizando Linker Script

La asignación de memoria para un archivo ejecutable (ELF), se realiza por medio de un archivo especial denominado Linker Script. Este archivo evita que los ELF entren en conflicto cuando se trabaja con arquitecturas de multiprocesadores (Asokan, 2007) (Tan, 2010). Evitando de esta forma que un archivo ELF trate de ocupar los registros de memoria de otro archivo del mismo tipo.

Procesador	Módulo de Memoria	DDR3 512MB
MicroBlaze 0	Base Address	0xA0000000
	Stack Size	0xFA000
	Heap Stack	0xFA000
MicroBlaze 1	Base Address	0xA1000000
	Stack Size	0xFA000
	Heap Stack	0xFA000
MicroBlaze 2	Base Address	0xA2000000
	Stack Size	0xFA000
	Heap Stack	0xFA000
MicroBlaze 3	Base Address	0xA3000000
	Stack Size	0xFA000
	Heap Stack	0xFA000

Tabla. 4.5. Asignación de memoria

En la Tabla. 4.5, se observa la asignación de memoria que se considerada para los cuatro procesadores. También se muestra el tamaño del Stack y Heap los cuales se crean con un valor estándar de 1KB, pero para la aplicación que se ejecuta en este proyecto, el valor se incrementa a 1MB.

4.6.3. Comunicación Entre Procesadores

Para establecer la comunicación entre los cuatro procesadores se utiliza la librería *fsl.h*, y se emplea las siguientes instrucciones para transmisión y recepción de mensajes (Xilinx, OS and Libraries Document Collection, 2011):

```
putfslx(variable, puerto, bandera)
getfslx(variable, puerto, bandera)
```

En donde:

Variable, correspondes al mensaje que se envía o recibe.

Puerto, es un valor entero que identifica al dispositivo de entrada o salida.

Bandera, corresponde a un grupo de posibles condiciones que controlan el flujo de datos.

Otra instrucción que se emplea en la comunicación, para supervisar la existencia de datos presentes en el FSL es la siguiente:

```
fsl_isinvalid(valido);
```

La cual responde en la variable *valido*, 1 si hay datos disponibles en el bus o 0 si no los hay. (Xilinx, OS and Libraries Document Collection, 2011)

Una vez programada la comunicación entre los cuatro procesadores, utilizando las instrucciones anteriormente mencionadas, y con su respectiva aplicación se cumple con una parte del proyecto. Con la arquitectura de cuatro procesadores interconectados por buses se obtiene un modelo a ser comparado con el nuevo modelo de comunicación del cual se hablará en el siguiente capítulo y posteriormente se tratará la aplicación de Esteganografía desarrollada para ambas arquitecturas.

CAPÍTULO 5

IMPLEMENTACIÓN DE LA ARQUITECTURA NOC SOBRE FPGA

En el capítulo anterior se explicó el diseño de un MPSoC comunicado por buses FSL. El presente capítulo trata sobre la implementación de una arquitectura de comunicación NoC que se integrará a los procesadores MicroBlaze dentro de un nuevo proyecto de MPSoC, cuya integración será finalizada en el **Capítulo 6**.

5.1. SELECCIÓN DE LA ARQUITECTURA NOC

La NoC que se utilizó para el desarrollo del presente proyecto de tesis se llama Hermes, la misma que fue diseñada y desarrollada por la Facultad de Informática de la Pontificia Universidad Católica de Rio Grande do Brasil. A continuación se especifica su funcionamiento.

5.1.1. NoC Hermes

Hermes fue creada en 2003, usa una topología escalable 2D-Mesh con algoritmo de enrutamiento XY y hace uso del modelo OSI (Open System Interconnection) para el manejo de datos. El modelo OSI es el estándar para toda

comunicación de red creada por la Organización Internacional para la Normalización (ISO) que describe el proceso de transmisión de información por medio de capas.

Para el caso de una NoC Hermes se utilizan fundamentalmente las capas inferiores del modelo OSI (ver Tabla. 5.1), las cuales son: Física, Enlace de Datos, Red y Transporte. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

7	Aplicación
6	Presentación
5	Sesión
4	Transporte
3	Red
2	Enlace de Datos
1	Física

Tabla. 5.1. Modelo OSI

- **Modelo OSI aplicado en NoC Hermes**

Capa Física: es la encargada de dar soporte en la comunicación física de una secuencia determinada de bits. En un computador de escritorio, el medio físico es el cable que de conexión a internet. Caso similar ocurre en una NoC, en donde el medio físico está dado por las interconexiones eléctricas entre componentes del sistema intrachip, representados de igual manera por cables.

La Figura. 5.1 ilustra la capa física de NoC Hermes interconectando dos switches. En esta NoC las conexiones se especifican en la Tabla. 5.2, en donde se puede leer las funciones que desempeña cada cable para la interconexión de Hermes.

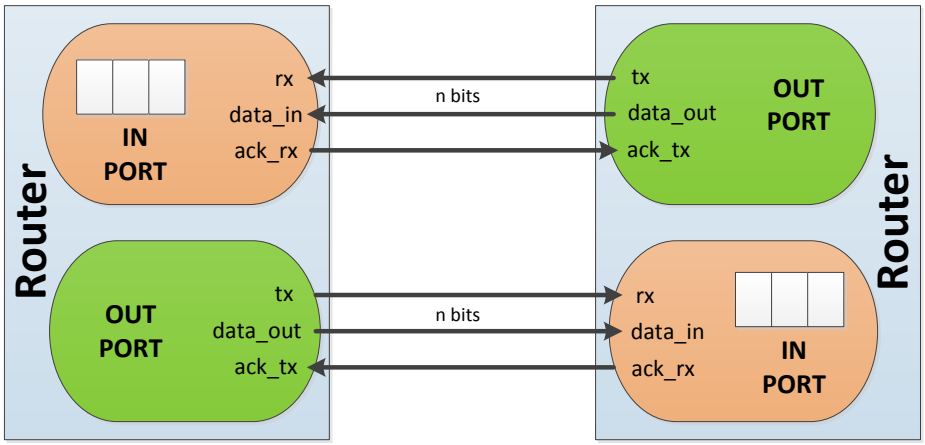


Figura. 5.1. Capa Física en NoC Hermes

Capa de Enlace de Datos: se encarga de transferir los datos desde el nodo de origen hacia el nodo de destino haciendo uso de un método de un control de flujo. El control de flujo que se aplica en NoC Hermes se denomina Handshake, el cual se encarga del correcto envió y recepción de los datos a través de los cables Rx, Tx, Ack_rx y Ack_tx que se los aprecia en la Tabla. 5.2. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

Switch Hermes	
Cable	**Descripción**
Rx	Indica dato disponible
Data_in	Dato a ser recibido
Ack_rx	Confirmación de dato
Tx	Indica dato disponible
Data_out	Dato a ser transmitido
Ack_tx	Confirmación de dato

Tabla. 5.2. Descripción de Conexiones

Capa de Red: es la encargada de orientar los datos hacia su destino y entregarlos utilizando protocolos de comunicación y enrutamiento de paquetes. A escala de microchip la técnica de enrutamiento implementada en NoC Hermes se llama Wormhole y hace uso del algoritmo XY para orientar a los paquetes. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

Capa de Trasporte: establece comunicación entre el nodo de origen y el nodo de destino, y se encarga de supervisar el tamaño de los paquetes con el objetivo de que no existan errores en las capas inferiores. En el presente proyecto, esta capa es implementada por medio de los procesadores conectados a NoC Hermes.

- **Estructura del Switch Hermes**

El Switch es el componente esencial en el enrutamiento y transferencia de los paquetes. La Figura. 5.2 muestra el Switch Hermes, el cual consta de un controlador lógico y de cinco puertos bidireccionales: North (Norte), West (Oeste), South (Sur), East (Este) y Local.

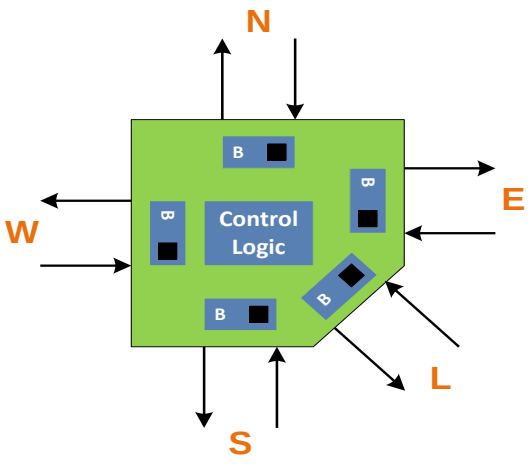


Figura. 5.2. Estructura del Switch Hermes

Todos los puertos son bidireccionales y poseen internamente un Buffer (interpretado por una B en la Figura. 5.2) que es utilizado para mantener la información temporalmente hasta su transmisión. De estos puertos solo el Local se conecta para interactuar con otro IP-Core, en tanto que los demás se interconectan con otros switches para formar una topología de red.

El Control Lógico de Hermes maneja la técnica de ruteo (algoritmo XY), el control de flujo (Handshake) y la técnica de conmutación de paquetes (Wormhole). Esta última técnica manipula los paquetes al dividirlos en flits, lo cual ya se ha mencionado en el **Capítulo 2**. Los flits se caracterizan en esta NoC, debido a que se puede manipular su tamaño antes de sintetizar el hardware final. Dicha modificación afecta a toda la estructura del paquete, que está conformado por una cabecera, en donde se posiciona el algoritmo XY, y el tamaño del cuerpo del paquete. Cabe destacar que cada switch tiene una única dirección de red, dada por el algoritmo XY (X para el eje horizontal e Y para el eje vertical). (Moraes, Vilar, Viera, Möller, & Copello, 2003)

5.2. GENERACIÓN DE UNA ARQUITECTURA NOC

La generación en código VHDL de NoC Hermes se realiza a través de la plataforma ATLAS. ATLAS es un entorno Java desarrollado por GAPH (Hardware Desing Support Group – PUCRS Facultad de Informática, Porto Alegre, Brasil), dedicado a la generación y evaluación de Networks on Chip. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

ATLAS posee 5 herramientas para el diseño, implementación y análisis de una NoC Hermes. Las mismas que se utilizan en las etapas de desarrollo

exclusivo para un proyecto de NoC Hermes y se encuentran detalladas en la Figura. 5.3.

- NoCGeneration
- TrafficGeneration
- NocSimulation
- TrafficEvaluation
- PowerEvaluation

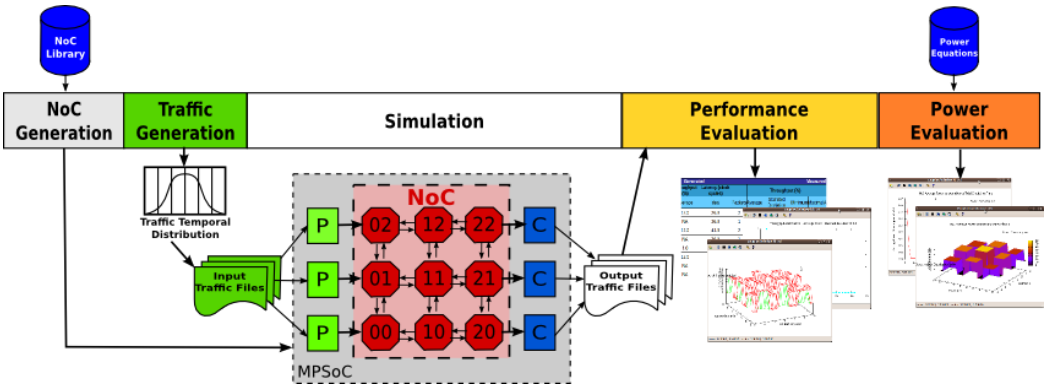


Figura. 5.3. Estructura de la Plataforma ATLAS

- **Noc Generation**

Esta herramienta permite configurar una NoC Hermes de acuerdo a varios parámetros específicos:

- Ancho de banda del canal.
- Dimensión
- Cantidad de canales virtuales
- Estrategias de control de flujo.

- **Traffic Generation**

En esta herramienta se configura distintos escenarios de tráfico para el algoritmo XY en una NoC Hermes, en los cuales se establece la frecuencia de operación, número de paquetes, dimensión del paquete y tasa de transferencia.

- **Noc Simulation**

En la Simulación, se inyectan los datos de tráfico generados por la herramienta Traffic Generation, directamente en la Network on Chip, produciéndose una comunicación efectiva entre cada uno de los procesadores del Chip.

- **Traffic Evaluation**

En esta herramienta es posible generar tablas, gráficas e informes para analizar resultados de desempeño.

- **Power Evaluation**

Esta herramienta está presente únicamente en versiones oficiales, bajo licencia exclusiva del desarrollador, en la misma se permite develar reportes de desempeño de eficiencia energética y consumo de potencia. Para el presente

proyecto se utilizará herramientas propias de la marca Xilinx, especificadas en el **Capítulo 3**.

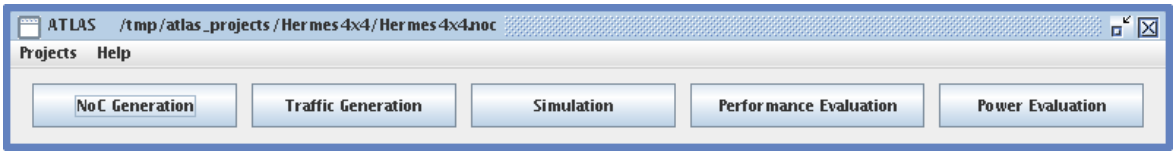


Figura. 5.4. Captura de Pantalla de las Herramientas de ATLAS

Previamente a la generación de la NoC es fundamental configurar la arquitectura según a las necesidades del diseñador.

5.2.1. Configuración Arquitectura NoC Hermes

Mediante ATLAS podemos realizar una configuración de Noc Hermes, en la Figura. 5.5 se puede apreciar tres herramientas para realizar la posterior generación:

1. Barra de Configuración
2. Área de Visualización
3. Botón Generador de NoC Hermes

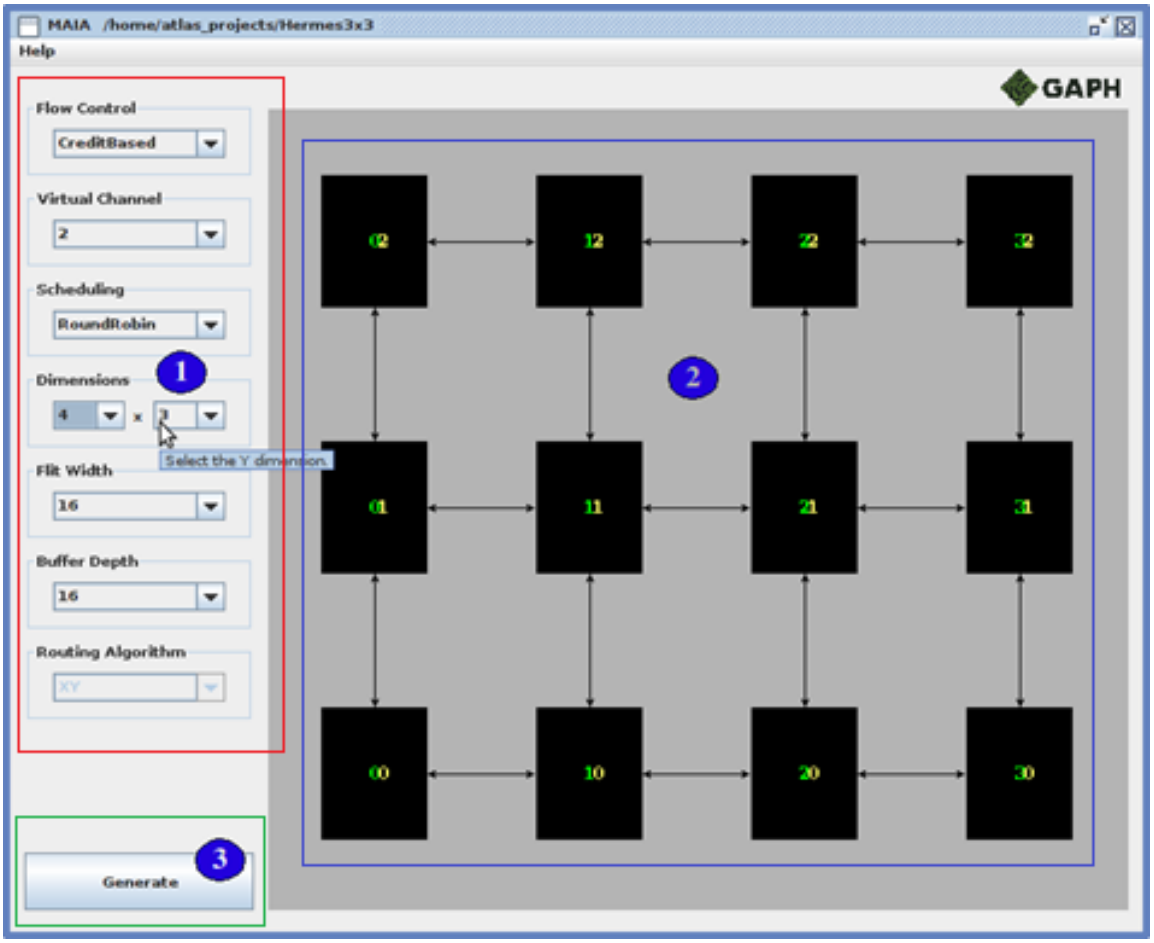
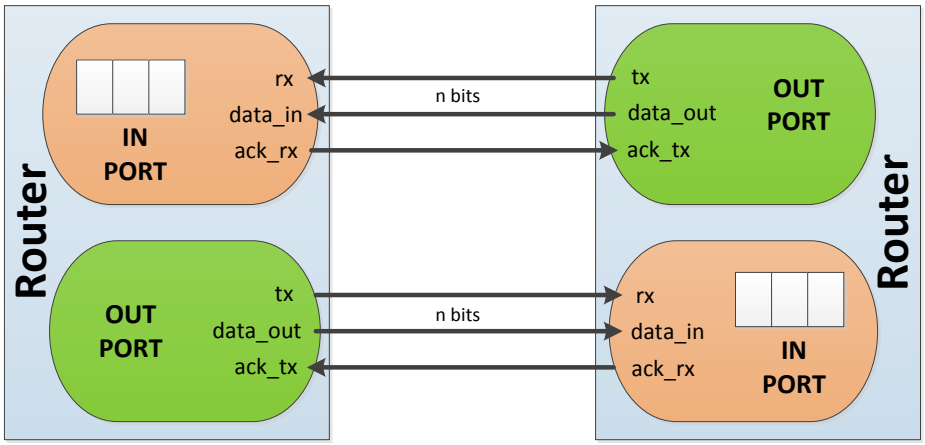


Figura. 5.5. Generación de Hermes mediante ATLAS

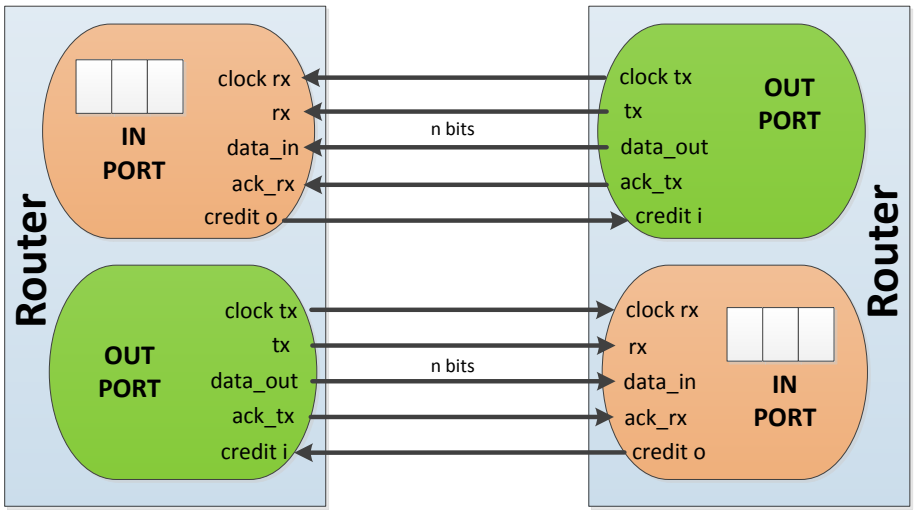
5.2.1.1. Barra de Configuración

- **Flow Control**

El control de flujo es el mecanismo que permite al transmisor, determinar si el receptor está apto para recibir datos. ATLAS proporciona dos tipos de control de flujo para Hermes: a) Handshake y b) Credit Based, los mismos que se pueden apreciar en el Figura. 5.6.



(a)



(b)

Figura. 5.6. Tipos de control de flujo en Hermes. (a) Handshake y (b) Credit Based

En el control de flujo Handshake, cuando un router A, desea transmitir información a un router B, el router A activa la señal de Tx y pone los datos en data_out. Cuando el router B percibe la señal activada Rx, almacena los datos en data_in y activa la señal ack_rx, lo que indica la recepción de información. Este protocolo de comunicación es asíncrono y cada flit consume al menos dos ciclos de reloj.

En el control de flujo Credit Based, cuando el Router A desea transmitir información al Router B, en primer lugar verifica si el canal seleccionado para la

comunicación credit_i esta activada. En caso de que este activo, el Router A, pone los datos en la señal data_out y activa la señal Tx. Cuando el Router B, recibe la señal Rx, almacena los datos en la señal data_in y disminuye el número de créditos de comunicación. El router A puede transmitir mientras la señal credit_i este activa. El número de créditos se aumenta, cuando un dato es consumido por el Router B. Este protocolo de comunicación se sincroniza mediante la señal clock_tx y cada dato consume al menos un ciclo de reloj.

- **Virtual Channel**

Un canal físico puede ser multiplexado por N canales virtuales, el flujo en dicho canal físico debe ser arbitrado de acuerdo a la existencia de cada canal virtual. Únicamente en el control de flujo Credit Based se puede aplicar el uso de canales virtuales, ver Figura. 5.7, por tanto, para la configuración de canales virtuales se especifican 3 opciones:

1. Sin canal virtual
2. Dos canales virtuales
3. Cuatro Canales virtuales

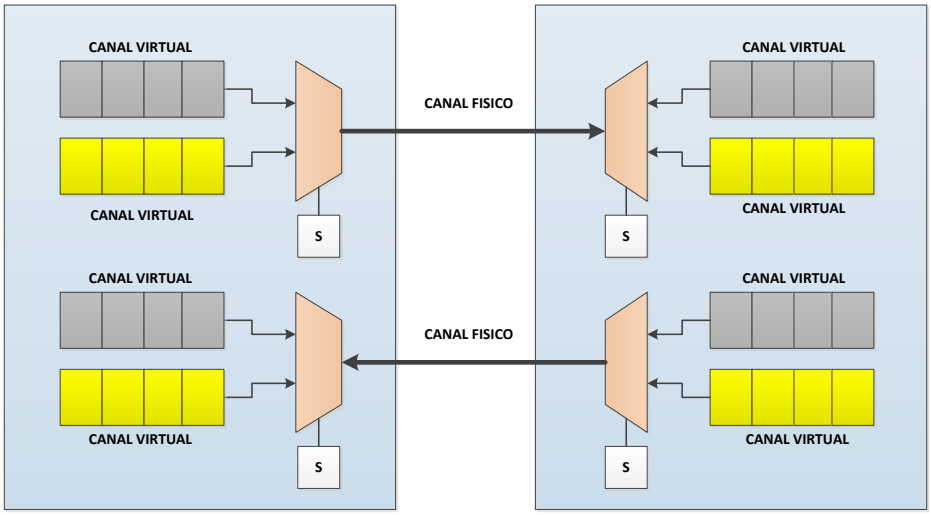


Figura. 5.7. Utilización de dos canales virtuales en control de flujo credit based

- **Scheduling Politic**

Cuando un canal físico se multiplexa por N canales virtuales, es necesario una jerarquización de los mismos. El esquema de jerarquización divide el ancho de banda del canal físico entre todos los canales virtuales mediante dos opciones: a) Round Robin y b) Priority

La jerarquización Round Robin divide el ancho de banda del canal físico en partes iguales para todos los canales virtuales que tienen datos para transmitir. Mientras que en la jerarquización Priority, cada canal virtual recibe una prioridad estática. Cuando dos canales disponen datos para la transmisión, el canal con la prioridad más alta, ocupa el 100% del ancho de banda del canal físico, aunque otros canales hayan solicitado el uso del canal previamente.

- **Network Dimension**

Dos parámetros componen el dimensionamiento de la NoC Hermes, uno variable para X y otro variable para Y, pudiendo ser valores comprendidos entre de 2 y 16. No necesariamente el dimensionamiento de la NoC Hermes es cuadrado.

- **Communication Channel Bandwidth**

El ancho de banda del canal define el número de bits que se transmite y/o recibe por un canal físico. En Hermes, este número corresponde a un tamaño de flit de 8, 16, 32 ó 64 bits.

- **Buffer Depth**

El dimensionamiento del buffer, define el número de flits que el buffer puede almacenar. ATLAS, permite la elección de dimensionamiento entre 4, 8, 16 ó 32. Se recomienda un dimensionamiento igual a 8 ó 16 flits para no interferir en el rendimiento de la Noc Hermes.

- **Routing Algorithm**

El algoritmo de ruteo define el camino tomado para la transmisión de un paquete, entre el Switch de origen y el Switch de destino. Actualmente, Hermes

solo admite un algoritmo XY, sin embargo se prevé el desarrollo de otros algoritmos como el West-First.

- **Testbench**

Esta herramienta permite generar escenarios de prueba para SystemC. Estos escenarios de prueba consisten en la lectura de archivos de entrada y escritura de archivos de salida. Cada archivo de entrada posee información que se transmite entre procesadores, y cada archivo de salida posee la información que se entrega al procesador. Para el desarrollo de este proyecto, se utilizó la herramienta ISim Simulator, en la misma que se manipuló los distintos escenarios para archivos de entrada y se analizó la información resultante en los archivos de salida.

El ISim Simulator permite la simulación de escenarios de tráfico empleando VHDL o verilog en un solo entorno de depuración, para el presente proyecto se utilizó únicamente VHDL y el análisis de sus resultados se especifica en el **Capítulo 9**.

5.2.1.2. Área de Visualización

En el Área de Visualización se puede apreciar un esquema gráfico de la configuración del dimensionamiento de una NoC Hermes, dado por las coordenadas X e Y.

5.2.1.3. Botón Generador de NoC Hermes

Una vez descrita las diferentes formas de configurar la plataforma ATLAS, se procede a la generación en VHDL de la arquitectura de red presionando el Botón Generate. Para información más detallada revisar el Anexo 4.

5.2.2. Generación de Código VHDL de la Arquitectura NoC Hermes a través de ATLAS

Después de configurar la NoC Hermes mediante ATLAS, ver Tabla. 5.3., nuevos archivos y carpetas son generados en el directorio raíz de ATLAS, los mismos que se detallan en Figura. 5.8, Tabla. 5.4 y Figura. 5.9, Tabla. 5.5.

Parámetros	Hermes
Topología	2D-Mesh
Control de Flujo	Handshake
Canales Virtuales	2
Dimensión Flit	16
Dimensión Buffer	16
Algoritmo de Ruteo	XY

Tabla. 5.3. Parámetros de configuración en ATLAS para generar una NoC Hermes

Carpeta / Archivo	Descripción
Carpeta NOC	Carpeta que contiene los archivos VHDL que describen a la NoC Hermes generada por ATLAS
Carpeta SC_NoC	Carpeta que contiene los archivos que describen los escenarios de simulación descritos en System C
Carpeta Traffic	Carpeta que contiene los archivos que describen la generación de trafico configurado en ATLAS
Archivo Prueba_Atlas.noc	Archivo de extensión *.noc que describe la configuración de la NoC Hermes generada por ATLAS
Archivo simulate.do	Archivo de extensión *.do que describe los escenarios de simulación generados por ATLAS
Archivo topNoC.vhdl	Archivo VHDL que contiene el Top con las instancias de toda la NoC Hermes generada por ATLAS

Tabla. 5.4. Descripción de Carpetas y Archivos generados por ATLAS

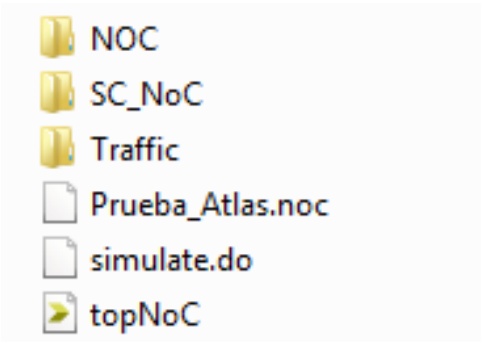


Figura. 5.8. Captura de Pantalla de los Archivos y Carpetas generados por ATLAS

Archivos VHDL	Contenido
Hermes_buffer	Memoria FIFO.
Hermes_package	Librería que contiene parámetros de control para el funcionamiento de la NoC.
Hermes_switchcontrol	Entidad que maneja el Algoritmo XY
NOC	La NoC propiamente dicha
ChaveBL	Switch conformado por las entidades Hermes_buffer y Hermes_switchcontrol. Tiene asignado el Algoritmo XY 0000.
ChaveBR	Switch conformado por las entidades Hermes_buffer y Hermes_switchcontrol. Tiene asignado el Algoritmo XY 0100.
ChaveTL	Switch conformado por las entidades Hermes_buffer y Hermes_switchcontrol. Tiene asignado el Algoritmo XY 0001.
ChaveTR	Switch conformado por las entidades Hermes_buffer y Hermes_switchcontrol. Tiene asignado el Algoritmo XY 0101.

Tabla. 5.5. Descripción de Archivos VHDL contenidos en la carpeta NOC

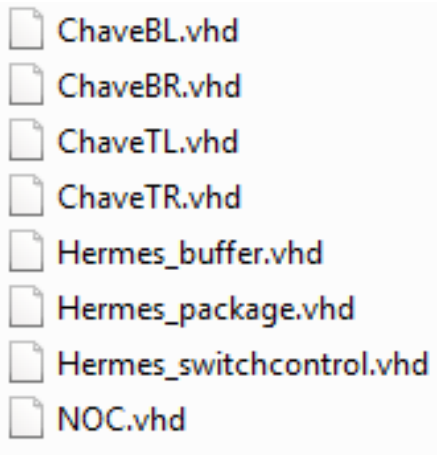


Figura. 5.9. Captura de Pantalla de los Archivos VHDL contenidos en la carpeta NOC

La carpeta SC_NoC, contiene los testbenchs descritos en SystemC (Ver Figura. 5.10).

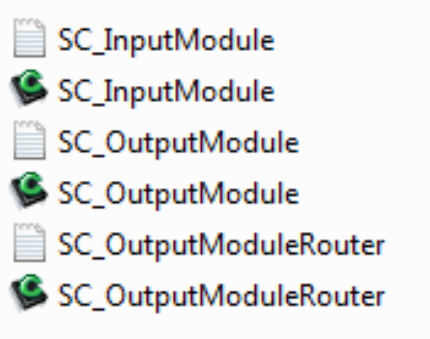


Figura. 5.10. Captura de Pantalla de los Archivos contenidos en la carpeta SC_NoC

5.2.3. Implementación de Código Generado sobre FPGA

Una vez generado el código VHDL por ATLAS, se procede a sintetizarlo sobre la FPGA. Para ello se debe crear un nuevo proyecto en la plataforma de desarrollo Xilinx ISE Design Suite 13.2. La configuración del proyecto mostrado en la Figura. 5.11 muestra los parámetros que deben seleccionarse para una FPGA ML605 Virtex 6.

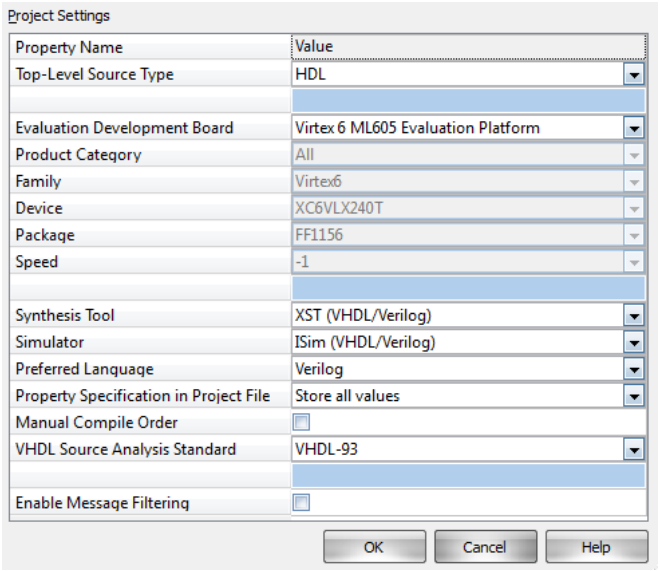


Figura. 5.11. Parámetros para Nuevo Proyecto de una Virtex 6

Con el proyecto nuevo ya creado, se añaden los archivos en VHDL generados por ATLAS para la NoC Hermes que se encuentran dentro de la carpeta NOC, así como el archivo topNoC. Para visualizar el código generado revisar el Anexo 7.

Luego que todos los archivos han sido importados con éxito, se procede a eliminar las líneas de código que permiten la simulación en SystemC, dado que, como se dijo anteriormente para el proceso de simulación se utilizará ISim Simulator. Las líneas de código que deben ser eliminadas, se muestran como líneas comentadas en la Figura. 5.12.

```
161
162     -- the component below, router_output, must be commented to simulate without SystemC
163  -- router_output: Entity work.outmodulerouter
164  -- port map(
165  --    clock => clock,
166  --    reset => reset,
167  --    tx_r0p0   => txN0000(EAST),
168  --    out_r0p0  => data_outN0000(EAST),
169  --    ack_ir0p0 => ack_txN0000(EAST),
170  --    tx_r0p2   => txN0000(NORTH),
171  --    out_r0p2  => data_outN0000(NORTH),
172  --    ack_ir0p2 => ack_txN0000(NORTH),
173  --    tx_r1p1   => txN0100(WEST),
174  --    out_r1p1  => data_outN0100(WEST),
175  --    ack_ir1p1 => ack_txN0100(WEST),
176  --    tx_r1p2   => txN0100(NORTH),
177  --    out_r1p2  => data_outN0100(NORTH),
178  --    ack_ir1p2 => ack_txN0100(NORTH),
179  --    tx_r2p0   => txN0001(EAST),
180  --    out_r2p0  => data_outN0001(EAST),
181  --    ack_ir2p0 => ack_txN0001(EAST),
182  --    tx_r2p3   => txN0001(SOUTH),
183  --    out_r2p3  => data_outN0001(SOUTH),
184  --    ack_ir2p3 => ack_txN0001(SOUTH),
185  --    tx_r3p1   => txN0101(WEST),
186  --    out_r3p1  => data_outN0101(WEST),
187  --    ack_ir3p1 => ack_txN0101(WEST),
188  --    tx_r3p3   => txN0101(SOUTH),
189  --    out_r3p3  => data_outN0101(SOUTH),
190  --    ack_ir3p3 => ack_txN0101(SOUTH));
```

Figura. 5.12. Archivo NOC.vhd

Una vez realizada la modificación ya se puede implementar el nuevo IP-Core, cuyo resultado se aprecia en Figura. 5.13. Cabe señalar que anteriormente en el **Capítulo 2**, se hizo mención que una arquitectura de red consta de tres elementos: un Elemento de Red (NE), una Interfaz de Red (NI) y una Elemento de Procesamiento (PE). De los mencionados hasta el momento se cuenta con el Elemento de Procesamiento (PE, MicroBlaze) y el Elemento de Red (NE, Switch

Hermes), por lo tanto hace falta una Interfaz de Red (NI) que permita enviar o receptar los paquetes de los sistemas involucrados en la arquitectura de red, la misma que se desarrollará en el capítulo siguiente.

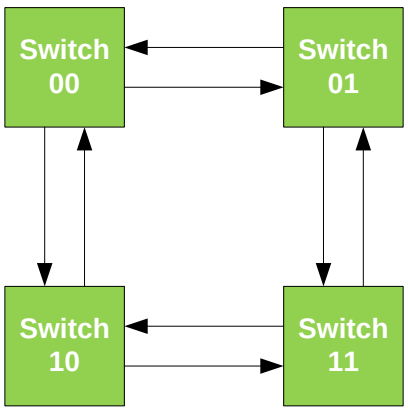


Figura. 5.13. Hermes en Topología 2D-Mesh

CAPÍTULO 6

DISEÑO E IMPLEMENTACIÓN DE UNA INTERFAZ DE RED

6.1. DISEÑO DE LA INTERFAZ DE RED

Antes de explicar el diseño propiamente dicho, se empieza analizando la operación de los IP-Cores relacionados (Bus FSL y NoC Hermes). Estos dispositivos al final se integran por intermedio de una interfaz de red, completando de esta forma la arquitectura de red. El dispositivo final será sintetizado e implementado dentro de una arquitectura MPSoC con el objetivo de transmitir flujos de datos reales a través de NoC Hermes.

6.1.1. Operación Fast Simplex Link (FSL)

FSL es un bus unidireccional de punto a punto utilizado para aplicaciones desarrolladas en FPGA. La interfaz FSL puede ser habilitada desde el propio MicroBlaze y se la utiliza para transferir datos hacia y desde los archivos de registro del procesador al hardware implementado. (Xilinx, LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11e), 2011)

Los puertos de entrada y salida del bus FSL que son utilizados en el diseño se especifican en la Tabla. 6.1.

Fast Simples Link (FSL)			
Maestro	**I/O**	**Esclavo**	**I/O**
FSL_M_DATA	In	FSL_S_DATA	Out
FSL_M_WRITE	In	FSL_S_READ	In
FLS_M_FULL	Out	FSL_S_EXISTS	Out

Tabla. 6.1. Puertos del FSL necesarios para el diseño

De la Tabla. 6.1 se analiza el modo de operación, empezando por el Maestro, por medio de la Figura. 6.1. La forma de transmisión de datos se la realiza cuando se activa en alto el FSL_M_WRITE, y mientras se mantenga así los datos se mantendrán circulando a través del FSL Maestro hasta que el FSL_M_WRITE vuelva a un nivel bajo o se emita una señal en alto por el FSL_M_FULL indicando que el bus está lleno. (Xilinx, LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11e), 2011)

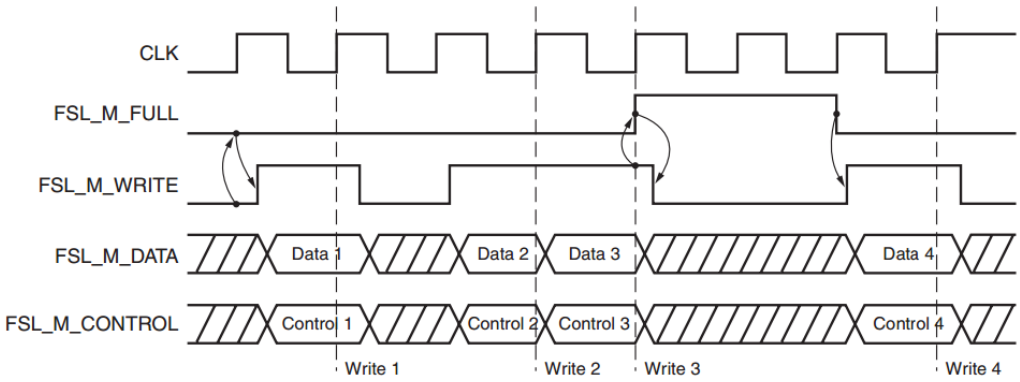


Figura. 6.1. Diagrama de Señales FSL Maestro

Los puertos del FSL Esclavo, mostrados en la Tabla. 6.1, se observan en la Figura. 6.2. Los datos en este caso se trasladan cuando el FSL_S_EXISTS se activa en alto y únicamente bajo esta condición se puede activar el FSL_S_READ, caso contrario si no existen datos en el bus, el FSL_S_READ se mantendrá en un nivel bajo. (Xilinx, LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11e), 2011)

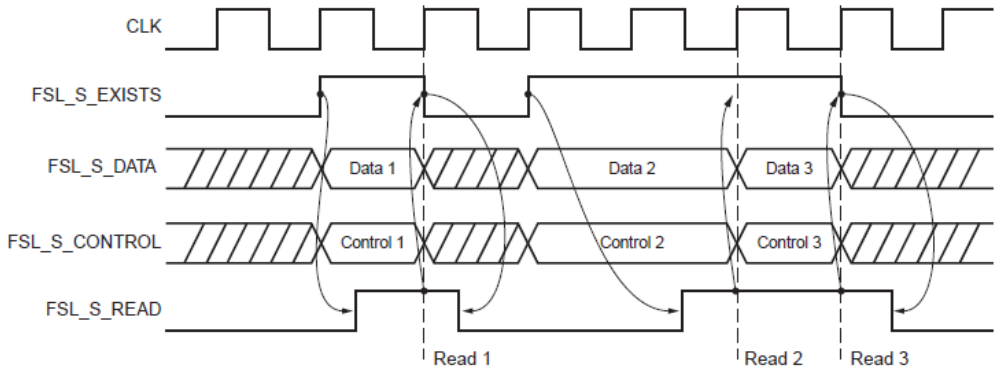


Figura. 6.2. Diagrama de Señales FSL Esclavo

En las figuras anteriores (Figura. 6.1 y Figura. 6.2), existen bits de control dados por los puertos FSL_M_CONTROL y FSL_S_CONTROL, los cuales proveen control adicional para la anotación de los datos que se transmiten. Estos puertos pueden usarse para una interfaz esclava de múltiples propósitos. Por ejemplo decodificar palabras que han sido transmitidas o usar un bit para indicar el inicio o el fin de una transmisión (Xilinx, LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11e), 2011). Dichos puertos tan solo se describen para dar a conocer su función, pero nunca fueron empleados en el diseño de la interfaz de red.

6.1.2. Operación Switch Hermes

La topología de Hermes es una 2D-Mesh que emplea control de flujo por Handshake y técnica de ruteo por algoritmo XY. Hermes transmite información desde un nodo de origen (Figura. 6.3) hacia un nodo de destino (Figura. 6.4) por medio de la conmutación de paquetes a través de la técnica Wormhole. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

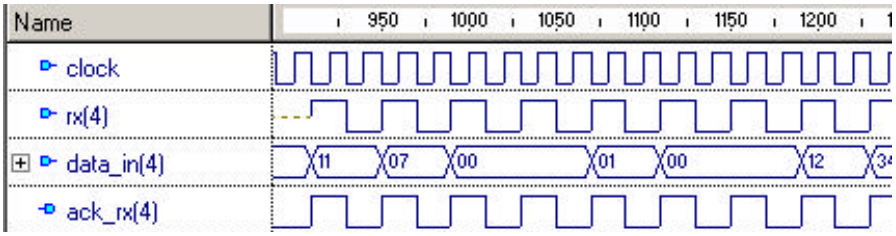


Figura. 6.3. Diagrama de señales Switch 00

El Switch 00 del diagrama de señales en la Figura. 6.3, emplea Handshake a través de rx y ack_rx, en donde cada flanco positivo va acompañado de un flit del paquete de datos por data_in. Este paquete va encabezado por la dirección de destino, que en este caso es 0x11 y con un tamaño de 0x07 flits de 16 bits cada uno. Dicho proceso sucede hasta que el counter_flit haya transmitido todo el paquete de datos. (Moraes, Vilar, Viera, Möller, & Copello, 2003)

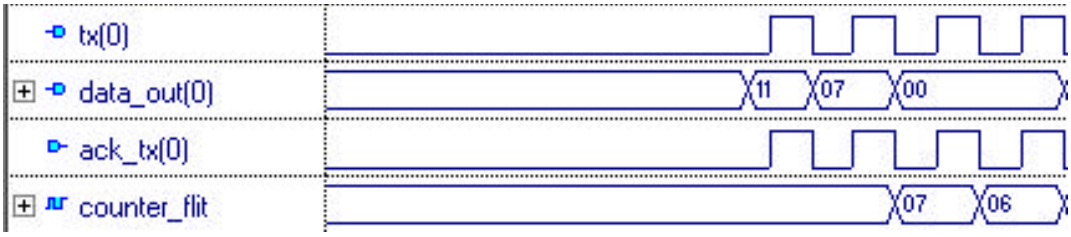


Figura. 6.4. Diagrama de señales Switch 11

Los datos al ser receptados en el nodo de destino presentan el mismo comportamiento, que se describió anteriormente, es decir que en la Figura. 6.4 se observa que tx se activa en alto cada vez que llega un nuevo dato. Del mismo modo la recepción se confirma por medio de ack_tx. La recepción de los flits finaliza una vez que el counter_flit haya receptado todo el paquete (Moraes, Vilar, Viera, Möller, & Copello, 2003). Cabe mencionar que el counter_flit no es un puerto físico, sino que es un control interno que es parte del control de flujo por Handshake.

La distribución de entradas y salidas presentes en el puerto local del Switch Hermes se muestran en la Tabla. 6.2.

Switch Hermes (Local Port)			
Receptor	**I/O**	**Transmisor**	**I/O**
rxloca	In	txlocal	Out
data_inlocal	In	data_outlocal	Out
ack_rxlocal	Out	ack_txlocal	In

Tabla. 6.2. Puertos del Switch Hermes necesarios pata el diseño

Estos puertos son los que se vincularán, junto con los mostrados en la Tabla. 6.1, en el diseño para crear la interfaz de red que interactuará con el bus FSL y con el switch de NoC Hermes.

6.1.3. Diagrama de Estados para el Desarrollo de la Interfaz de Red

Basándose en la explicación anterior, se establecen dos diagramas de estados. Uno para comunicar al MicroBlaze con el Switch y otro para comunicar al Switch con el MicroBlaze de forma unidireccional. En otras palabras, el NI diseñado consta de dos módulos internos que se ilustran en la Figura. 6.5.

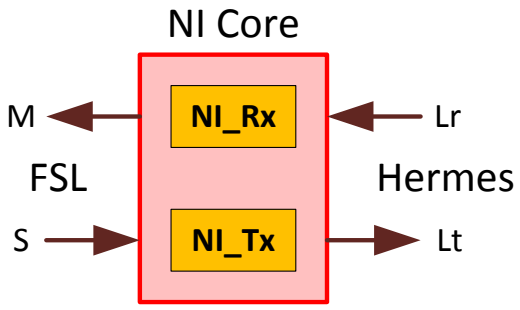


Figura. 6.5. Módulos internos del NI

De donde, NI_Rx y NI_Tx son las máquinas secuenciales desarrollas a partir de los diagramas de estados que se muestran en la Figura. 6.6 y Figura. 6.7 respectivamente, que serán explicados a continuación después de cada figura. Cada uno de estos módulos tiene terminaciones que se conectan al maestro (M) y

esclavo (S) del FSL, y al puerto local de recepción (Lr) y transmisión (Lt) de la NoC.

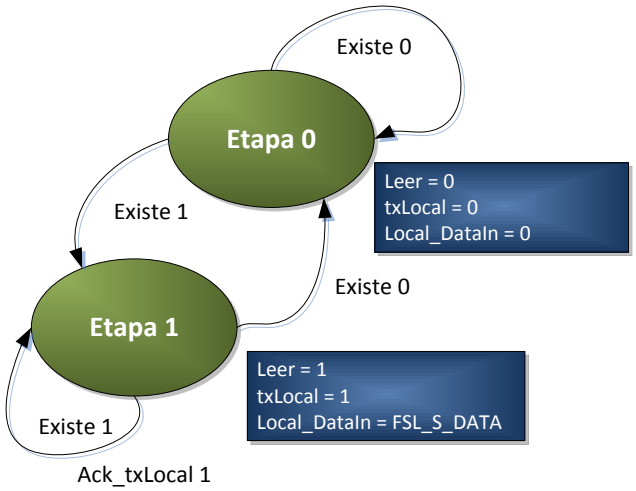


Figura. 6.6. Diagrama de estados para la interfaz de red de transmisión de datos (Tx_FSL_Hermes)

El Transmisor de Datos, permite el paso de información proveniente del MicroBlaze hacia Hermes por medio de la señal FSL_S_EXISTS (Existe) la cual es 0 cuando no hay datos disponible en el bus FSL, razón por la cual permanece en la etapa 0, caso contrario pasa a la etapa 1 disparando la señal de Leer (FSL_S_READ en 1). Se mantendrá en la etapa 1 hasta que la señal de Existe vuelva a 0. Mientras se mantenga en su estado actual, es decir etapa 0 o etapa 1, el resto de señales se comportaran tal y como se aprecian en los recuadros que están a un lado de cada estado.

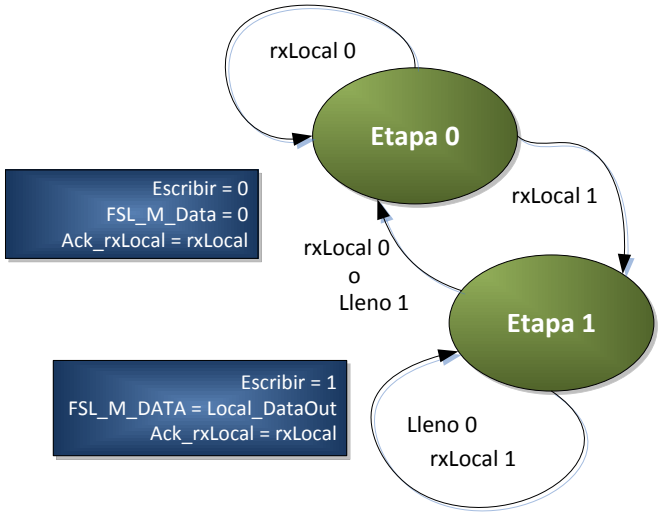


Figura. 6.7. Diagrama de estados para la interfaz de red de recepción de datos (Rx_Hermes_FSL)

El Receptor de Datos, permite el paso de información proveniente de Hermes hacia el MicroBlaze por medio de la señal rxLocal, que es 0 cuando no hay datos disponibles en el Switch de Hermes así que se mantendrá en la etapa 0, caso contrario pasa a la etapa 1 en donde se verifica que el bus no esté saturado por medio de la señal FSL_M_FULL (Lleno) la cual debe mantenerse en 0, permitiendo así que la señal de Escribir se active (FSL_M_WRITE en 1). Se volverá al estado inicial (etapa 0) una vez que txLocal vuelva a ser 0, el resto de señales se comportaran de acuerdo al estado en el que se encuentren tal y como se aprecia en el recuadro adjunto a cada estado.

6.2. IMPLEMENTACIÓN VHDL DE LA INTERFAZ DE RED

Para implementar en código los diseños mostrados a través de los diagramas de estados, explicados anteriormente, se necesita primero tener en claro las entradas y salidas que tendrá el nuevo dispositivo. Por lo tanto la Tabla. 6.3, pone en lista los puertos locales del Switch Hermes y los puertos del FSL, los cuales se entrelazan por medio de la interfaz de red que se creó.

INTERFAZ DE RED							
Entidad de Transmisor				**Entidad de Receptor**			
FSL	**I/O**	**Local Port**	**I/O**	**FSL**	**I/O**	**Local Port**	**I/O**
Data_in	In	Data_out	Out	Data_out	Out	Data_in	In
Exists	In	txLocal	Out	Lleno	In	rxLocal	In
Leer	Out	Ack_txLocal	In	Escribir	Out	Ack_rxLocal	Out

Tabla. 6.3. Puerto Internos de la Interfaz de red Diseñada

Las entradas y salidas, mostradas en la tabla anterior, constituyen los módulos internos que posee la interfaz de red desarrollada. Dichos entidades se ilustran en la Figura. 6.8 y Figura. 6.9.

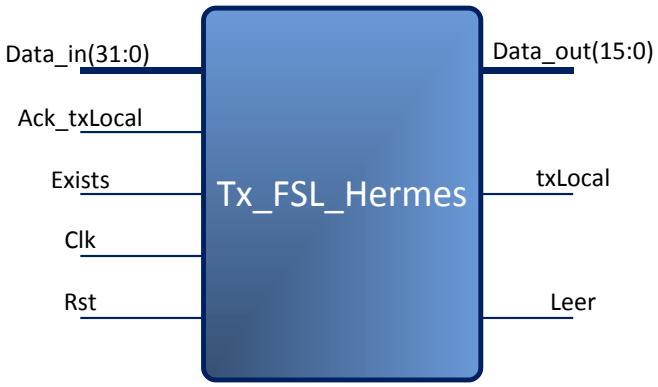


Figura. 6.8. Módulo interno para la Transmisión de Datos

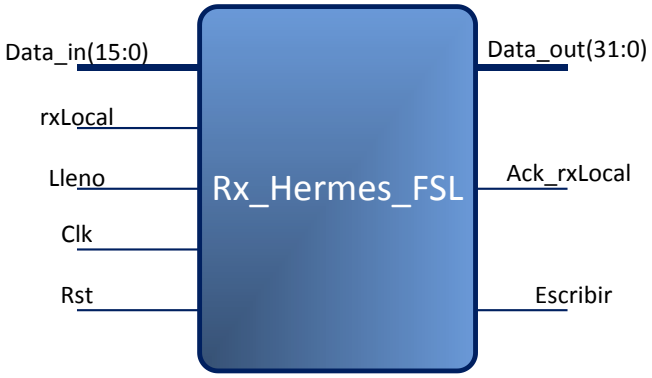


Figura. 6.9. Módulo interno para la Recepción de Datos

Ambos módulos son generados individualmente a partir del código VHDL (utilizando el software ISE 13.2) que puede ser apreciado en el Anexo 8 del presente proyecto de tesis, al igual que el código VHDL implementado para interconectar ambos módulos en un solo IP-Core. Esta implementación resultante se proyecta a través de la Figura. 6.10.

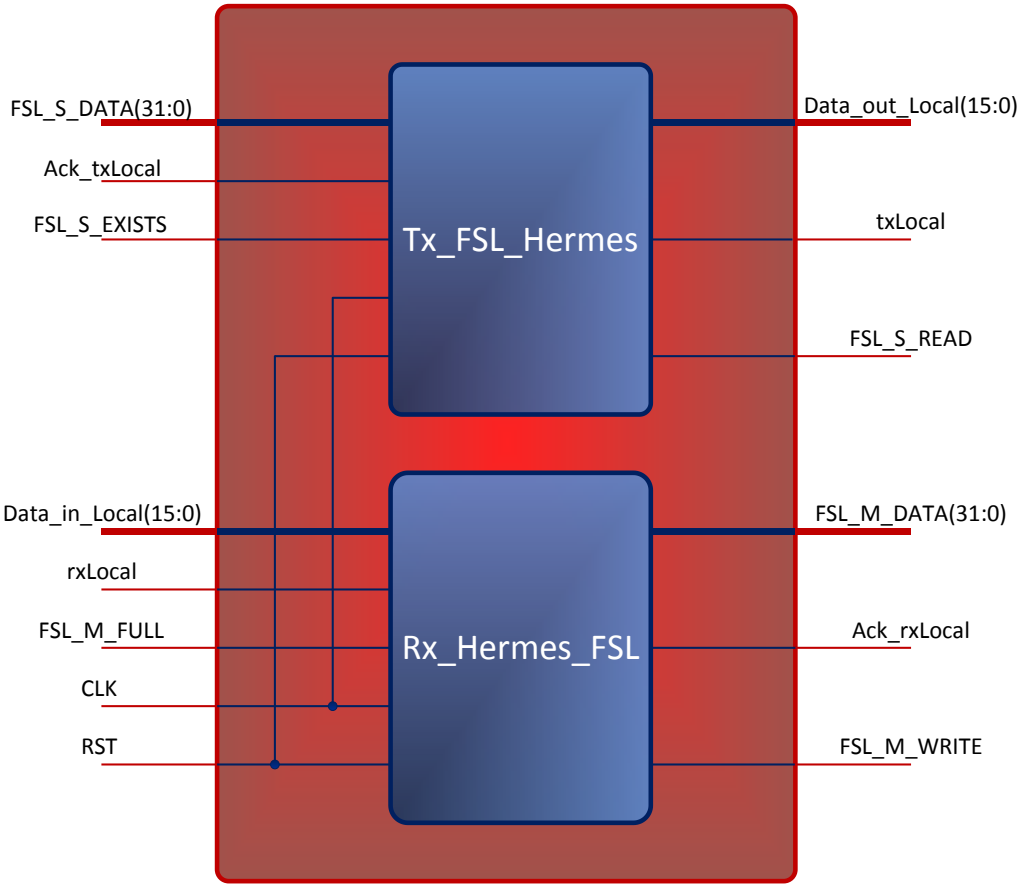


Figura. 6.10. IP-Core de la Interfaz de Red implementada

De la figura anterior se realiza una tabla en donde constan todas las entradas y salidas listas para ser conectadas al MicroBlaze (por medio del bus FSL) y a Hermes (Por medio de Puerto Local), completando así la arquitectura red.

Interfaz de Red (WiNi_Core)			
FSL	**I/O**	**Local Port**	**I/O**
FSL_S_DATA	In	Data_out_Local	Out
FSL_S_EXISTS	In	txLocal	Out
FSL_S_READ	Out	Ack_txLocal	In
FSL_M_DATA	Out	Data_in_Local	In
FSL_M_FULL	In	rxLocal	In
FSL_M_WRITE	Out	Ack_rxLocal	Out

Tabla. 6.4. Puertos de la Interfaz de red listos para conexión

El presente capítulo explicó el diseño propuesto de una interfaz de red (dispositivo que toma el nombre de WiNi_Core) que se realizó para interconectar a NoC Hermes con los MicroBlaze. En el siguiente capítulo se explicará la integración de la nueva interfaz dentro de la arquitectura de Hermes y los medios empleados para importar el nuevo IP-Core hacia la plataforma XPS, desde la plataforma ISE.

CAPÍTULO 7

IMPLEMENTACIÓN DE UN MPSOC A TRAVÉS DE NOC HERMES MEDIANTE INTERFAZ DE RED

En este capítulo se explica el proceso de interconexión de cuatro procesadores comunicados entre sí por medio de una NoC. La Figura. 7.1 detalla de forma esquemática la arquitectura de comunicación descrita anteriormente.

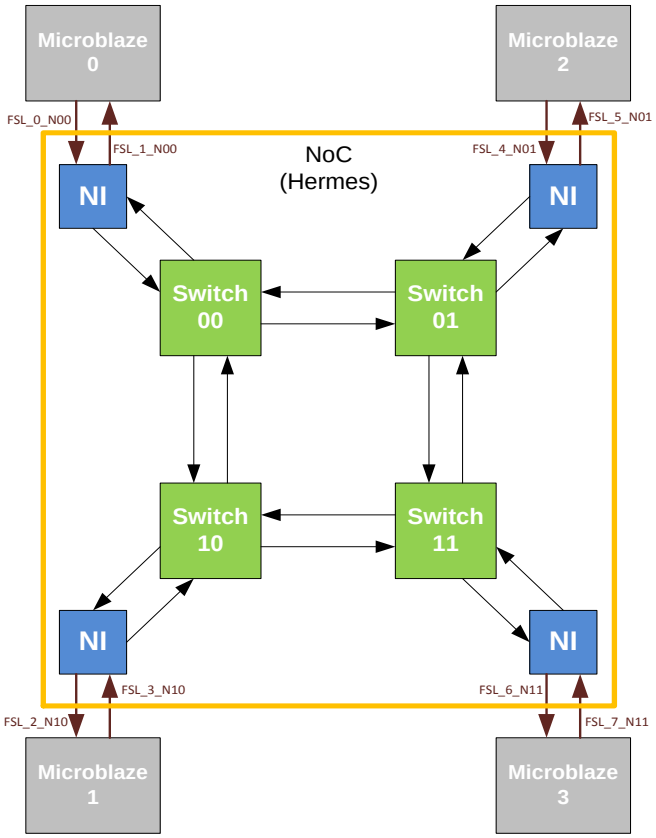


Figura. 7.1. Esquema de conexión MicroBlaze – NoC a través de la NI

En el presente capítulo se comienza por el diseño del MPSoC, luego se importa el módulo de comunicaciones conformado por NoC Hermes y el NI denominado WiNi_Core.

7.1. DISEÑO DE UNA INFRAESTRUCTURA MPSOC DE CUATRO PROCESADORES QUE UTILIZA UNA NOC PARA SU INTERCONEXIÓN

En la Figura. 7.1 se detalla la interconexión de cuatro procesadores MicroBlaze interconectados por NoC Hermes. Para el desarrollo de esta arquitectura se considera el diseño realizado en el **Capítulo 4**, el mismo que es modificado con la inclusión de NoC Hermes y el NI WiNi_Core. La NoC Hermes fue desarrollada por la Pontificia Universidad Católica de Rio Grande (Brasil) y adaptada a través de una NI desarrollada en la Escuela Politécnica del Ejército.

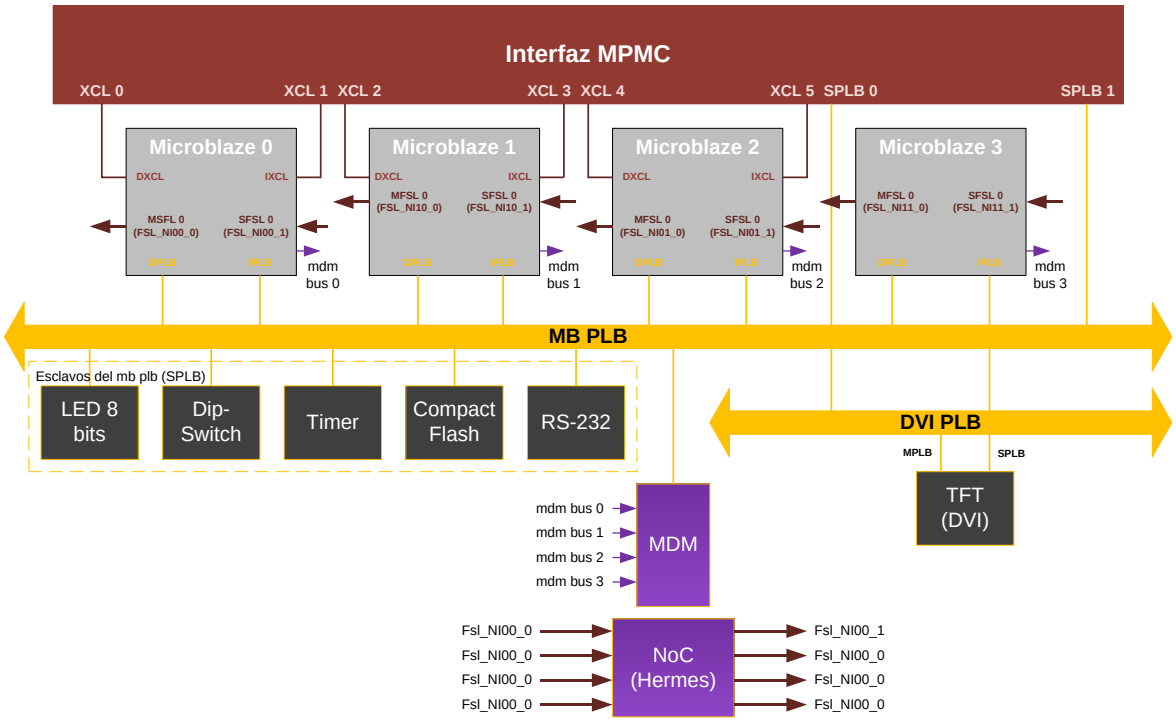


Figura. 7.2. Arquitectura de 4 de procesadores interconectados por una NoC

Para el desarrollo de la arquitectura de comunicación interconectada por NoC Hermes mediante NI se utiliza programación en VHDL y la herramienta *Create and Import Peripheral* (Ver Figura. 7.3). La herramienta que permite la importación de IP-Cores se describe en el Anexo 5.

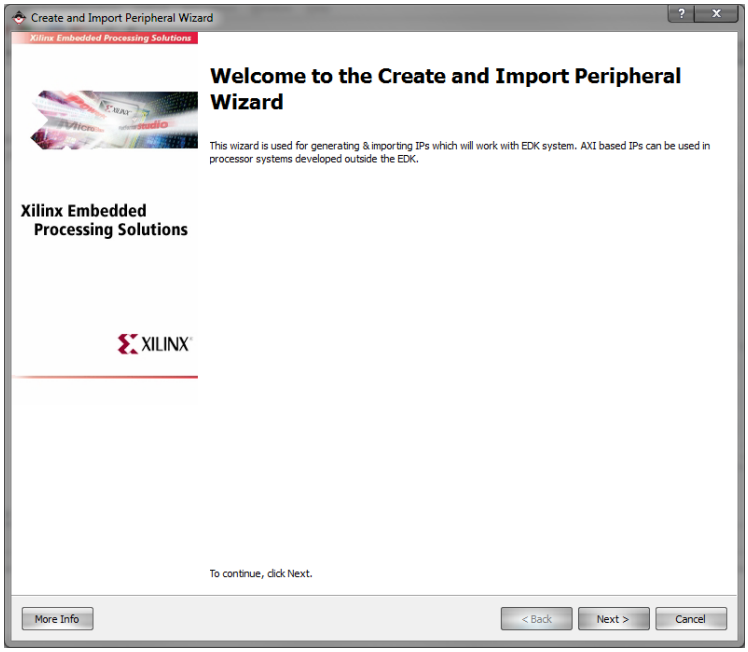


Figura. 7.3. Wizard para la importación de un IP-Core

7.1.1. Conexión Elemento de Red (NE) hacia Interfaz de Red (NI)

En la Figura. 7.4 se muestra al switch Hermes conectado a la interfaz de red. La conexión se da entre el puerto local del switch y el puerto local de la interfaz de red. El puerto local para la transmisión y recepción de datos se representa de forma esquemática por medio de flechas (ver Tabla. 6.4 del **Capítulo 6**)

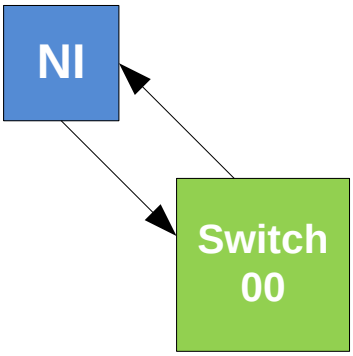


Figura. 7.4. Conexión NE (Switch) con NI (WiNi_Core)

WiNi_Core hace uso de la sentencia restringida *Entity* para el proceso de interconexión dentro de la declaración *Architecture*, pues la declaración tradicional *Component* es ineficiente al replicar cuatro veces el mismo dispositivo. El código VHDL de WiNi_Core se encuentra descrito en el Anexo 8.

Por lo tanto el esquema de la Figura. 7.4, es conectado físicamente por medio de las siguientes instrucciones creadas en VHDL:

```
NI0000: Entity work.WiNi_Core(Interfaz_Red)
   Port Map (  FSL_S_DATA        =>    NI0000_FSL_S_DATA,
               FSL_S_READ        =>    NI0000_FSL_S_READ,
               FSL_S_EXISTS      =>    NI0000_FSL_S_EXISTS,
               FSL_M_DATA        =>    NI0000_FSL_M_DATA,
               FSL_M_WRITE       =>    NI0000_FSL_M_WRITE,
               FSL_M_FULL        =>    NI0000_FSL_M_FULL,
               txLocal           =>    Senal_rxLocal(0),
               ack_txLocal       =>    Senal_ack_rxLocal(0),
               Data_out_Local    =>    Senal_data_inLocal(0),
               rxLocal           =>    Senal_txLocal(0),
               ack_rxLocal       =>    Senal_ack_txLocal(0),
               Data_in_Local     =>    Senal_data_outLocal(0),
               Clk => CLK,
               Rst => RST);
```

Las conexiones para comunicarse hacia el bus FSL se identifican como NIXXXX_FSL_X_X, y las conexiones para interactuar con el Swicht 00 se las realiza a través de señales internas. Para relacionar las señales internas del Switch 00 se emplean arreglos que se encuentran dentro del package *Hermes_Package* (librería incluida con los módulos de Hermes y descrita en el **Capítulo 5**). El arreglo *regNrot*, tiene una longitud de 4 bits para el tipo de flujo de control Handshake y el arreglo *arrayNrot_regflit*, tiene una longitud de 4 vectores de 16 bits cada uno para Datos. Tanto el *regNrot* como el *arrayNrot_regflit* son importantes para crear el arreglo de señales denominadas como *Senal_X(n)*.

```
Architecture Hermes of Top_NoC is
   Signal Senal_rxLocal,
          Senal_ack_rxLocal,
          Senal_txLocal,
          Senal_ack_txLocal : regNrot := (others => '0');
   Signal Senal_data_inLocal,
          Senal_data_outLocal : arrayNrot_regflit;
```

7.1.2. Conexión MicroBlaze – NoC Hermes

El IP-Core que se utiliza para la comunicación entre MicroBlaze y NoC Hermes contiene los elementos de red con sus respectivas interfaces de red. Este IP-Core descrito en VHDL se lo denomina Top_NoC. Top_NoC se importa desde el ISE hacia el XPS en donde toma el nombre de NoC_Hermes mediante la herramienta *Create and Import Peripheral*. La Figura. 7.1 representa el esquema de conexión entre un MicroBlaze y un switch de NoC Hermes mediante WiNI_Core. El Código VHDL de NoC_Hermes se lo describe en el Anexo 9.

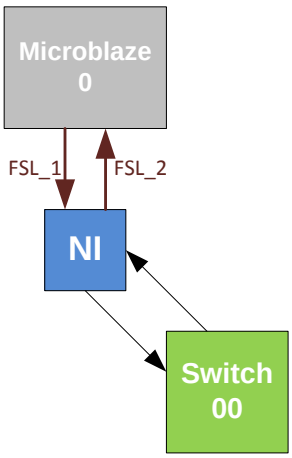


Figura. 7.5. Conexión final, NoC Hermes con PE (MicroBlaze)

Luego de importar el IP-Core NoC_Hermes al XPS se lo puede encontrar en la pestaña IP Catalog en la sección de Local PCores creados por el diseñador como se muestra en la Figura. 7.6.

Figura. 7.6. Pestaña IP Catalog con NoC Hermes

Al seleccionar IP-Core NoC_Hermes del IP Catalog se realiza las siguientes configuraciones dentro del archivo MPD:

```
## Bus Interfaces
BUS_INTERFACE BUS = SFSL0, BUS_STD = FSL, BUS_TYPE = SLAVE
BUS_INTERFACE BUS = MFSL0, BUS_STD = FSL, BUS_TYPE = MASTER

## Generics for VHDL or Parameters for Verilog
PARAMETER   C_FSL_DATA_SIZE = 32, DT = integer, RANGE = (32)
## Ports
PORT  NI0000_FSL_S_DATA = FSL_S_DATA, DIR = I, VEC = [C_FSL_DATA_SIZE-
1:0], BUS = SFSL0
PORT  NI0000_FSL_S_READ = FSL_S_READ, DIR = O, BUS = SFSL0
PORT  NI0000_FSL_S_EXISTS = FSL_S_EXISTS, DIR = I, BUS = SFSL0
```

```
PORT  NI0000_FSL_M_DATA = FSL_M_DATA, DIR = O, VEC = [C_FSL_DATA_SIZE-
1:0], BUS = MFSL0
PORT  NI0000_FSL_M_WRITE = FSL_M_WRITE, DIR = O, BUS = MFSL0
PORT  NI0000_FSL_M_FULL = FSL_M_FULL, DIR = I, BUS = MFSL0
```

En la sección *Bus interface*, se asigna la identificación del bus maestro o esclavo, definidos en Bus_Type. Mientras que en *Generics for VHDL or Parameters for Verilog*, se asigna el tamaño del bus para datos del FSL que tiene un valor máximo de 32 bits. En la sección *Ports*, se establecen las correspondencias de las conexiones provenientes del software ISE para ser conectados con sus equivalentes en los terminales del bus FSL.

Utilizando la configuración del archivo MPD de NoC_Heremes se pueden interconectar los cuatro MicroBlaze mediante buses FSL hacia el NI y mediante lenguaje VHDL se interconectan los cuatro NI con los cuatro Switches de la topología 2D-Mesh de Hermes a través de sus respectivos Puertos Locales.

En la Tabla. 7.1 se muestran los recursos utilizados en el FPGA Virtex-6.

Recursos utilizados en FPGA ML605 Virtex 6			
Lógica Utilizada	**Utilizado**	**Capacidad Total**	**Porcentaje**
Número de LUT's	15381	150720	10%
Número de Registros de Silicio	15462	301440	5%

Tabla. 7.1. Consumo FPGA ML605 Virtex-6

Finalmente se han desarrollado dos arquitecturas de multiprocesamiento, una interconectada por buses FSL especificada en el **Capítulo 4** y la otra interconectada por NoC Hermes con la ayuda de una Network Interface especificada en el presente capítulo. Dichas arquitecturas fueron diseñadas con el

fin de evaluar su desempeño sobre una aplicación que demande mínima latencia y mínima perdida de paquetes, motivo por el cual en el siguiente capítulo se desarrolla una aplicación que cumpla con estos parámetros.

CAPÍTULO 8

APLICACIÓN DE ESTEGANOGRAFÍA

8.1. INTRODUCCIÓN

Esteganografía proviene del vocabulario griego *estagano*, que significa encubierto y *graphos* que significa escritura, por tanto el término Esteganografía significa: el arte de escribir de forma oculta. (Death).

La Esteganografía trabaja con meta información, es decir información dentro de otra información y ha sido empleada desde tiempos remotos dentro de la vida del ser humano. Sin embargo, hoy en día la esteganografía ha tomado mayor importancia, debido a que existen mayores recursos para su aplicación dentro del campo de la informática y las comunicaciones. (Death)

Al referirse al campo de la informática, se quiere decir que el ocultamiento de mensajes secretos bien puede aplicarse en una gran variedad de ficheros (documentos, imágenes, audio, video, programas ejecutables etc.). Estos ficheros pueden pasar sin ser detectados pese al esfuerzo de filtros informáticos o de personal especializado que se encarga de su detección. (Observatorio de la Seguridad de la Información)

En el campo de las comunicaciones la esteganografía se aplica en el protocolo TCP/IP, convirtiéndolo en un canal de comunicación ideal para dicha práctica. Para este caso se usan las cabeceras del protocolo, en donde se oculta la información siempre y cuando la conexión no se vea interrumpida o se afecte la integridad de los paquetes intercambiados. (Observatorio de la Seguridad de la Información)

8.1.1. Bases para la Esteganografía

El portador para el caso del presente proyecto de tesis es una imagen sobre la cual se aplicará la técnica de esteganografía por sustitución del bit menos significativo. Dicha técnica es comúnmente empleada, aunque pueden existir otras técnicas más complejas que emplean algoritmos para el proceso de ocultamiento. (Death) (Observatorio de la Seguridad de la Información)

Como ya se mencionó, el método de esteganografía que se aplicará es por sustitución con la cual se ocultarán datos binarios camuflados dentro de un grupo de bits que componen una imagen o portador, procurando que la imagen resultante aparente ser la original. Dicho proceso se logrará mediante la ejecución de un método que permita ocultar caracteres ASCII convertidos en tramas de bits.

8.1.2. Método para la Esteganografía por Sustitución del Bit Menos Significativo

El método depende del tipo de imagen con el que se esté tratando, que bien pueden ser imagines en formato bmp, jpg, pnm, etc. con una gran variedad de colores. Las imágenes monocromáticas, en escala de grises o con baja diversidad

de color presentan un cierto grado de notoriedad cuando se las usan como portadoras. (Death) (Observatorio de la Seguridad de la Información)

Una vez claro el tipo de fichero con el que se trabajará, entonces se deben tener en cuenta las siguientes consideraciones:

1. Toda información, que se quiera ocultar debe ser convertida a lenguaje binario.
2. Las cabeceras de los ficheros nunca deben ser modificadas.
3. Se tiene que mantener claves muy bien definidas para la obtención segura de la información esteganografiada.

8.2. DESARROLLO DE UNA APLICACIÓN ESTEGANOGRAFÍCA APLICADA SOBRE MPSOC

Para la aplicación se ha seleccionado el formato PPM (Portable PixMap) por ser ideal para trabajos relacionados con procesamiento de imágenes. Su utilización es sencilla así como la programación realizada para adquirir su información. Además mantiene bajo el consumo de memoria lo que lo hace apto para ser empleado en una tarjeta FPGA como la Virtex 6.

8.2.1. Estructura del Formato PNM (Portable Anymap)

El formato PNM contiene a los formatos de imagen PPM, PGM (Portable GrayMap) y PBM (Portable BitMap). De los tres, el que es de interés es el de extensión PPM, por almacenar imágenes en color, las otras extensiones son para imágenes en escala de grises y monocromáticas respectivamente. (Vazquez)

Formato PPM

Existe una versión binaria y otra ASCII, en el primer caso, la imagen se codifica como una secuencia de bits, y en el segundo caso, se codifica en un archivo de texto plano, donde cada pixel viene representado por tres números del código RGB, en donde el máximo valor es 255 y el mínimo 0. (Procesamiento de Imáganes, 2008)

La estructura de una imagen en formato PPM, viene dada por una cabecera que contiene la información sobre el archivo (tipo de imagen, dimensión y máximo valor de color) y la sección de datos que almacena la imagen, todo en ASCII o binario. (Procesamiento de Imágenes Digitales)

Cabecera	
3	Formato de la imagen; P3 para ASCII o P6 para binario.
3	Dimensión de la imagen dadas por el ancho y el alto respectivamente.
255	Valor máximo que puede tomar una componente de color.

Tabla. 8.1. Cabecera de la Imagen en Formato PPM

Cuerpo de la imagen								
R	**G**	**B**	**R**	**G**	**B**	**R**	**G**	**B**
0	0	0	0	0	0	255	0	0
0	0	0	0	255	0	255	255	255
0	0	255	255	255	255	255	255	255

Tabla. 8.2. Sección de datos Imágenes en Formato PPM

Tanto la Tabla. 8.1 como la Tabla. 8.2, ejemplifican la estructura interna del archivo en formato PPM (abierto por medio de un editor de texto), sin embargo la

siguiente figura muestra lo que realmente se aprecia cuando se abre una de estas imágenes.

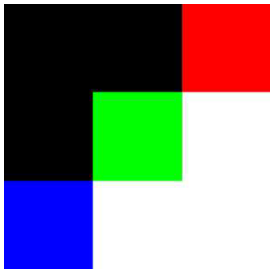


Figura. 8.1. Ejemplo Imagen Formato PPM

En la Figura. 8.1 se puede apreciar claramente la correspondencia de los pixeles conformados por las componentes RGB mostradas en la Tabla. 8.2. Incluso se puede apreciar la dimensión de la imagen que es de 3x3 (Tabla. 8.1).

8.2.2. Codificación de los Caracteres Dentro de la Imagen ppm

Como ya se mencionó, el formato ppm está compuesto por tres componentes RGB correspondientes a un pixel. Por lo tanto para introducir un caracter dentro de varios pixeles se hace uso de la primera consideración, mencionada anteriormente, para poder hacer pequeñas variaciones que sean imperceptibles para el ojo humano. Para cumplir con esto es necesario modificar el bit menos significativo del pixel y sustituirlo por el binario del byte equivalente del caracter desplazándolo desde el menos significativo hacia el más significativo, tal y como se muestra en la siguiente tabla (Juan).

Pixel	Color	Bits Pixel	Bits Carácter (A)	Bits Pixel Final	Bits Carácter	
1	R	11111111	01000001	11111111	1	A
	G	11111111	01000001	11111110	0	
	B	11111111	01000001	11111110	0	
2	R	11111111	01000001	11111110	0	
	G	11111111	01000001	11111110	0	
	B	11111111	01000001	11111110	0	
3	R	11111111	01000001	11111111	1	
	G	11111111	01000001	11111110	0	
	B	11111111	xxxxxxxx	1111111x	x	x

Tabla. 8.3. Ejemplo de introducción de un carácter en un grupo de tres pixeles.

En la Tabla. 8.3 se puede apreciar que es necesario tener tres pixeles para camuflar un byte, correspondiente a la letra A. Nótese que cada pixel se forma por una componente de rojo (R), una de verde (G) y otra de azul (B), este último no se utiliza en el tercer pixel pero termina siendo parte del grupo de pixeles que esteganografían ha dicha letra en código binario. Además es preferible mantenerlo sin alteración por facilidad de programación.

De esta manera se procede al desarrollo de un software que permita cumplir con lo expuesto en la Tabla. 8.3, para ello se toma en consideración los siguientes diagramas de flujo.

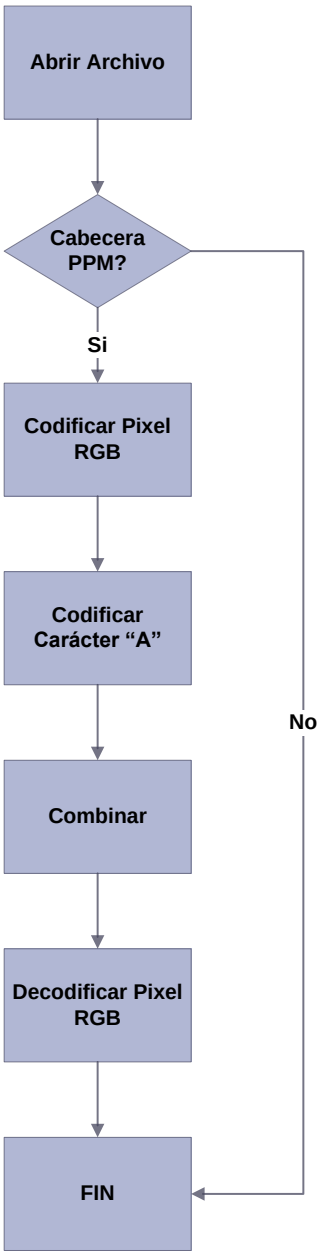


Figura. 8.2. Diagrama de Flujo para la Esteganografía

Como se pude apreciar en el diagrama de flujo de la Figura. 8.2 el software abre el fichero, verifica que exista un archivo de extensión ppm, extrae la información de la imagen para luego seguir con la codificación de la misma a binario y luego hacer lo mismo con los caracteres. Posteriormente se combina los bits del caracter con los bits de los pixeles para finalmente decodificarlos y guardarlos. Caso contrario, si no existe imagen con formato ppm el programa finaliza.

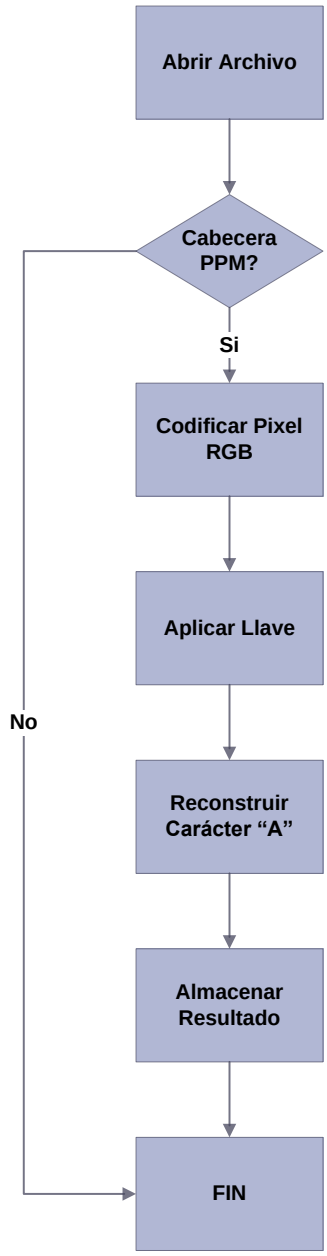


Figura. 8.3. Diagrama de Flujo para Realizar la Decodificación de la Imagen Esteganografiada

Por otro lado la Figura. 8.3, muestra el proceso inverso para decodificar la información contenida en la imagen que ha sido esteganografiada. En este proceso se debe abrir el fichero, comprobar si es de una imagen en formato ppm para luego decodificar nuevamente los pixeles, después aplicar la llave que permite descifrar el mensaje. Se reconstruye el caracter y posteriormente se lo almacena en un archivo de texto simple. Caso contrario si no hay archivo ppm se termina el programa.

Los diagramas de flujo mostrados en la Figura. 8.2 y Figura. 8.3 son una abstracción de todo el programa desarrollado para el presente proyecto. Los procesos de esteganografía y de decodificación dependen de la arquitectura de comunicación entre los procesadores sobre los cuales se ejecuta la aplicación.

Los procesos de esteganografía y de decodificación se ejecutan en procesadores distintos y los resultados son expuestos por medio de un monitor que se conecta al FPGA mediante el puerto DVI.

Las dimensiones de las imágenes empleadas en esta aplicación son de 240x320 pixeles, con capacidad para ocultar 25600 caracteres. Por los recursos de memoria del FPGA únicamente se pudo ocultar un 2% de los 25600 caracteres que se pueden contener dentro de la imagen. Esta limitante únicamente permite utilizar imágenes de un tamaño menor o igual a 240x320 pixeles.

8.2.3. Aplicación Sobre Procesadores MicroBlaze Comunicados por Buses FSL

En la Figura. 8.4 se puede apreciar una arquitectura de cuatro procesadores MicroBlaze comunicados por buses FSL. El MicroBlaze 0 y el MicroBlaze 2 realizan el proceso de esteganografía de forma paralela sobre dos imágenes distintas. Mientras que el MicroBlaze 1 y el MicroBlaze 3 realizan el proceso de decodificación de forma paralela para extraer la información contenida en las imágenes.

El paralelismo se interrumpe cuando los MicroBlaze se comunican entre sí para acceder a recursos compartidos del FPGA. Los procesadores comparten el

acceso a la memoria Compact Flash (CF) para la adquisición de imágenes y mensajes, y la proyección de resultados en un monitor a través del puerto DVI.

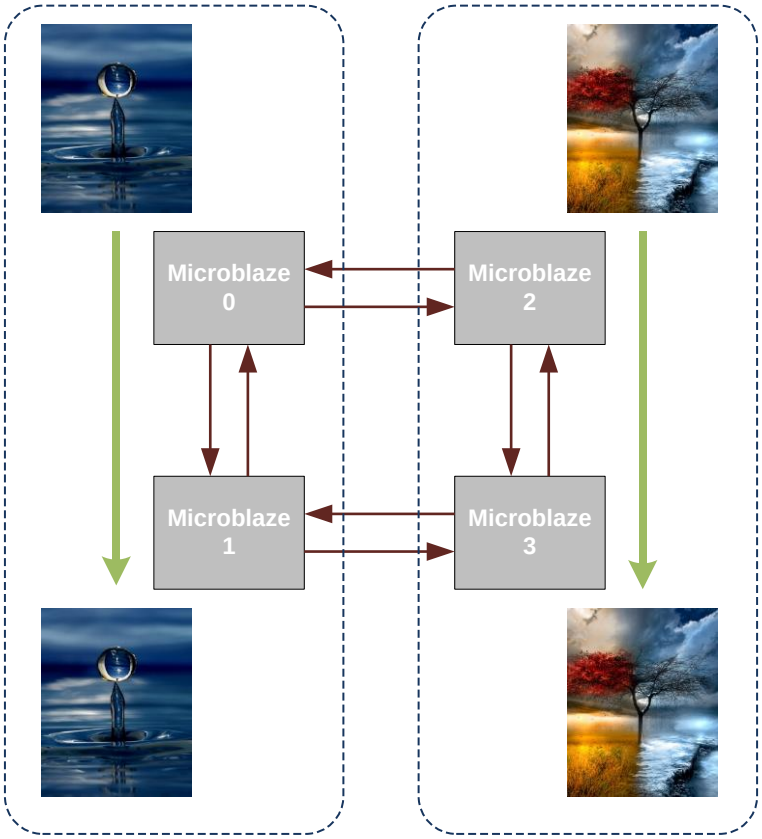


Figura. 8.4. Proceso de Esteganografía y de Decodificación de mensaje sobre primera arquitectura

En la arquitectura de multiprocesamiento interconectado por buses FSL los procesadores intercambian un dato por instrucción de arbitraje. Cada instrucción de arbitraje permite acceder a los recursos compartidos que tienen disponibles. Mientras que para la transmisión de información de pixeles cada MicroBlaze procesa un arreglo de tres datos. El arreglo corresponde a cada una de las componentes del modelo RGB. El código de esta aplicación se lo puede revisar en el Anexo 10.

8.2.4. Aplicación Sobre Procesadores MicroBlaze Comunicados por NoC Hermes

En la Figura. 8.5 se puede apreciar una arquitectura de cuatro procesadores MicroBlaze comunicados por NoC Hermes. El MicroBlaze 0 y el MicroBlaze 2 realizan el proceso de esteganografía de forma paralela sobre dos imágenes distintas. Mientras que el MicroBlaze 1 y el MicroBlaze 3 realizan el proceso de decodificación de forma paralela para extraer la información contenida en las imágenes.

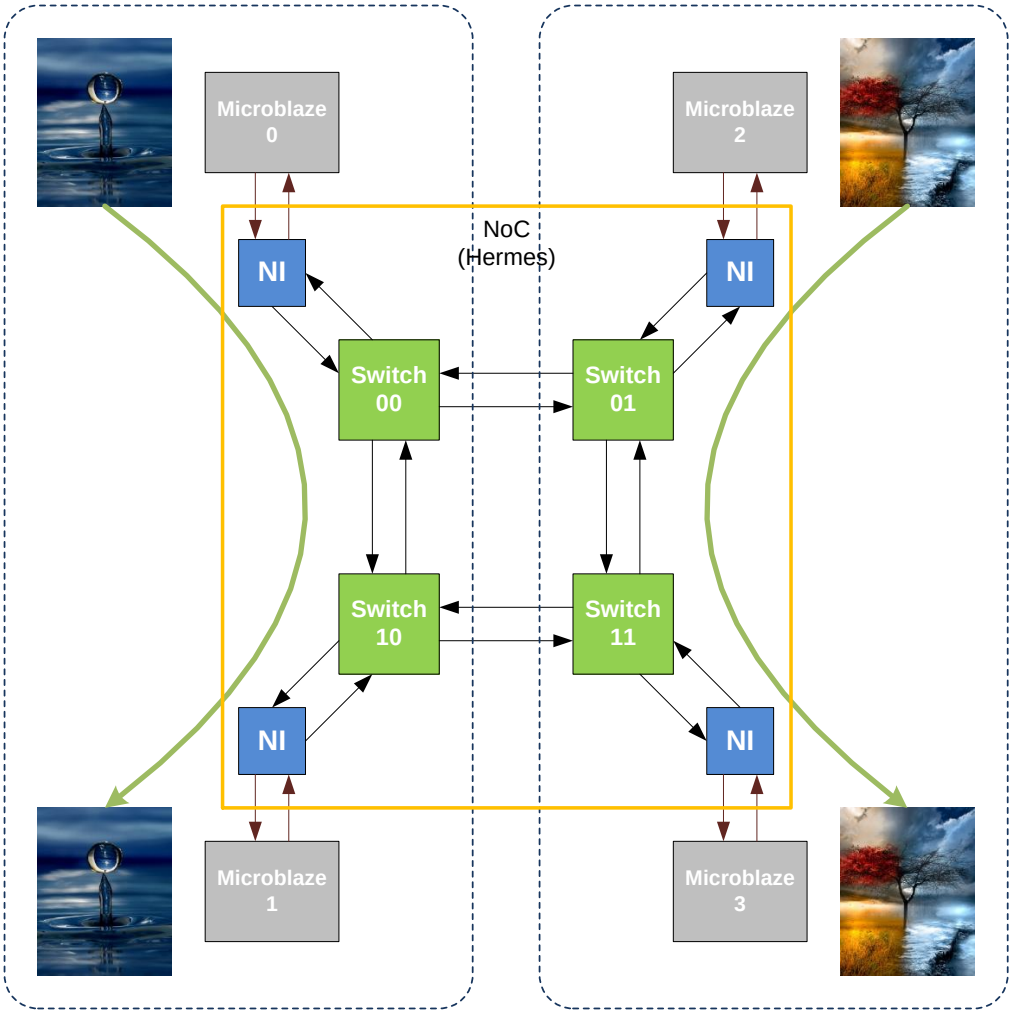


Figura. 8.5. Proceso de Esteganografía y de Decodificación de mensaje sobre segunda arquitectura

El paralelismo se interrumpe cuando los MicroBlaze se comunican entre sí para acceder a recursos compartidos del FPGA. Los procesadores comparten el acceso a la memoria Compact Flash (CF) para la adquisición de imágenes y mensajes, y la proyección de resultados en un monitor a través del puerto DVI.

En la arquitectura de multiprocesamiento interconectado por NoC los procesadores intercambian un paquete de arbitraje. Cada paquete de arbitraje contiene información de direccionamiento XY, información de tamaño y el dato de instrucción de acceso a los recursos compartidos que tienen disponibles. Mientras que para la transmisión de información de pixeles cada MicroBlaze procesa un paquete que contiene un arreglo de tres datos. El paquete contiene direccionamiento XY, tamaño del paquete y el arreglo que corresponde a cada una de las componentes del modelo RGB. El código de esta aplicación se lo puede revisar en el Anexo 10.

El procesamiento de una imagen, desde un MicroBlaze a otro para la arquitectura interconectada por buses FSL y para la arquitectura interconectada por NoC Hermes, ocurre de forma paralela haciendo posible el procesamiento múltiple. De esta forma obtienen dos imágenes esteganografiadas con diferentes mensajes junto con los mensajes decodificados y guardados en una archivo de texto simple dentro de la CF. Todos los archivos, tanto las imágenes originales y esteganografiadas, los archivos fuente y decodificados en texto simple son desplegados en la pantalla de un monitor.

En el **Capítulo 9** se evalúa el desempeño y se analizan los resultados obtenidos para las arquitecturas de multiprocesamiento interconectadas por buses FSL y por una NoC Hermes de 2x2 en topología 2D-Mesh. Las arquitecturas de multiprocesamiento ejecutan una aplicación de esteganografía para dos imágenes distintas sobre el FPGA ML605 Virtex-6.

CAPÍTULO 9

PRUEBAS Y ANÁLISIS DE RESULTADOS

Para la realización de pruebas y la obtención de los resultados que están expuestos en este capítulo se usan las herramientas ISim (Simulador del ISE) y XPower Analizer, analizadas en el **Capítulo 3** que están incluidos con el software ISE 13.2 de Xilinx.

En el presente capítulo se exponen escenarios de pruebas realizados sobre una arquitectura interconectada por NoC Hermes y otra interconectada por Buses FSL. Finalmente en la parte de análisis de resultados se comparan los datos calculados, simulados y estimados para los distintos escenarios de prueba. Los análisis y resultados de estos escenarios de prueba permiten evaluar latencia y consumo de potencia para cada una de las arquitecturas.

9.1. PRUEBAS DE LATENCIA PARA ARQUITECTURAS INTERCONECTADAS POR NOC HERMES Y POR BUSES FSL

9.1.1. Pruebas de Transmisión de Datos en NoC Hermes sin Interfaz de Red

Este apartado trata sobre la simulación realizada en NoC Hermes el cual ya se caracterizó en el **Capítulo 5**. Por lo tanto a continuación se mostrará las simulaciones realizadas para comprobar la funcionalidad de Hermes a través de generadores de tráfico.

9.1.1.1. Modelo de Pruebas

La Figura. 9.1, muestra el modelo de pruebas a realizarse que corresponde a una configuración 2D-Mesh de Switches Hermes para cuatro procesadores MicroBlaze, sobre la cual se harán las pruebas que determinan la viabilidad de esta NoC como medio de comunicación.

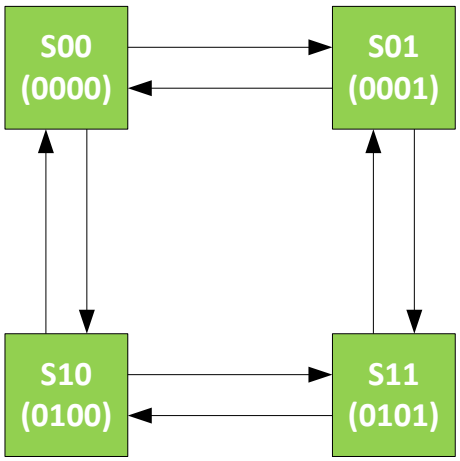


Figura. 9.1. Hermes en configuración de 2D-Mesh

Para plantear el modelo de pruebas se utilizó generadores de tráfico NoCgen. Los mismos que fueron descargados de la página web del grupo de investigación System Level Synthesis correspondiente al laboratorio TIMA (Técnicas de Información y Microelectrónica para la Arquitectura de Sistemas Integrados) bajo la supervisión del Centro Nacional de Investigación Científica (CNRS), el Instituto

Politécnico de Grenoble (Grenoble IPN) y la Universidad Joseph Fourier (UJF). (TIMA)

- **Generadores de Tráfico NoCgen**

Los generadores de tráfico NoCgen están programados en código abierto VHDL. A fin de utilizar estos generadores de tráfico fue necesario modificar su programación para que se ajuste a la arquitectura planteada. Al utilizar los generadores de tráfico NoCgen se demuestra que Hermes es un medio idóneo para la transferencia de paquetes de diferente tamaño. Estos generadores se disponen dentro de la topología de Hermes según la Figura. 9.2.

Los generadores de tráfico se componen de los siguientes archivos incluidos en la descarga de la página mencionada:

Archivo	Descripción
ParameterPackage.vhd	Librería para la configuración de parámetros necesarios para evaluar la NoC.
Top_noc.vhd	La NoC propiamente dicha interconectada con los generadores de tráfico NoCgen
Traffic_generator.vhd	Generador de Tráfico.
Traffic_receptor.vhd	Receptor de Tráfico.

Tabla. 9.1. Archivos generadores tráfico

Los archivos *Traffic_generator.vhd* y *Traffic_receptor.vhd* se conectan a cada uno de los Switches de Hermes representados por GTYY y RxYY respectivamente (ver Figura. 9.2) donde YY es el número de elementos de red.

Las conexiones de los GTYY y RxYY se las realiza a través del archivo *Top_NoC.vhd* en donde se entrelazan las conexiones de los Switches de Hermes con los generadores (GTYY) y receptores (RxYY) de tráfico. Finalmente el archivo *ParameterPackage.vhd* es una librería en donde se configuran las condiciones bajo las que se realizan las pruebas necesarias.

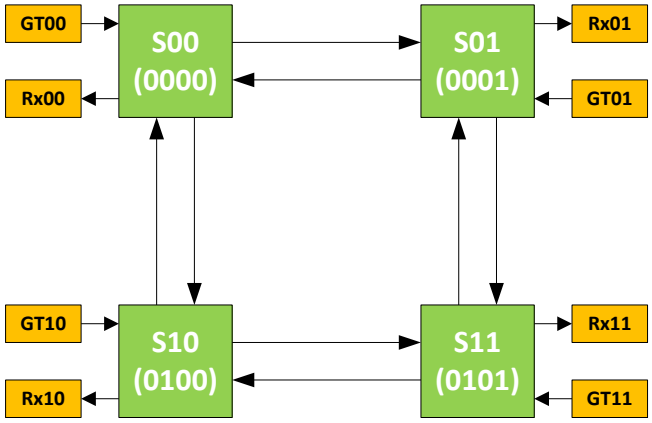


Figura. 9.2. Generadores de Tráfico y Receptores de Tráfico NoCgen conectados a Switches Hermes

- **Transmisión de Paquetes**

Los generadores de tráfico transmiten paquetes de tamaño variable de entre 3 a 16 flits, con un tamaño de flit de hasta 16 bits.

Datos de Transmisión en los Generadores de Tráfico			
N° Flit	**Estructura del Paquete**	**Flit en Binario**	**Flit en Hexadecimal**
1	Dirección IP	---	---
2	Tamaño del Paquete	---	---
3	Dato 1	0000000000000000	0x0000
4	Dato 2	0000000000000000	0x0000
5	Dato 3	1111111111111110	0xfffe

Datos de Transmisión en los Generadores de Tráfico			
N° Flit	**Estructura del Paquete**	**Flit en Binario**	**Flit en Hexadecimal**
6	Dato 4	1111111111111100	0xfffc
7	Dato 5	1111111111111000	0xfff8
8	Dato 6	1111111111110000	0xfff0
9	Dato 7	1111111111100000	0xffe0

Tabla. 9.2. Paquete de transmisión programado en los generadores de tráfico

La Tabla. 9.2, muestra la configuración por defecto de un generador de tráfico para 9 flits utilizados en una simulación de prueba con los generadores de tráfico. Obsérvese además que en la Tabla. 9.2 algunos casilleros están vacíos. Esto se debe a que son parámetros que se pueden variar por medio de la librería *ParameterPackage.vhd* que viene incluida con la descarga del NoCgen [95].

El generador de tráfico ha sido programado para transmitir un máximo de nueve flits (7 Datos) aun cuando aparentemente se pueden hacer pruebas con mayor cantidad de datos. Al realizar una simulación con mayor cantidad de datos se puede verificar que a partir del décimo flit (Dato 8) el valor del Dato 7 se repite. Por esta razón y dado que sólo se trata de una prueba, no se ha considerado esta inconsistencia de programación y se realizaron pruebas donde el tamaño de paquetes varían entre 3 a 16 flits.

- **Recepción de Paquetes**

Cada uno de los módulos Rx que se observan en la Figura. 9.3, es un IP-Core que recepta la información transmitida por el generador de tráfico y es el encargado de evaluar el tiempo que tarda Hermes en transmitir y recibir la

información entre cualquier nodo que se configure según la librería *ParameterPackage.vhd* mencionada anteriormente. En otras palabras, calcula la latencia entre los Switches de Hermes que sirve como referencia en el posterior desarrollo de análisis de resultados.

- **Latencia en Hermes**

La ecuación para latencia en NoC Hermes es la siguiente (Moraes, Vilar, Viera, Möller, & Copello, 2003):

$$Latencia = \left(\sum_{i=1}^{n} R_i\right) + P \times 2$$

Y corresponde al tiempo que tarda para transmitir información desde el origen al destino, en donde:

n: Número de elementos de red.
R_i: Tiempo de dispersión.
P: Tamaño del paquete.

9.1.1.2. Escenario de Pruebas sobre NoC Hermes utilizando Generadores de Tráfico para dos Nodos

En la Figura. 9.3, se observa con línea roja los módulos que se activarán en la simulación, es decir el S00 y el S10. El Generador (GT00) inyectará un paquete de datos al Switch 00 (S00) con destino al Switch 10 (S10), que a su vez envía la información al Receptor (Rx00) el cual evalúa el tiempo total de la transmisión realizada.

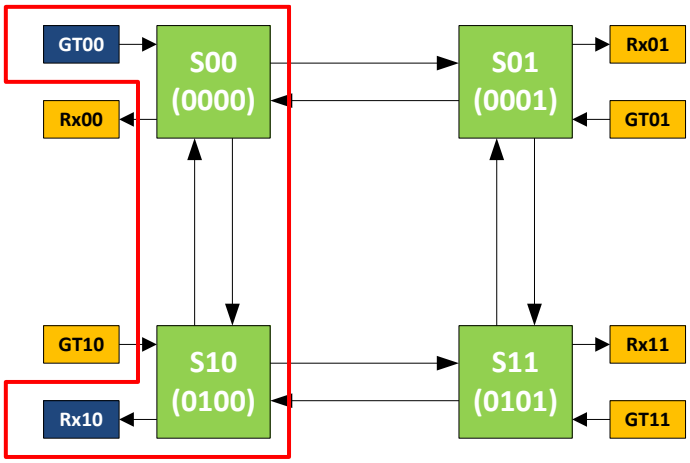


Figura. 9.3. Generadores y Receptores de Tráfico NoCgen conectados a Hermes para dos nodos

Antes de simular, se realizan cálculos teóricos que posteriormente serán comparados con los resultados del ISim (ver Tabla. 9.3). Estos cálculos se hacen con un tiempo de dispersión de 6.5µs, para dos nodos y con un tamaño variable de paquetes de 48 a 256 bits.

$$Latencia = (6.5 + 6.5) + (3 \times 2) = 19\mu s$$

No de Bits	Latencia un Nodo (µs)	Latencia dos Nodos	Tamaño Paquete	Tamaño por 2	Latencia en NoC (µs)
48	6,5	13	3	6	19
64	6,5	13	4	8	21
80	6,5	13	5	10	23
96	6,5	13	6	12	25
112	6,5	13	7	14	27
128	6,5	13	8	16	29
144	6,5	13	9	18	31
160	6,5	13	10	20	33
176	6,5	13	11	22	35

No de Bits	Latencia un Nodo (µs)	Latencia dos Nodos	Tamaño Paquete	Tamaño por 2	Latencia en NoC (µs)
192	6,5	13	12	24	37
208	6,5	13	13	26	39
224	6,5	13	14	28	41
240	6,5	13	15	30	43
256	6,5	13	16	32	45

Tabla. 9.3. Cálculos de Latencia en Hermes para dos nodos

Las simulaciones que a continuación se muestran prueban a NoC Hermes en la transmisión de paquetes con los siguientes tamaños:

- **Transmisión de 1 paquete de 5 flits desde S00 hacia S10**

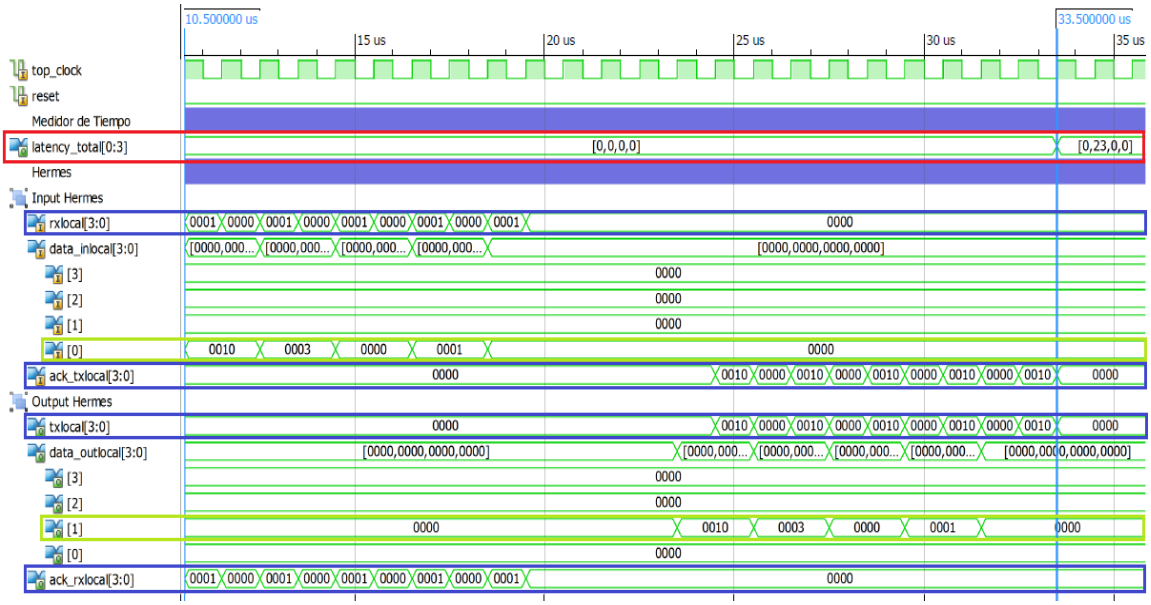


Figura. 9.4. Simulación en ISim del modelo de pruebas para una transmisión de 5 flits

La Figura. 9.4, muestra un recuadro rojo que engloba la latencia total medida con cuatro valores [V1, V2, V3, V4] donde cada valor corresponde a cada receptor

de tráfico, cuyo valor puede ser comparado con la Tabla. 9.3, que contiene los valores calculados. En el mismo gráfico se muestra en verde los datos transmitidos por S00 (data_inlocal[0]) y los datos receptados por S10 (data_outlocal[1]), el resto de señales que están en azul corresponden al control de flujo por Handshake que se realiza para la confirmación en la transmisión y recepción de los datos.

La latencia medida en la Figura. 9.4 equivale a la diferencia entre: X1: 33.5µs – X2: 10.5µs = ΔX: 23µs.

- **Transmisión de 1 paquete de 7 flits desde S00 hacia S10**

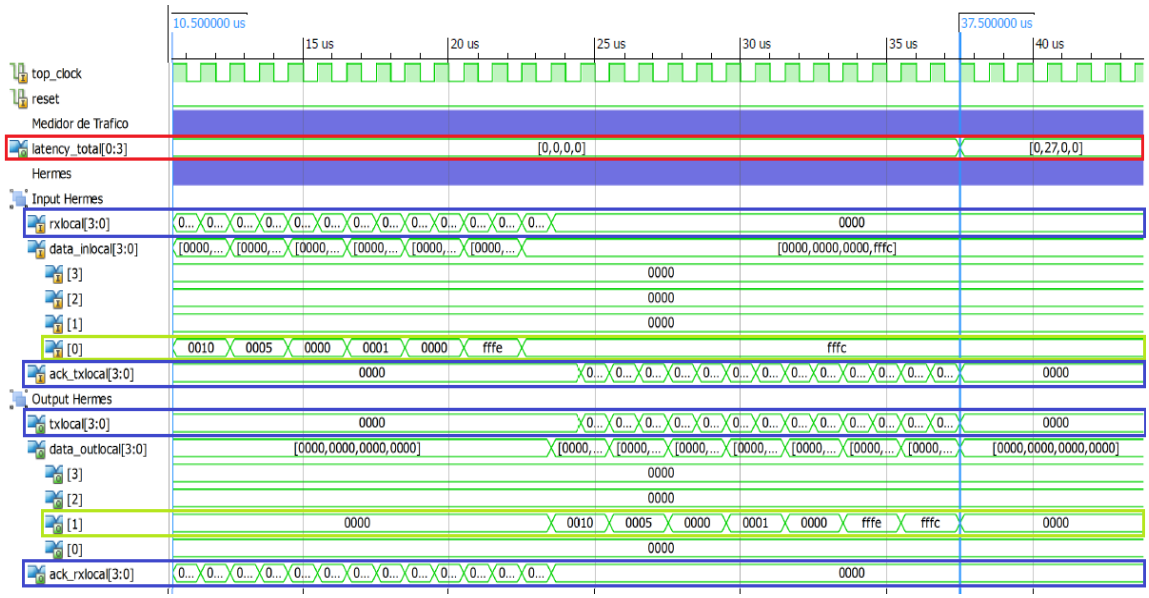


Figura. 9.5. Simulación en ISim del modelo de pruebas para una transmisión de 7 flits

La latencia medida es la Figura. 9.5 equivale a la diferencia entre: X1: 37.5µs – X2: 10.5µs = ΔX: 27µs.

- **Transmisión de 1 paquete de 9 flits desde S00 hacia S10**

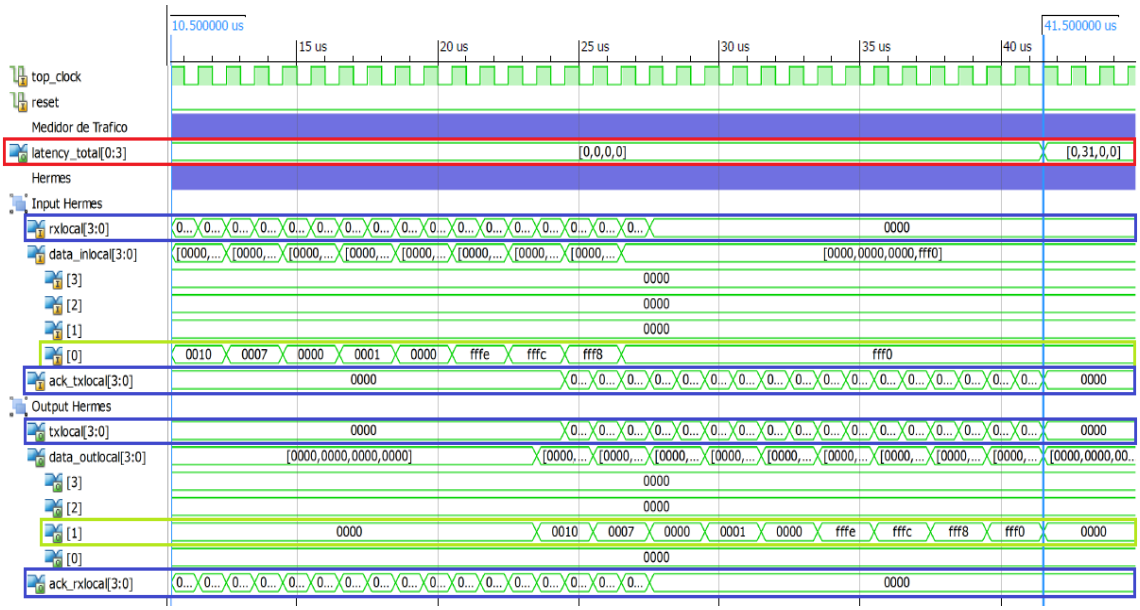


Figura. 9.6. Simulación en ISim del modelo de pruebas para una transmisión de 9 flits

La latencia medida en la Figura. 9.6 equivale a la diferencia entre: X1: 41.5µs – X2: 10.5µs = ΔX: 31µs.

La Figura. 9.5 y Figura. 9.6, son muy similares a la Figura. 9.4 la principal diferencia radica en el valor de latencia debido al incremento de la longitud del paquete.

9.1.1.3. Escenario de Pruebas sobre NoC Hermes utilizando Generadores de Tráfico para tres Nodos

En la Figura. 9.7 se muestra en línea roja otro escenario de simulación en donde se pone a prueba tres nodos de la arquitectura planteada que son: S00, S10 y S11. En este escenario de pruebas el Generador (GT00) inyectará un

paquete de datos al Switch 00 (S00) con destino al Switch 11 (S11), que a su vez envía la información al Receptor (Rx11) el cual evalúa el tiempo total de la transmisión realizada. La información del S00 atraviesa el nodo S10 con destino al S11.

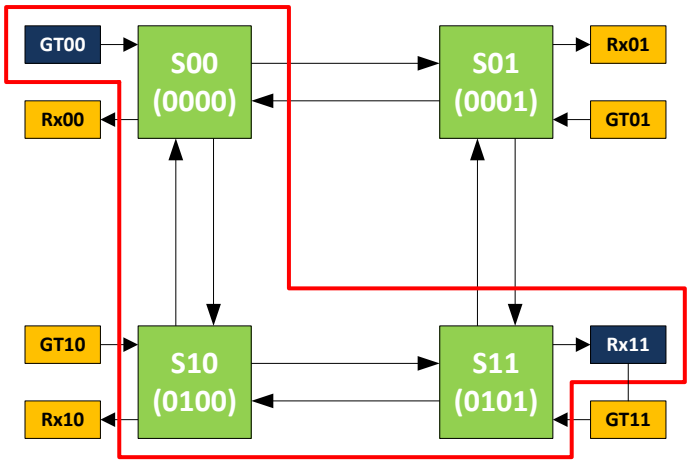


Figura. 9.7. Generadores y Receptores de Tráfico NoCgen conectados a Hermes para tres nodos

Antes de simular, se realizan cálculos teóricos que posteriormente serán comparados con los resultados del ISim (ver Tabla. 9.4). Estos cálculos se hacen con un tiempo de dispersión de 6.5µs, para tres nodos y con un tamaño variable de paquetes de 48 a 256 bits.

$$Latencia = (6.5 + 6.5 + 6.5) + (3 \times 2) = 25.5\mu s$$

No de Bits	Latencia un Nodo (µs)	Latencia tres Nodos	Tamaño Paquete	Tamaño por 2	Latencia en NoC (µs)
48	6,5	19,5	3	6	25,5
64	6,5	19,5	4	8	27,5
80	6,5	19,5	5	10	29,5
96	6,5	19,5	6	12	31,5
112	6,5	19,5	7	14	33,5

No de Bits	Latencia un Nodo (µs)	Latencia tres Nodos	Tamaño Paquete	Tamaño por 2	Latencia en NoC (µs)
128	6,5	19,5	8	16	35,5
144	6,5	19,5	9	18	37,5
160	6,5	19,5	10	20	39,5
176	6,5	19,5	11	22	41,5
192	6,5	19,5	12	24	43,5
208	6,5	19,5	13	26	45,5
224	6,5	19,5	14	28	47,5
240	6,5	19,5	15	30	49,5
256	6,5	19,5	16	32	51,5

Tabla. 9.4. Cálculos de Latencia en Hermes para tres nodos

Las simulaciones que a continuación se muestran prueban a NoC Hermes en la transmisión de paquetes con los siguientes tamaños:

- **Transmisión de 1 paquete de 5 flits desde S00 hacia S11:**

Figura. 9.8. Simulación en ISim del modelo de pruebas para una transmisión de 5 flits

La Figura. 9.8, muestra un recuadro rojo que engloba la latencia total medida con cuatro valores [V1, V2, V3, V4] donde cada valor corresponde a cada receptor de tráfico, cuyo valor puede ser comparado con la Tabla. 9.4, que contiene los valores calculados. En el mismo gráfico se muestra en verde los datos transmitidos por S00 (data_inlocal[0]) y los datos receptados por S11 (data_outlocal[3]), el resto de señales que están en azul corresponden al Handshake que se realiza para la confirmación en la transmisión y recepción de los datos.

La latencia medida en la Figura. 9.8 equivale a la diferencia entre: X1: 40.5µs – X2: 10.5µs = ΔX: 30µs.

- **Transmisión de 1 paquete de 7 flits desde S00 hacia S11**

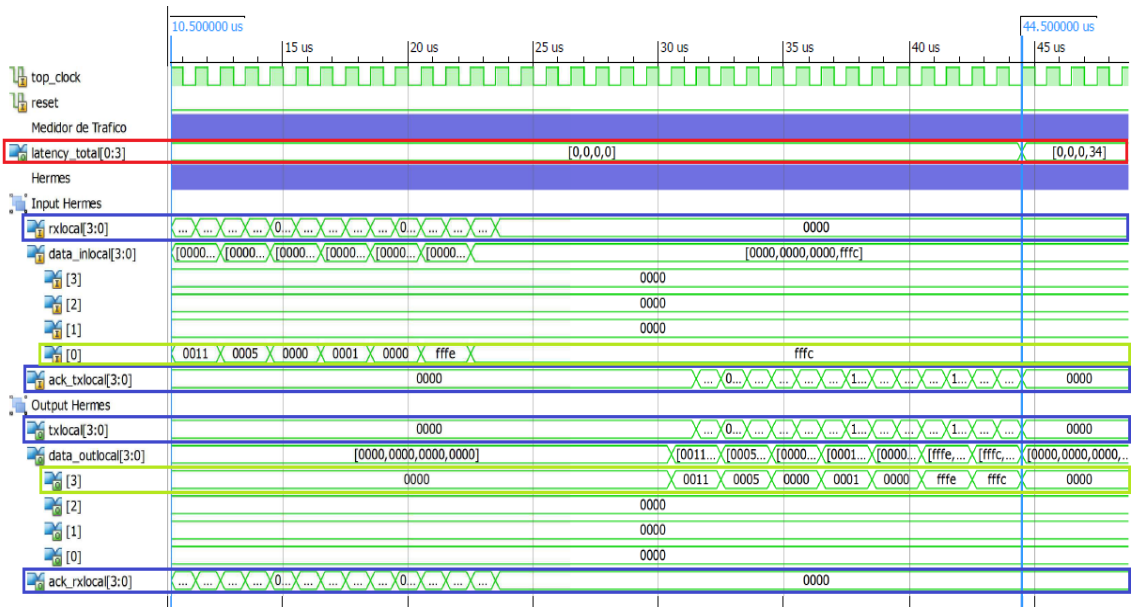


Figura. 9.9. Simulación en ISim del modelo de pruebas para una transmisión de 7 flits

La latencia medida en la Figura. 9.9 equivale a la diferencia entre: X1: 44.5µs – X2: 10.5µs = ΔX: 34µs.

- **Transmisión de 1 paquete de 9 flits desde S00 hacia S11:**

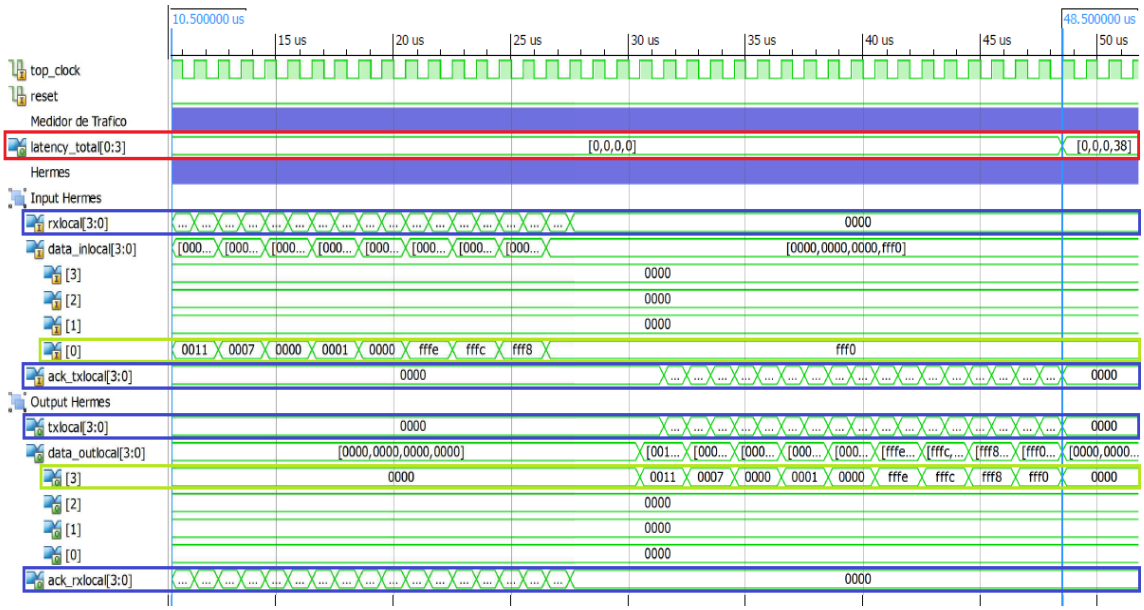


Figura. 9.10. Simulación en ISim del modelo de pruebas para una transmisión de 9 flits

La latencia medida en la Figura. 9.10 equivale a la diferencia entre: X1: 48.5µs – X2: 10.5µs = ΔX: 38µs.

La Figura. 9.9 y Figura. 9.10, son muy similares a la Figura. 9.8 la principal diferencia radica en el valor de latencia debido al incremento de la longitud del paquete.

9.1.1.4. Resultados obtenidos en las simulaciones a dos y tres nodos

Las tablas que se muestran a continuación son los resultados que se han obtenido al simular paquetes compuestos desde 48 hasta 256 bits.

La Tabla. 9.5 muestra los resultados obtenidos respecto a la latencia en la simulación de una arquitectura NoC Hermes entre dos nodos.

No de Bits	Tamaño Paquete	Latencia Simulada (µs)
48	3	20
64	4	22
80	5	23
96	6	25
112	7	27
128	8	29
144	9	31
160	10	33
176	11	35
192	12	37
208	13	39
224	14	41
240	15	43
256	16	45

Tabla. 9.5. Resultados obtenidos en simulación de NoC Hermes entre dos nodos

La Tabla. 9.6 muestra los resultados obtenidos respecto a la latencia en la simulación de una arquitectura NoC Hermes entre tres nodos.

No de Bits	Tamaño Paquete	Latencia Simulada (µs)
48	3	27
64	4	29
80	5	30
96	6	32
112	7	34
128	8	36
144	9	38
160	10	40
176	11	42
192	12	44
208	13	46
224	14	48
240	15	50

No de Bits	Tamaño Paquete	Latencia Simulada (µs)
256	16	52

Tabla. 9.6. Resultados obtenidos en simulación de NoC Hermes entre tres nodos

9.1.2. Pruebas de Transmisión de Datos en Bus FSL

En este apartado se trata la simulación de tráfico de información realizada en un bus FSL el cual comunica un procesador MicroBlaze con un IP-Core de diseño específico (interfaz de red). Por lo tanto a continuación se mostrará las simulaciones realizadas.

9.1.2.1. Modelo de Pruebas

En principio se deseaba medir la latencia de transmisión de datos entre dos MicroBlaze. Sim embargo, dado que a través de un MicroBlaze solo se puede simular una transmisión de datos y para que otro MicroBlaze recepte datos, se requiere realizar cambios en la configuración interna del hardware del mismo. Hacer esto genera errores en la simulación. A fin de medir la latencia, fue necesario transmitir datos desde un MicroBlaze a otro elemento utilizando FSL como medio de conexión al elemento escogido que fue la interfaz de red diseñada para NoC Hermes.

La Figura. 9.11 muestra un bus FSL conectado entre el MicroBlaze y la interfaz de red (NI_Core). La Interfaz de red fue modificada para poder hacer más fácil la simulación con el Bus FSL, de esta forma el NI consume los paquetes transmitidos permitiendo medir la latencia en el FSL_IN.

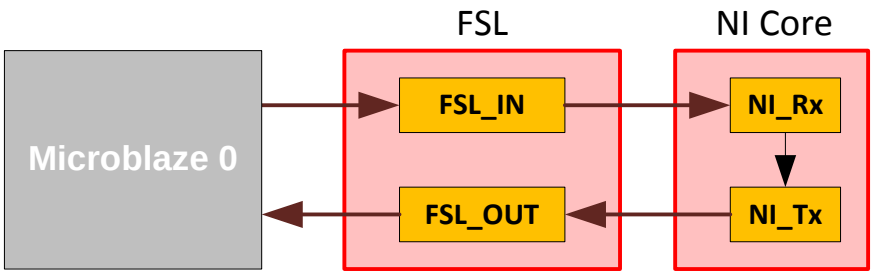


Figura. 9.11. Procesadores MicroBlaze interconectado por un Bus FSL

- **Transmisión de Información**

A través de FSL se realizaron transmisiones de datos correspondientes a los mostrados en las siguientes tablas, para simular una transmisión de 5, 7 y 9 flits, con el fin de comparar los resultados obtenidos con NoC.

Paquete de 5 Datos Transmitidos a través de FSL			
N° Flit	Estructura del Paquete	Flit en Binario	Flit en Hexadecimal
1	Dato 1	0000000000010000	0x0010
2	Dato 2	0000000000000011	0x0003
3	Dato 3	1111111111111111	0xffff
4	Dato 4	1111111111111110	0xfffe
5	Dato 5	1111111111111100	0xfffd

Tabla. 9.7. Paquete de transmisión de 5 flits ingresado en simulación con FSL

Paquete de 7 Datos Transmitidos a través de FSL			
N° Flit	Estructura del Paquete	Flit en Binario	Flit en Hexadecimal
1	Dato 1	0000000000010000	0x0010
2	Dato 2	0000000000000111	0x0003
3	Dato 3	1111111111111111	0xffff

Paquete de 7 Datos Transmitidos a través de FSL			
N° Flit	Estructura del Paquete	Flit en Binario	Flit en Hexadecimal
4	Dato 4	1111111111111110	0xfffe
5	Dato 5	1111111111111101	0xfffd
6	Dato 6	1111111111111100	0xfffc
7	Dato 7	1111111111111011	0xfffb

Tabla. 9.8. Paquete de transmisión de 7 flits ingresado en simulación con FSL

Paquete de 9 Datos Transmitidos a través de FSL			
N° Flit	Estructura del Paquete	Flit en Binario	Flit en Hexadecimal
1	Dato 1	0000000000010000	0x0010
2	Dato 2	0000000000001001	0x0009
3	Dato 3	1111111111111111	0xffff
4	Dato 4	1111111111111110	0xfffe
5	Dato 5	1111111111111101	0xfffd
6	Dato 6	1111111111111100	0xfffc
7	Dato 7	1111111111111011	0xfffb
8	Dato 8	1111111111111010	0xfffa
9	Dato 9	1111111111111001	0xfff9

Tabla. 9.9. Paquete de transmisión de 9 flits ingresado en simulación con FSL

Las tablas muestran todos los datos que se transmitieron en una simulación de prueba utilizando el bus FSL. Obsérvese además que en el caso del FSL el paquete mínimo a ser transmitido es de un sólo flit, a diferencia de NoC Hermes en donde el paquete mínimo que se puede transmitir está compuesto de tres flits (dirección, tamaño y datos).

- **Latencia en bus FSL**

La transmisión de datos en el bus FSL es de dos ciclos de reloj, uno para la escritura de datos en el maestro y otro para lectura en el lado del esclavo [88].

$$Latencia = R_i + (P \times 2)$$

Por lo tanto, la ecuación anterior describe la latencia en el bus FSL que es similar a la descrita para NoC Hermes, en donde:

R_i: Tiempo de dispersión.
P: Tamaño del paquete.

Destacando que el valor del tiempo de dispersión (R_i) para este caso de simulación corresponde a 2µs.

9.1.2.2. Escenario de Pruebas en Bus FSL Transmitiendo Datos

En la Figura. 9.12 se muestra en azul el escenario de simulación en donde se pone a prueba al bus FSL. En este escenario de pruebas el MicroBlaze inyecta un paquete de datos al FSL_IN con destino al NI_Rx el cual consume los datos transmitidos, lo que permite medir latencia con la ayuda del simulador ISim.

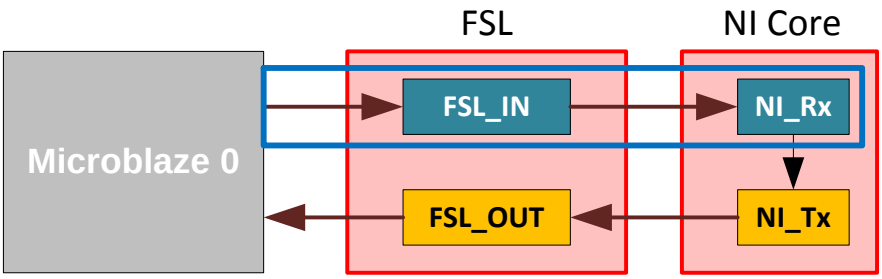


Figura. 9.12. Escenario de Simulación de un Procesadores MicroBlaze interconectado por un Bus FSL

Antes de simular, se realizan cálculos teóricos que posteriormente serán comparados con los resultados del ISim (ver Tabla. 9.10). Estos cálculos se hacen con un tiempo de dispersión de 2µs, con un tamaño variable de paquetes de 16 a 256 bits.

$$Latencia = 2 + (5 \times 2) = 12\mu s$$

No de Bits	Latencia FIFO FSL (µs)	Tamaño Paquete	Tamaño por 2	Latencia FSL (µs)
16	2	1	2	4
32	2	2	4	6
48	2	3	6	8
64	2	4	8	10
80	2	5	10	12
96	2	6	12	14
112	2	7	14	16
128	2	8	16	18
144	2	9	18	20
160	2	10	20	22
176	2	11	22	24
192	2	12	24	26
208	2	13	26	28
224	2	14	28	30
240	2	15	30	32
256	2	16	32	34

Tabla. 9.10. Cálculos de latencia para un Bus FSL que interconecta dos MicroBlaze

Los cálculos de latencia mostrados en la Tabla. 9.10 serán comparados con las simulaciones que se muestran a continuación. En las siguientes figuras se puede apreciar el FSL_IN, que es uno de los dos buses FSL que se utiliza para transferir información desde el MicroBlaze hacia el NI. En estas figuras se observan también varios puertos que constituyen al mencionado bus, de ellos los más relevantes son los que se encuentran encerrados en un rectángulo de color. En color morado se identifica el FSL_M_DATA y el FSL_M_WRITE, del lado del maestro, y en anaranjado se encuentran el FSL_S_DATA, FSL_S_READ y el FSL_S_EXISTS. En cuanto al FSL_M_FULL, FSL_RST y el SYS_RST, no se los toma en cuenta ya que en este caso nunca se activan.

- **Transmisión de 1 paquete de 5 flits**

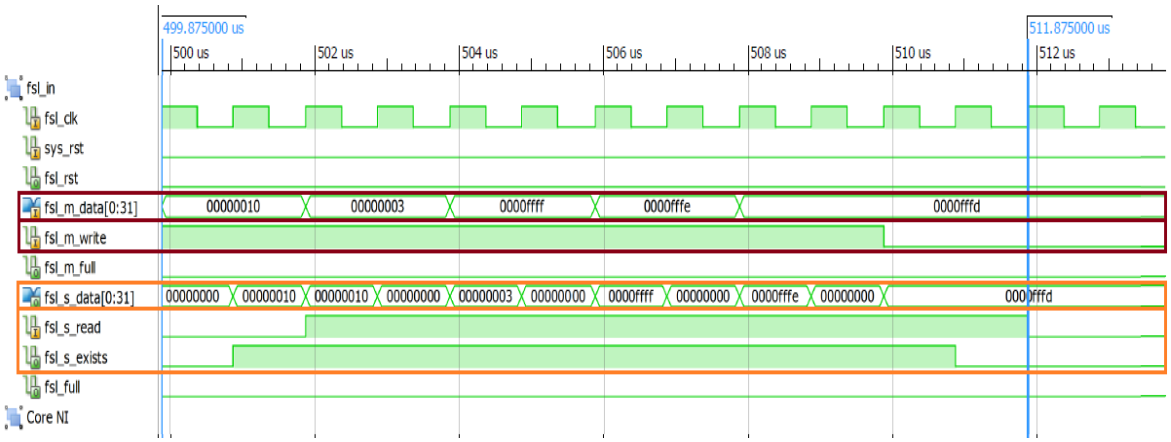


Figura. 9.13. Simulación en ISim del modelo de pruebas para una transmisión de 5 flits

La latencia medida en la Figura. 9.13 equivale a la diferencia entre: X1: 511.875µs – X2: 499.875µs = ΔX: 12µs.

- **Transmisión de 1 paquete de 7 flits**

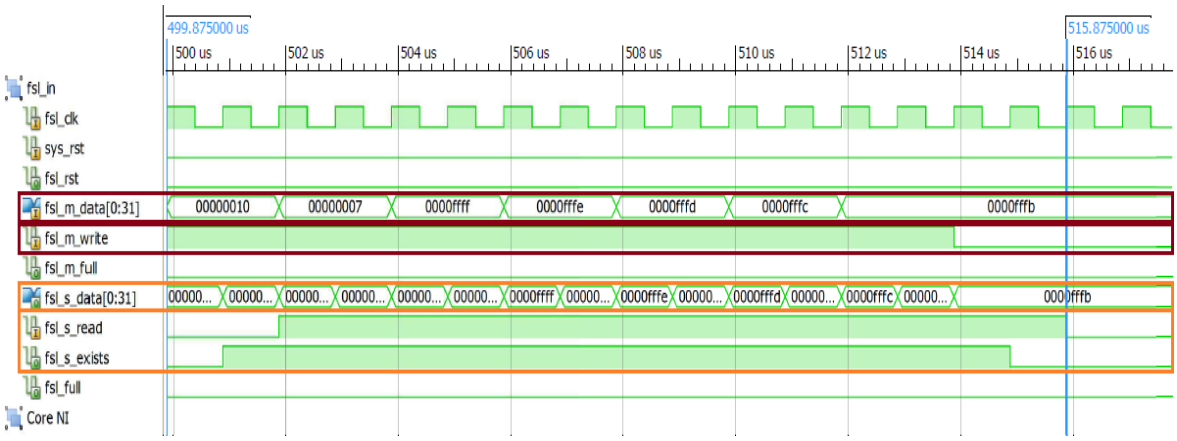


Figura. 9.14. Simulación en ISim del modelo de pruebas para una transmisión de 7 flits

La latencia medida en la Figura. 9.13 equivale a la diferencia entre: X1: 515.875µs – X2: 499.875µs = ΔX: 16µs.

- **Transmisión de 1 paquete de 9 flits**

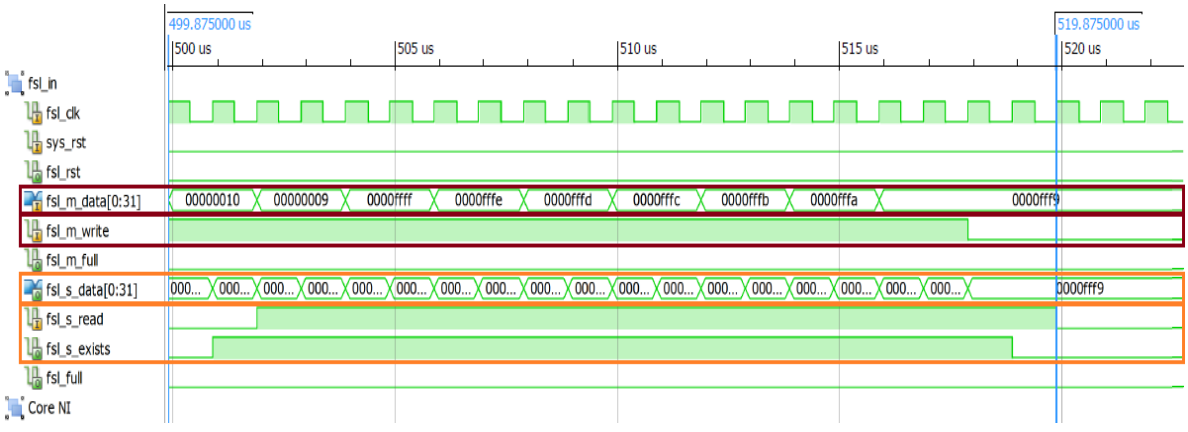


Figura. 9.15. Simulación en ISim del modelo de pruebas para una transmisión de 9 flits

La latencia medida en la Figura. 9.13 equivale a la diferencia entre: X1: 519.875µs – X2: 499.875µs = ΔX: 20µs.

Las simulaciones mostradas en las Figura. 9.13, Figura. 9.14 y Figura. 9.15 demuestran el retraso de 2µs en el FSL que se menciona en su respectiva hoja técnica, dado principalmente en la memoria FIFO interna a la que le toma 1µs en leer el bus maestro y 1µs en escribir en el bus esclavo del FSL. [88]

9.1.2.3. Resultados Obtenidos en la Simulación con FSL

La Tabla. 9.11 muestra los resultados obtenidos al simular el bus FSL con transmisión de paquetes desde 16 hasta 256 bits.

No de Bits	Tamaño Paquete	Simulación FSL (µs)
16	1	4
32	2	6
48	3	8
64	4	10
80	5	12
96	6	14
112	7	16
128	8	18
144	9	20
160	10	22
176	11	24
192	12	26
208	13	28
224	14	30
240	15	32
256	16	34

Tabla. 9.11. Resultados obtenidos en simulación con FSL

9.1.3. Pruebas de Transmisión de Datos en NoC Hermes con Interfaz de Red

Este apartado trata sobre la simulación realizada sobre los Switches de Hermes conectados a los respectivos NI (WiNi_Core) explicados en el **Capítulo 6**

y **Capítulo 7**. A continuación se mostrará las simulaciones realizadas para demostrar la implementación de WiNi_Core dentro una arquitectura MPSoC interconectada por NoC Hermes.

9.1.3.1. Modelo de Pruebas

La Figura. 9.16, muestra a los Switches de Hermes conectados a las Interfaces de Red (WiNi_Core). Como se mencionó en el **Capítulo 6**, WiNi_Core emplea buses FSL para completar la comunicación necesaria para intercambiar información entre MicroBlaze y NoC Hermes. Por esta razón se utiliza buses FSL como inyectores de tráfico para realizar la medición de latencia.

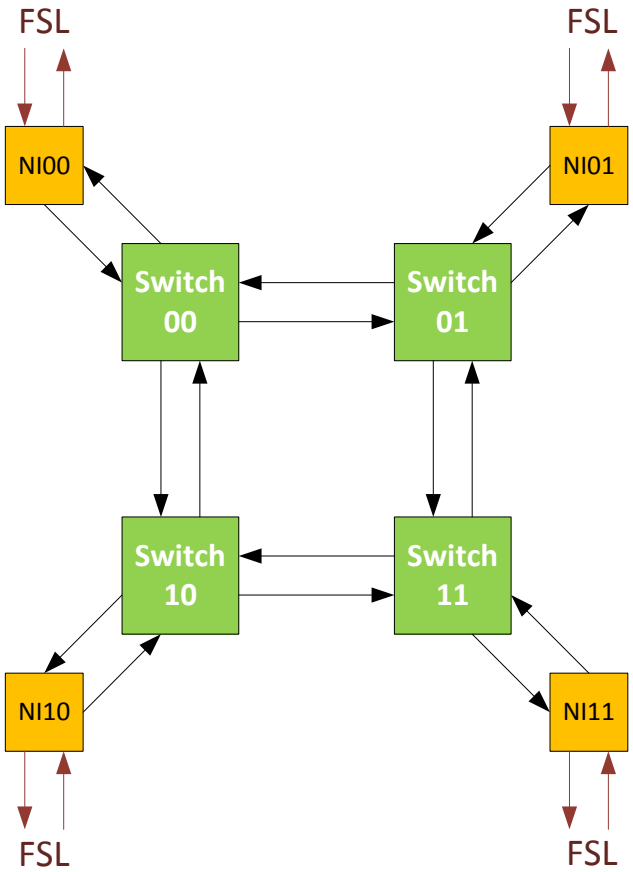


Figura. 9.16. Interfaces de red WiNi_Core conectados a Switches Hermes

9.1.3.2. Transmisión de Información

Para realizar pruebas sobre el modelo escogido se utilizará la aplicación de esteganografía descrita en el **Capítulo 8**. En la transmisión de datos de la aplicación de esteganografía se utiliza paquetes de 5 flits para enviar la información. En la Figura. 9.17, se plantea la estructura del paquete que se simula, en donde R, G y B representan los colores rojo verde y azul del pixel.

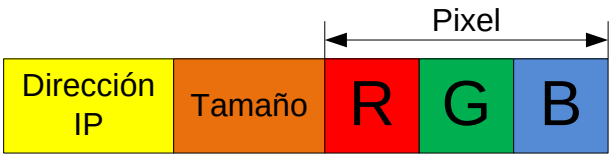


Figura. 9.17. Estructura del paquete a simular

La estructura del paquete de 5 flits se muestra en la Tabla. 9.12, en donde el único parámetro variable es la dirección de destino dado por el algoritmo XY.

Estructura del Paquete que Constituye un Pixel		
N° Flit	**Estructura del Paquete**	**Contenido del Paquete**
1	Dirección IP	---
2	Tamaño del Paquete	0000000000000011
3	Dato 3	R
4	Dato 4	G
5	Dato 5	B

Tabla. 9.12. Paquete de 5 flits que transmite un pixel

De manera que en dicha aplicación por cada pixel se arman paquetes de 80 bits que serán transmitidos *n* cantidad de veces según el tamaño de pixeles de la imagen con la que se está trabajando.

9.1.3.3. Escenario de Simulación

Tomando como principio las simulaciones realizadas en las pruebas con NoC Hermes sin NI, se plantea el siguiente escenario que simula la trasmisión de un paquete de 5 flits. El tráfico de información se lo ejecuta desde el NI00 con destino al NI10 (ver Figura. 9.18). Para insertar información se utilizará a los buses FSL como inyectores de tráfico. Cabe señalar que en las simulaciones mostradas posteriormente, la medición de latencia es desde el NI00 hasta el NI10 faltando añadir la latencia causada por los buses FSL.

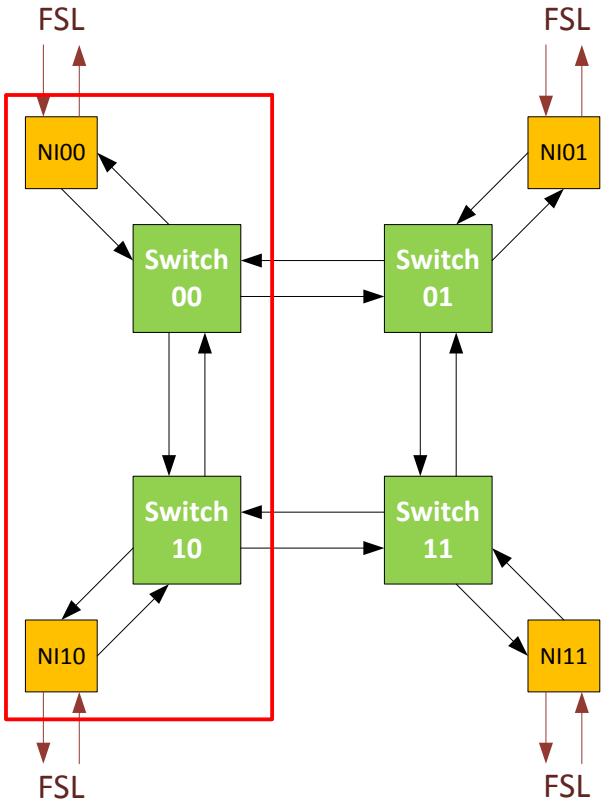


Figura. 9.18. Escenario de simulación para Interfaz de Red Conectado con NoC Hermes

La simulación permite obtener un único resultado que luego será comparado con el obtenido en el modelo de pruebas mostrado en la Figura. 9.3 para una transmisión de 80 bits.

9.1.3.4. Simulación de Switches Hermes con sus Respectivas Interfaces de Red

La siguiente simulación se realiza según lo indicado en la Figura. 9.18, en donde el paquete de 5 flits (80 bits) se transmite desde el NI00 con destino al NI10. Estas mediciones pueden ser apreciadas en la Figura. 9.19.

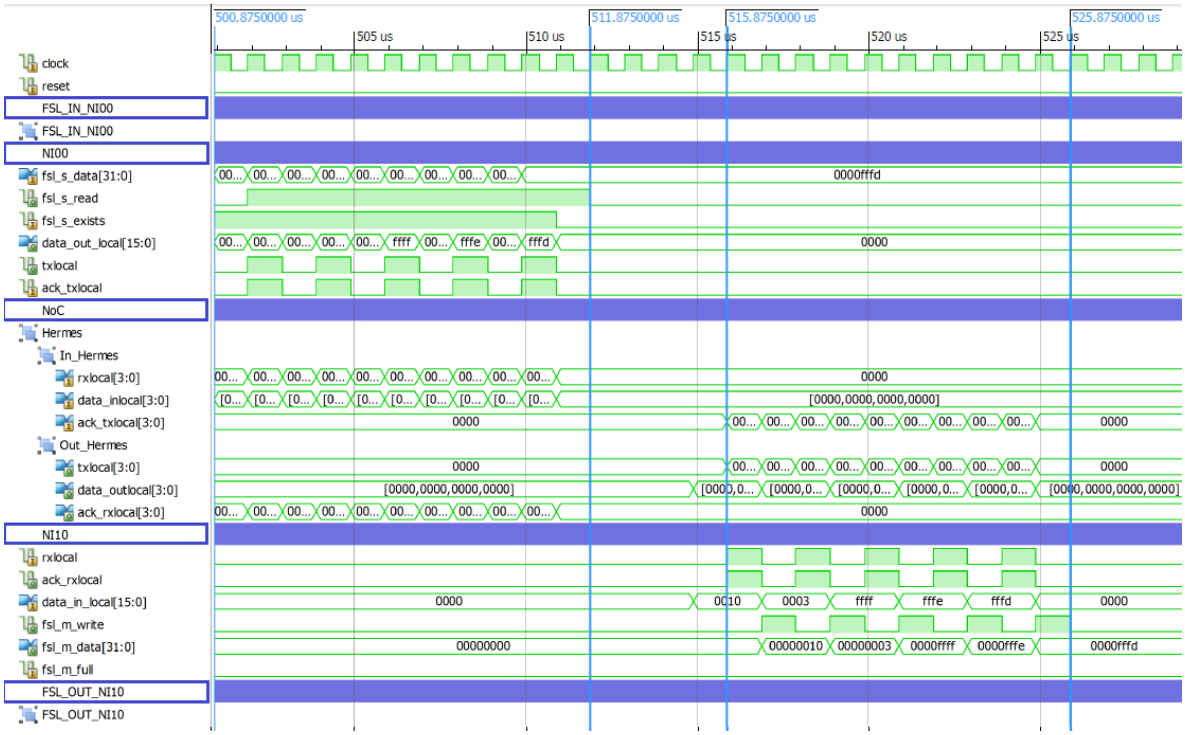


Figura. 9.19. Simulación en ISim del modelo de pruebas para una transmisión de 5 flits

9.1.3.5. Resultados Obtenidos en la Simulación de Switches Hermes con sus Respectivas Interfaces de Red

La Tabla. 9.13 muestra el resultado obtenido de la simulación realizada con los Switches Hermes conectados a sus respectivas Interfaces de Red.

No de Bits	Tamaño del Paquete	Latencia Simulada (µs)
80	5	25

Tabla. 9.13. Resultado obtenido en simulación de NoC Hermes con WiNi_Core

En la Tabla. 9.14 se muestra las latencias obtenidas sobre los NI simulados.

No de Bits	Tamaño del Paquete	Latencia NI00 (µs)	Latencia NI10 (µs)
80	5	11	10

Tabla. 9.14. Resultados obtenidos en latencia sobre los NI00 y NI10

9.2. ANÁLISIS DE RESULTADOS EN LATENCIA

Para el caso de análisis de resultados en latencia entre FSL y NoC Hermes se especifica que las pruebas obtenidas fueron realizadas por medio de simulaciones de punto a punto (ver Figura. 9.3). Este motivo se justifica debido a las limitaciones presentes en los generadores de tráfico, los cuales solo permiten trasmisiones de datos de un único origen a un único destino.

9.2.1. Análisis de Resultados para NoC Hermes en Dos y Tres Nodos

Para realizar el respectivo análisis de resultados primero se parte de los datos que han sido calculados y simulados. La Tabla. 9.15 muestra un resumen de todos los datos que se han obtenido para el análisis de resultados de NoC Hermes entre dos y tres nodos.

Núm. de Bits	Latencia NoC Hermes 2 Nodos			Latencia NoC Hermes 3 Nodos		
	Calculada (µs)	Simulación (µs)	Diferencia (µs)	Calculada (µs)	Simulación (µs)	Diferencia (µs)
48	19	20	1	25,5	27	1,5
64	21	22	1	27,5	29	1,5
80	23	23	0	29,5	30	0,5
96	25	25	0	31,5	32	0,5
112	27	27	0	33,5	34	0,5
128	29	29	0	35,5	36	0,5
144	31	31	0	37,5	38	0,5
160	33	33	0	39,5	40	0,5
176	35	35	0	41,5	42	0,5
192	37	37	0	43,5	44	0,5
208	39	39	0	45,5	46	0,5
224	41	41	0	47,5	48	0,5
240	43	43	0	49,5	50	0,5
256	45	45	0	51,5	52	0,5

Tabla. 9.15. Resume de valores de latencia calculados y simulados

En la Tabla. 9.15 se ilustra una diferencia en latencia entre los cálculos realizados y los valores obtenidos en las simulaciones. Para las transmisiones con cargas de 48 y 64 bits existe la diferencia de 1µs que se da entre los valores calculados y simulados. En tanto que para transmisiones con mayor número de bits la diferencia se reduce a 0 para la latencia en NoC Hermes a dos nodos y 0.5 para la latencia en NoC Hermes a tres nodos. Estos cambios en la diferencia entre los valores calculados y simulados se dan por la carga de paquetes, los cuales al ir creciendo en tamaño prolongan el tiempo requerido para su transmisión. Además el tiempo de transmisión de un paquete de datos también depende mucho de los nodos presentes en el enrutamiento de la información.

En la Figura. 9.20 se puede apreciar las curvas de tendencia de latencia simulada en NoC Hermes para dos y tres nodos.

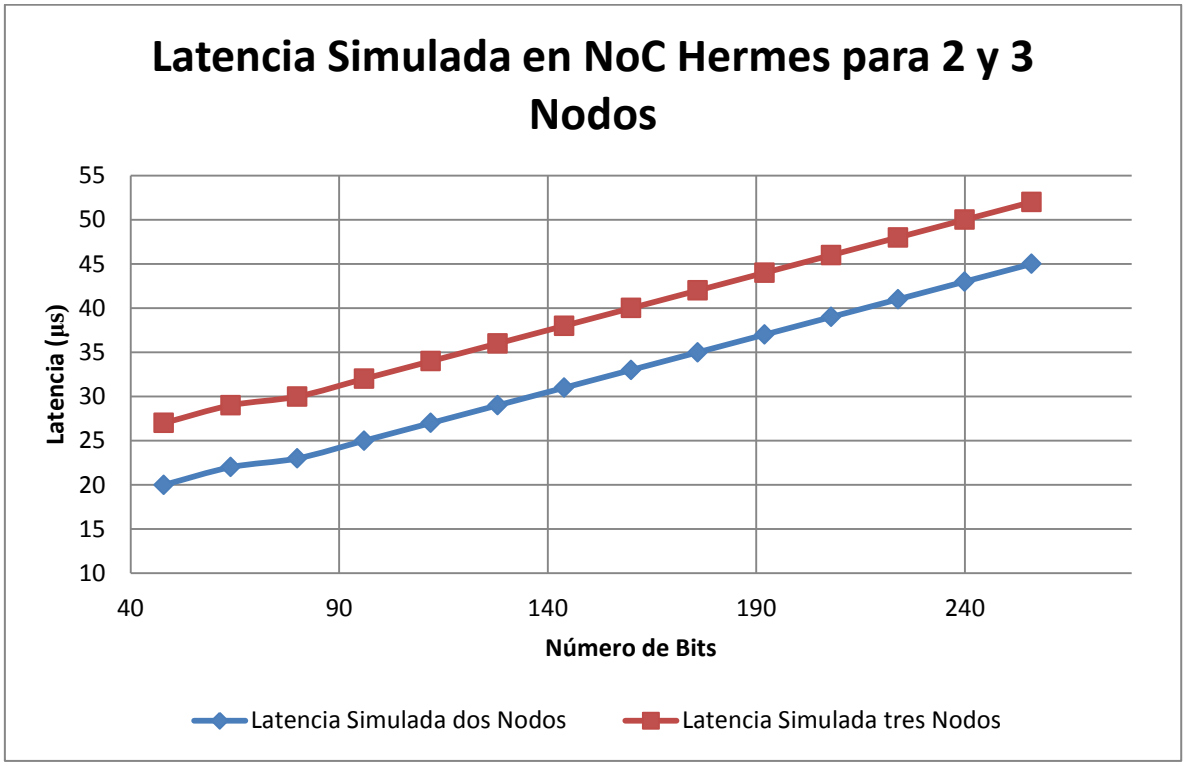


Figura. 9.20. Gráfica de comparación entre NoC Hermes para 2 y 3 nodos

Como se puede observar en la Figura. 9.20 existe el incremento en latencia por aumento del número de bits transmitidos. También existe un diferencial constante debido al número de nodos. Por lo tanto se deben realizar pruebas en topologías de más de cuatro nodos con el objetivo de verificar si la diferencia se mantiene constante o cambia.

9.2.2. Análisis de Resultados entre FSL y NoC Hermes a Dos Nodos

La Tabla. 9.16 muestra el comportamiento entre dos elementos de procesamiento utilizando dos medios de comunicación diferentes. Estos medios de comunicación son NoC Hermes y Bus FSL.

Núm. de Bits	Latencia NoC Hermes 2 Nodos		Latencia FSL	
	Calculada (µs)	Simulación (µs)	Calculada (µs)	Simulación (µs)
48	19	20	8	8
64	21	22	10	10
80	23	23	12	12
96	25	25	14	14
112	27	27	16	16
128	29	29	18	18
144	31	31	20	20
160	33	33	22	22
176	35	35	24	24
192	37	37	26	26
208	39	39	28	28
224	41	41	30	30
240	43	43	32	32
256	45	45	34	34

Tabla. 9.16. Resumen de valores de latencia calculados y simulados

En la Tabla. 9.16 se muestran los datos recolectados de los escenarios de pruebas tanto para NoC Hermes como para el bus FSL. La diferencia entre la latencia del bus FSL y la latencia de NoC Hermes a dos nodos es 11µs. Esta diferencia se mantiene constante sin importar el número de bits de transmisión.

La Figura. 9.21 muestra los valores de los datos simulados para latencia entre NoC Hermes a dos nodos y bus FSL. Esto se debe principalmente a que el tiempo de dispersión (R_i) es mayor en NoC Hermes con respecto a FSL. Sin embargo como se mencionó las simulaciones fueron hechas de punto a punto. Por este motivo se evidencia que es imperceptible evaluar el rendimiento en latencia para una arquitectura de multiprocesamiento MPSoC interconectado por NoC Hermes de 2x2 en topología 2D-Mesh. De manera que se recomienda el desarrollo de trabajos futuros sobre arquitecturas MPSoC más grandes en donde se pueda percibir el potencial real en latencia de NoC Hermes con respecto a buses FSL.

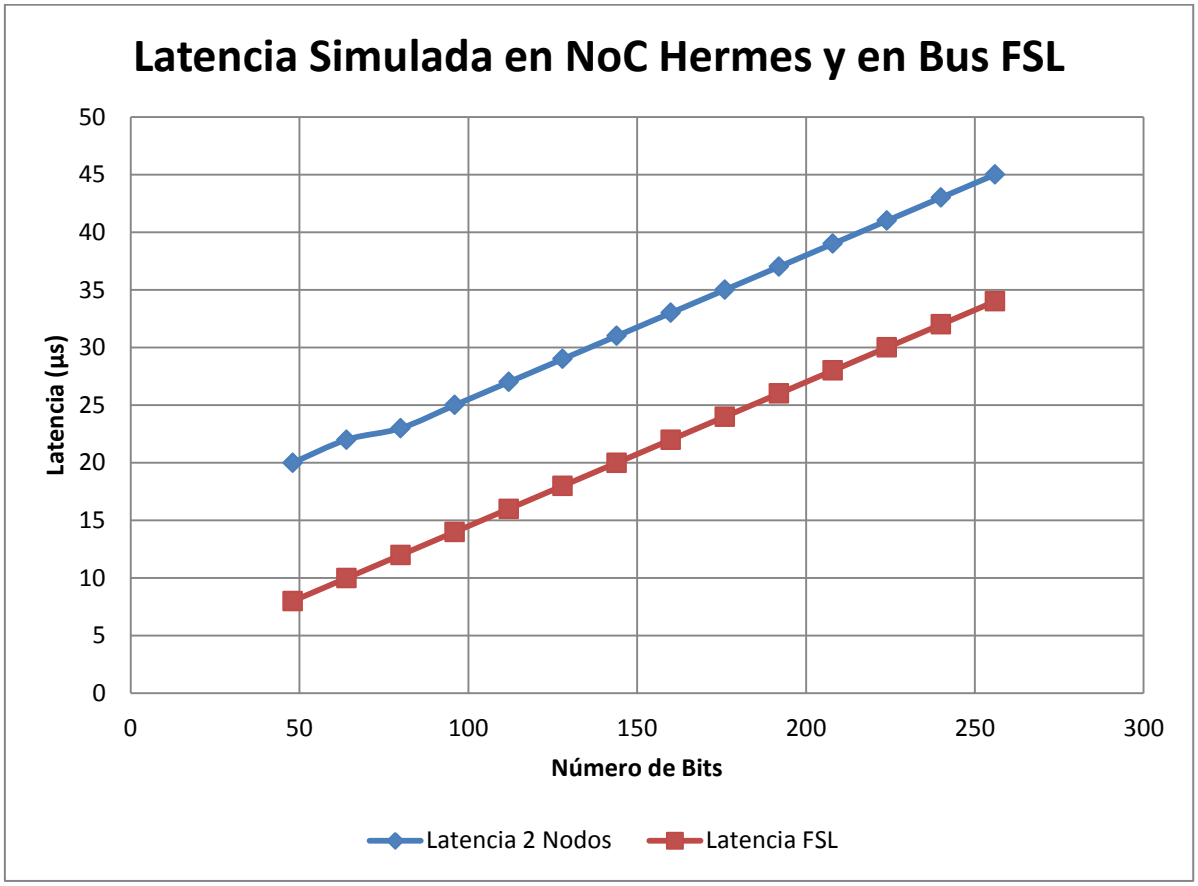


Figura. 9.21. Grafica de comparación entre NoC Hermes y FSL

9.2.3. Análisis de Resultados entre FSL y NoC Hermes con Interfaz de Red

La Figura. 9.22 muestra varios escenarios de pruebas realizados para medir latencia. En cuadro de color rojo se muestra las pruebas realizadas en NoC Hermes a dos nodos y en cuadro amarillo las pruebas sobre NoC Hermes con Interfaz de red. La primera prueba fue realizada con paquetes de tamaño desde 48 hasta 256 bits. En la otra prueba realizó una sola transmisión de 80 bits que simula un pixel de la aplicación de esteganografía. Todos los escenarios mencionados fueron realizados con transmisiones de punto a punto.

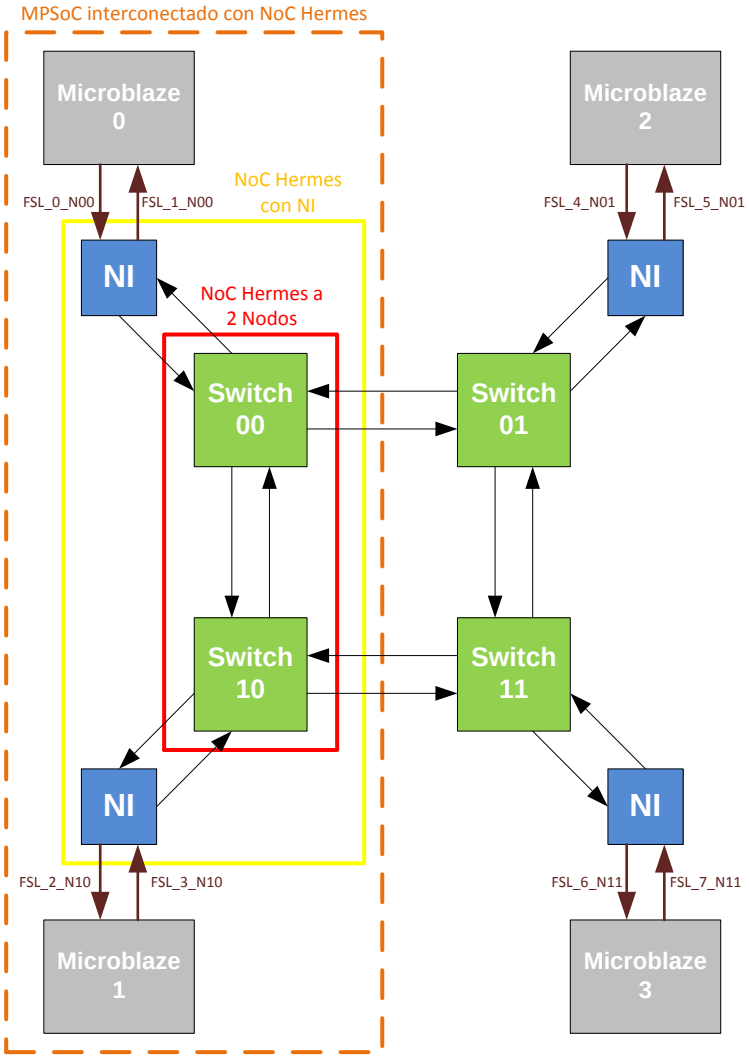


Figura. 9.22. Arquitectura MPSoC de cuatro procesadores MicroBlaze interconectados por NoC Hermes usando Interfaces de Red

La Tabla. 9.17 muestra los datos obtenidos de las simulaciones con NoC Hermes a dos nodos y NoC Hermes con interfaz de red para un paquete de 80 bits. Dado que solo se hicieron simulaciones, en donde aparentemente se incrementa la latencia, los valores obtenidos no interpretan la realidad.

Latencias Simuladas		
Núm. Bits	**NoC Hermes sin NI (µs)**	**NoC Hermes con NI (µs)**
80	23	25

Tabla. 9.17. Datos de simulación en NoC con NI y NoC sin NI

De la Tabla. 9.17 se puede concluir que la latencia en NoC Hermes con NI se incrementa en 2µs adicionales. Por lo tanto se demuestra que aparentemente la interfaz de red diseñada para el proyecto de tesis incrementa la latencia en el valor mencionado.

En conclusión lo ideal para obtener mejores resultados sería llegar a realizar pruebas sobre toda la arquitectura de multiprocesamiento tal como se observa en el cuadro tomate de la Figura. 9.22. Estas pruebas deberían ser realizadas directamente sobre la aplicación en donde se medirían valores de latencia más fiables que los valores de latencias que han sido simulados.

Por otra parte la simulación en NoC Hermes con Interfaz de Red también permite tomar en cuenta los valores de latencia sobre el NI diseñado. La Tabla. 9.18 muestra las latencias de los NI00 y NI10.

Latencias Simuladas			
Núm. Bits	**FSL (µs)**	**NI00 (µs)**	**NI10 (µs)**
80	12	11	10

Tabla. 9.18. Latencia en NI

Aunque lamentablemente se realizaron mediciones de latencia de punto a punto, en la Tabla. 9.18 se muestra que estas mediciones realizadas en simulaciones permitieron demostrar la latencia de 2µs en FSL. Al bus FSL le toma 1µs para escribir en el lado del maestro y 1µs en leer en el lado del esclavo. Esta latencia en total suma 12µs para un paquete de 80 bits. Por lo tanto la latencia en el NI00 corresponde a la latencia en lectura del bus FSL. Es decir, que le toma tan solo 1µs menos que al FSL para trasmitir información una vez que adquiere los datos. En otras palabras la latencia del NI00 depende mucho del tiempo de respuesta del esclavo del FSL ante datos presentes en su memoria FIFO interna.

Lo mismo se puede concluir con el NI10 pues en este caso a la memoria FIFO le toma 1µs en receptar nuevos datos escritos sobre el maestro del FSL.

9.3. PRUEBAS DE ESTIMACIÓN DE POTENCIA PARA ARQUITECTURAS INTERCONECTADAS POR NOC HERMES Y POR BUSES FSL

La herramienta XPA permite realizar diversas pruebas para determinar la estimación de potencia en cada una de las arquitecturas mediante la configuración de la tasa de conmutación tanto para Flip – Flops como para elementos I/O. XPA utiliza para su análisis de estimación de potencia varios archivos que contienen la información del diseño, conexiones internas y actividades de conmutación contenidos en los archivos *.NCD, *.XPA y *.PCF. A continuación se describe cada uno de los archivos:

- *Placed and routed design database* (NCD file): Contiene la base de datos de toda la configuración lógica, la información de enrutamiento y conexionado, así como las características específicas del FPGA sobre el cual se va a implementar el diseño.

- *Settings File* (XPA file): Contiene el ambiente de estimación y los datos de tasas de conmutación para cada elemento del diseño.

- *Physical Constraint* (PCF file): Contiene la información de toda la actividad de conmutación para elementos lógicos, elementos I/O y señales de reloj.

Los resultados obtenidos mediante la herramienta XPA representan la estimación de potencia dinámica del diseño. La energía dinámica consumida por un circuito digital CMOS es directamente proporcional a la actividad de conmutación. Esta energía varía en el tiempo con la actividad de conmutación, los niveles de voltaje utilizados, los recursos lógicos energizados, el conexionado, el

enrutamiento de elementos y la potencia estática disipada por los elementos energizados pero que no están configurados.

9.3.1. Estimación de Potencia Dinámica para una Arquitectura Interconectada por NoC Hermes

Para iniciar el proceso de estimación de potencia mediante XPA para una arquitectura interconectada por NoC Hermes, se debe ingresar manualmente en cada uno de los campos de análisis los valores estimados de potencia de acuerdo a cada escenario de prueba.

9.3.1.1. Modelo de Estimación

Para determinar un modelo de estimación se utiliza la variación de la tasa de conmutación tanto para Flip Flops, como para elementos I/O. En la Tabla. 9.19 se enlistan los escenarios de pruebas bajo los cuales se realizó la estimación de potencia para una arquitectura interconectada por NoC Hermes.

Tasa de Conmutación		
Tasa I/Os	**Tasa Flip Flops**	**Escenarios de Prueba**
12.5 %	12.5 %	Escenario 1
12.5 %	25 %	Escenario 2
25 %	12.5 %	Escenario 3
25 %	25 %	Escenario 4
50 %	50 %	Escenario 5
75 %	75 %	Escenario 6

Tabla. 9.19. Tasa de conmutación para diferentes escenarios de estimación de potencia

9.3.1.2. Resultados Obtenidos en Estimación de Potencia Dinámica de una Arquitectura Interconectada por NoC Hermes

Tasa de Conmutación			Estimación de Potencia (W)							
Tasa I/Os	**Tasa Flip Flops**	**Escenarios de Prueba**	Señales de Reloj	Bloques Lógicos	Señales Control	BRAM	MMCMS	I/Os	Potencia Estática	TOTAL
12.5 %	**12.5 %**	**Escenario 1**	0.258	0.028	0.032	0.374	0.215	1.256	3.000	5.163
12.5 %	**25 %**	**Escenario 2**	0.258	0.047	0.050	0.381	0.215	1.256	3.001	5.208
25 %	**12.5 %**	**Escenario 3**	0.258	0.033	0.045	0.374	0.215	1.293	3.001	5.219
25 %	**25 %**	**Escenario 4**	0.258	0.046	0.061	0.381	0.215	1.293	3.002	5.255
50 %	**50 %**	**Escenario 5**	0.258	0.100	0.122	0.393	0.215	1.366	3.007	5.460
75 %	**75 %**	**Escenario 6**	0.258	0.144	0.181	0.402	0.215	1.439	3.011	5.649

Tabla. 9.20. Escenarios de pruebas de estimación de potencia de una arquitectura interconectada por NoC Hermes

9.3.2. Estimación de Potencia Dinámica para una Arquitectura Interconectada por Buses FSL

9.3.2.1. Modelo de Estimación

Para determinar un modelo de estimación se utiliza la variación de la tasa de conmutación tanto para Flip Flops, como para elementos I/O. en la Tabla. 9.21 se enlistan los escenarios de pruebas bajo los cuales se realizó la estimación de potencia para una arquitectura interconectada por buses FSL.

Tasa de Conmutación		
Tasa I/Os	**Tasa Flip Flops**	**Escenarios de Prueba**
12.5 %	12.5 %	Escenario 1
12.5 %	25 %	Escenario 2
25 %	12.5 %	Escenario 3
25 %	25 %	Escenario 4
50 %	50 %	Escenario 5
75 %	75 %	Escenario 6

Tabla. 9.21. Tasa de conmutación para diferentes escenarios de estimación de potencia

9.3.2.2. Resultados Obtenidos en Estimación de Potencia Dinámica de una Arquitectura Interconectada por Buses FSL

Tasa de Conmutación			Estimación de Potencia (W)							
Tasa I/Os	**Tasa Flip Flops**	**Escenarios de Prueba**	Señales de Reloj	Bloques Lógicos	Señales Control	BRAM	MMCMS	I/Os	Potencia Estática	TOTAL
12.5 %	**12.5 %**	**Escenario 1**	0.232	0.026	0.032	0.391	0.215	1.256	3.000	5.152
12.5 %	**25 %**	**Escenario 2**	0.232	0.044	0.049	0.398	0.215	1.256	3.001	5.194
25 %	**12.5 %**	**Escenario 3**	0.232	0.031	0.047	0.391	0.215	1.293	3.001	5.209
25 %	**25 %**	**Escenario 4**	0.232	0.049	0.062	0.398	0.215	1.293	3.002	5.250
50 %	**50 %**	**Escenario 5**	0.232	0.094	0.123	0.411	0.215	1.366	3.006	5.446
75 %	**75 %**	**Escenario 6**	0.232	0.136	0.182	0.420	0.215	1.439	3.011	5.635

Tabla. 9.22. Estimación de potencia de una arquitectura interconectada por Buses FSL

9.4. PRUEBAS DE ESTIMACIÓN DE CONSUMO TOTAL DE POTENCIA PARA ARQUITECTURAS INTERCONECTADAS POR NOC HERMES Y POR BUSES FSL IMPLEMENTADAS SOBRE FPGA

La herramienta XPE permite la estimación de consumo total de potencia en el chip para cada una de las arquitecturas, haciendo referencia al cálculo de disipación de energía térmica consumida internamente dentro del FPGA. La disipación de energía térmica permite la habilitación de los dispositivos, así como su conmutación.

La estimación de consumo total de potencia en el chip definida por XPE es igual a la sumatoria de la potencia estática de los dispositivos y la potencia estática del diseño. En donde, la potencia estática de los dispositivos representa la energía necesaria para que un elemento funcione y esté disponible para su configuración. Mientras tanto la potencia estática del diseño representa el consumo adicional de energía cuando el dispositivo ya se encuentra configurado, pero todavía no existe una actividad de conmutación programada. La potencia estática del diseño incluye la energía estática de los elementos I/O y de los gestores de señales de reloj.

Para iniciar el proceso de estimación de consumo total de potencia en el chip mediante XPE, se debe exportar el archivo *.NCD que contiene toda la configuración lógica, la información de enrutamiento y las características específicas del FPGA sobre el cual se va a implementar el diseño.

9.4.1. Estimación de Consumo Total de Potencia para Arquitecturas Interconectadas por NoC Hermes

La estimación de consumo total de potencia para una arquitectura interconectada por NoC Hermes implementada sobre la FPGA ML605 Virtex-6 mediante la herramienta XPE se la realizará con tasas de conmutación y tasas de habilitación fijas representadas en Tabla. 9.23 y Tabla. 9.24.

Tasa de Conmutación	
Elementos Lógicos	12.5 %
Bloques de RAM	50 %
DSP	12.5 %
I/O	12.5 %

Tabla. 9.23. Tasa de conmutación para arquitectura NoC Hermes en consumo total de potencia

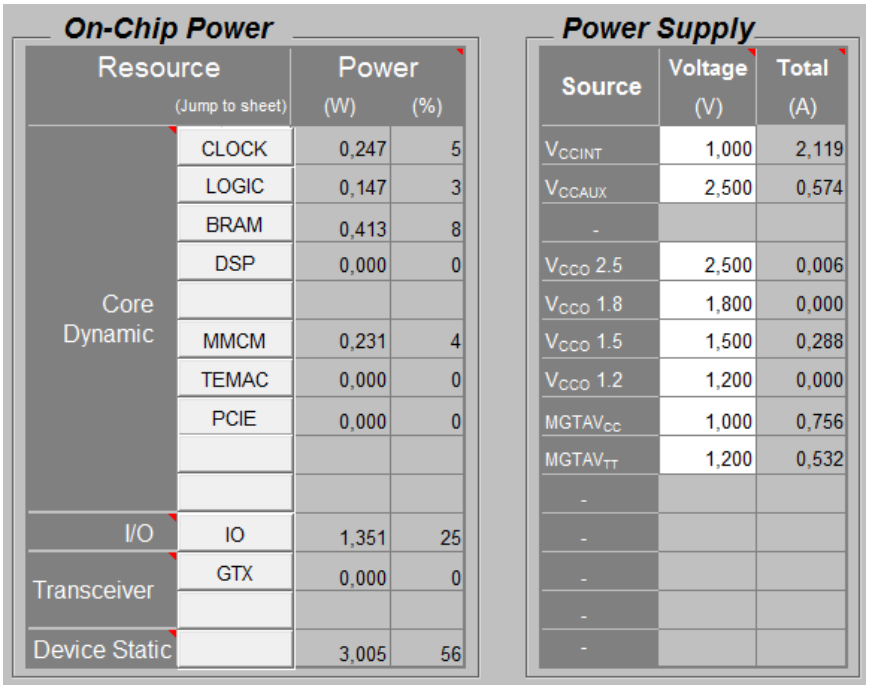
Tasa de Habilitación	
Bloques de RAM	25 %
Bloques de RAM Para Escritura	50 %
Bidi Output	50 %
Output	100 %

Tabla. 9.24. Tasa de habilitación para arquitectura NoC Hermes en consumo total de potencia

9.4.1.1. Resultados Obtenidos en Estimación de Consumo Total de Potencia para una Arquitectura Interconectada por NoC Hermes

La Figura. 9.23 representa un resumen del consumo de potencia de cada tipo de elemento que conforma el FPGA Virtex-6, así como el consumo de potencia de

cada una de las fuentes de alimentación que componen la arquitectura interconectada por NoC Hermes. Para mayor detalle revisar ANEXO 11.



On-Chip Power

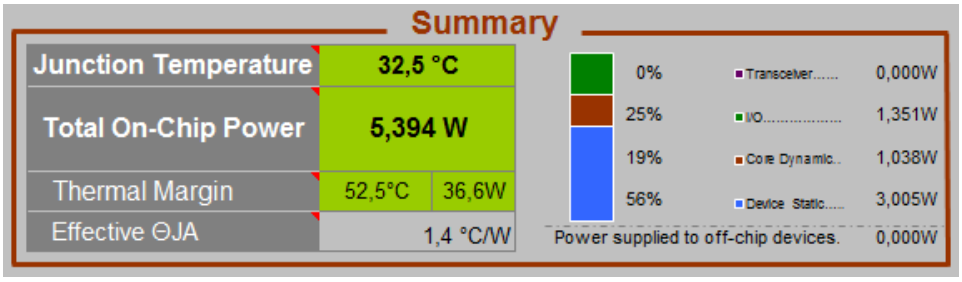
Resource	(Jump to sheet)	Power (W)	(%)
Core Dynamic	CLOCK	0,247	5
	LOGIC	0,147	3
	BRAM	0,413	8
	DSP	0,000	0
	MMCM	0,231	4
	TEMAC	0,000	0
	PCIE	0,000	0
I/O	IO	1,351	25
Transceiver	GTX	0,000	0
Device Static		3,005	56

Power Supply

Source	Voltage (V)	Total (A)
V_{CCINT}	1,000	2,119
V_{CCAUX}	2,500	0,574
-		
V_{CCO} 2.5	2,500	0,006
V_{CCO} 1.8	1,800	0,000
V_{CCO} 1.5	1,500	0,288
V_{CCO} 1.2	1,200	0,000
$MGTAV_{CC}$	1,000	0,756
$MGTAV_{TT}$	1,200	0,532
-		
-		
-		
-		
-		

Figura. 9.23. Consumo de potencia estática y fuentes de alimentación para NoC Hermes

La Figura. 9.24 representa un cuadro de resumen de la estimación de consumo total de potencia para una arquitectura interconectada por NoC Hermes implementada sobre el FPGA Virtex-6.



Summary

Junction Temperature	32,5 °C	
Total On-Chip Power	5,394 W	
Thermal Margin	52,5°C	36,6W
Effective ΘJA	1,4 °C/W	

0%	Transceiver.....	0,000W
25%	I/O.....	1,351W
19%	Core Dynamic..	1,038W
56%	Device Static.....	3,005W
Power supplied to off-chip devices.		0,000W

Figura. 9.24. Cuadro Resumen de consumo total de potencia en NoC Hermes

La Figura. 9.25 representa la cantidad de recursos lógicos utilizados en la implementación de una arquitectura interconectada por NoC Hermes.

Utilization		
FFs	15.256	5%
LUTs	17.250	11%
Combinatorial	15.076	10%
Shift Registers	706	3%
Distributed RAMs	1.468	

Figura. 9.25. Recursos lógico utilizados en arquitectura NoC Hermes

9.4.2. Estimación de Consumo Total de Potencia para Arquitecturas Interconectadas por Buses FSL

La estimación de consumo total de potencia para una arquitectura interconectada por Buses FSL implementada sobre la FPGA ML605 Virtex-6 mediante la herramienta XPE se la realizará con tasas de conmutación y tasas de habilitación fijas representadas en Tabla. 9.25 y Tabla. 9.26.

Tasa de Conmutación	
Elementos Lógicos	12.5 %
Bloques de RAM	50 %
DSP	12.5 %
I/O	12.5 %

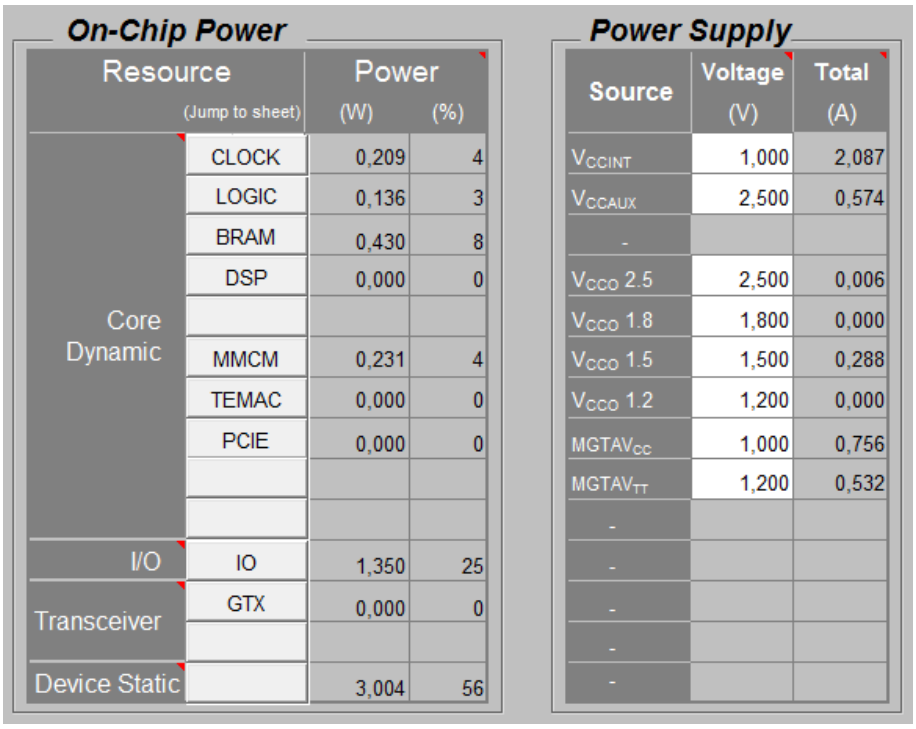
Tabla. 9.25. Tasa de conmutación para arquitectura de buses FSL en consumo total de potencia

Tasa de Habilitación	
Bloques de RAM	25 %
Bloques de RAM Para Escritura	50 %
Bidi Output	50 %
Output	100 %

Tabla. 9.26. Tasa de habilitación para arquitectura de buses FSL en consumo total de potencia

9.4.2.1. Resultados Obtenidos en Estimación de Consumo Total de Potencia para una Arquitectura Interconectada por Buses FSL

La Figura. 9.26 representa un resumen del consumo de potencia de cada tipo de elemento que conforma el FPGA Virtex-6, así como el consumo de potencia de cada una de las fuentes de alimentación que componen la arquitectura interconectada por buses FSL. Para mayor detalle revisar ANEXO 12.



On-Chip Power

Resource		Power (W)	Power (%)
Core Dynamic	CLOCK	0,209	4
	LOGIC	0,136	3
	BRAM	0,430	8
	DSP	0,000	0
	MMCM	0,231	4
	TEMAC	0,000	0
	PCIE	0,000	0
I/O	IO	1,350	25
Transceiver	GTX	0,000	0
Device Static		3,004	56

Power Supply

Source	Voltage (V)	Total (A)
V_{CCINT}	1,000	2,087
V_{CCAUX}	2,500	0,574
-		
V_{CCO} 2.5	2,500	0,006
V_{CCO} 1.8	1,800	0,000
V_{CCO} 1.5	1,500	0,288
V_{CCO} 1.2	1,200	0,000
$MGTAV_{CC}$	1,000	0,756
$MGTAV_{TT}$	1,200	0,532
-		
-		
-		
-		
-		

Figura. 9.26. Consumo de potencia estática y fuentes de alimentación para buses FSL

La Figura. 9.27 representa un cuadro de resumen de la estimación de consumo total de potencia para una arquitectura interconectada por Buses FSL implementada sobre el FPGA Virtex-6.

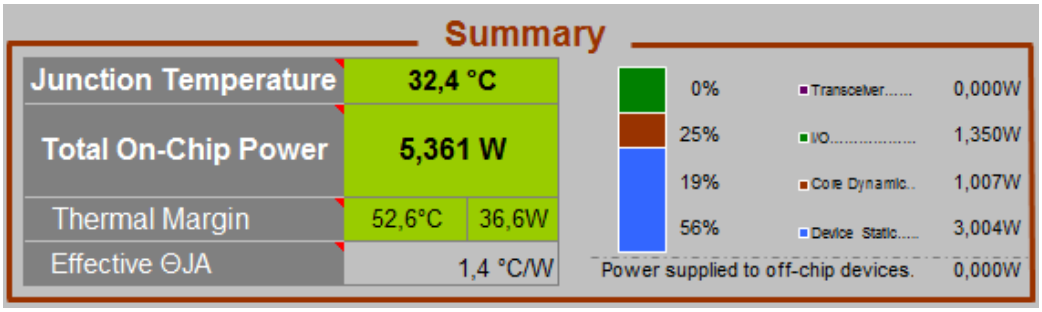


Figura. 9.27. Cuadro Resumen de consumo total de potencia en buses FSL

La Figura. 9.28 representa la cantidad de recursos lógicos utilizados en la implementación de una arquitectura interconectada por buses FSL.

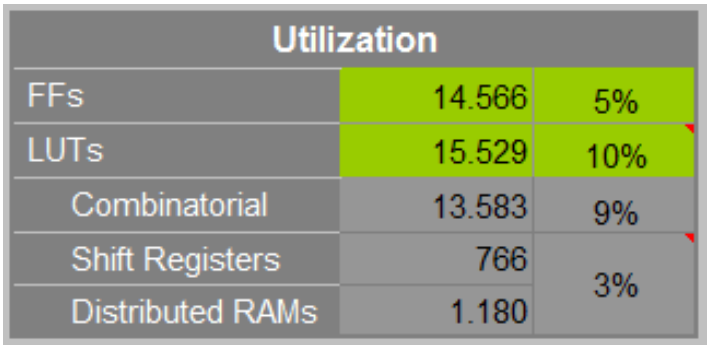


Figura. 9.28. Recursos lógico utilizados en arquitectura de buses FSL

9.5. ANÁLISIS DE RESULTADOS EN ESTIMACIÓN DE POTENCIA

9.5.2. Análisis de Resultados en Estimación de Potencia Dinámica para las Arquitecturas Interconectadas por NoC Hermes y por Buses FSL

En la Tabla. 9.27 se muestra la comparación de valores de estimación de potencia para una arquitectura interconectada por NoC Hermes y una arquitectura interconectada por Buses FSL, tomando como parámetro de variación la tasa de conmutación.

Tasa de Conmutación			Estimación de Potencia (W)		
Tasa I/Os	**Tasa Flip Flops**	**Escenarios de Prueba**	**NoC Hermes**	**Buses FSL**	**Diferencia**
12.5 %	12.5 %	Escenario 1	5.163	5.152	0.011
12.5 %	25 %	Escenario 2	5.208	5.194	0.014
25 %	12.5 %	Escenario 3	5.219	5.209	0.010
25 %	25 %	Escenario 4	5.255	5.250	0.005
50 %	50 %	Escenario 5	5.460	5.446	0.014
75 %	75 %	Escenario 6	5.649	5.635	0.011

Tabla. 9.27. Comparación de estimación de potencia entre FSL y NoC Hermes

En la tabla se aprecia diferenciales de estimación de potencia que resultan de la comparación entre las arquitecturas interconectadas por NoC Hermes y por buses FSL. Estos diferenciales no se mantienen constantes en ninguno de los escenarios de prueba porque no existe la misma cantidad de elementos en cada una las arquitecturas lo que hace que cada escenario entregue un resultado de estimación diferente.

En la Figura. 9.29 se compara el consumo total de potencia para cada una de las arquitecturas implementadas. Al realizar un análisis comparativo de estimación de potencia dinámica consumida por los elementos del FPGA, se determina que en la infraestructura conectada por NoC Hermes, el consumo de potencia es mayor a la infraestructura conectada por buses FSL en todos los escenarios de estimación. También se puede observar un incremento de potencia directamente proporcional al incremento de la tasa conmutación en cada uno de los escenarios de prueba tanto para la arquitectura interconectada por NoC Hermes como para la arquitectura interconectada por Buses FSL.

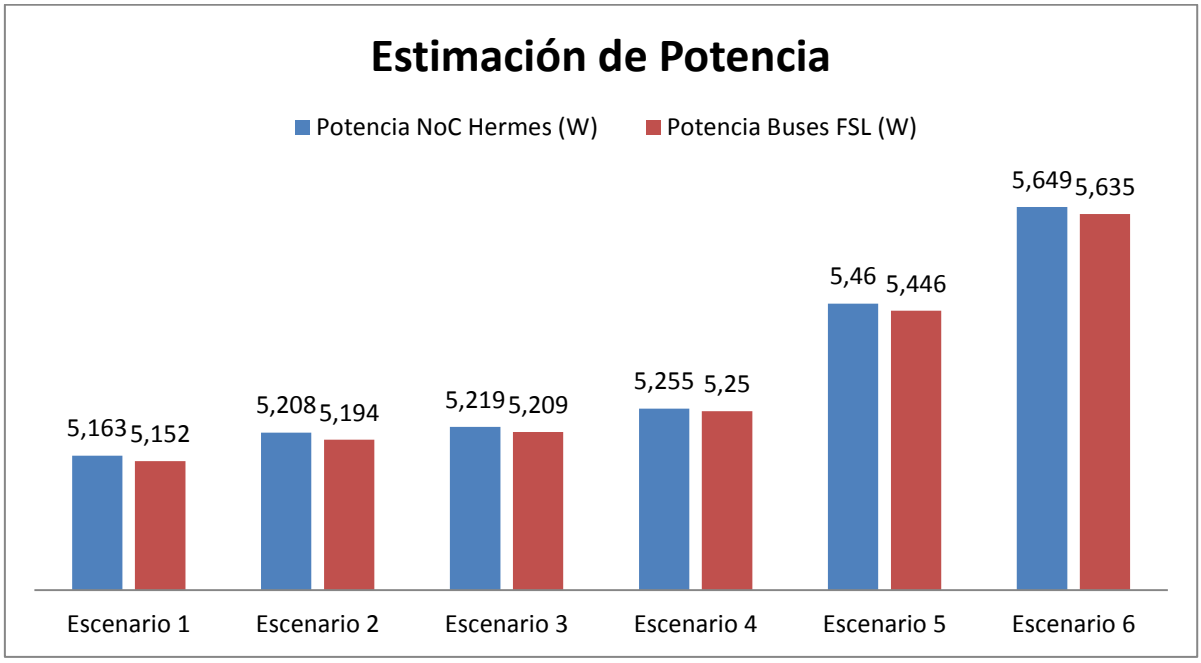


Figura. 9.29. Análisis comparativo de estimación de potencia en función de los escenarios de estimación

En conclusión el incremento en el consumo de potencia en la arquitectura interconectada por NoC Hermes se debe a la cantidad de elementos lógicos presentes en los Switches de Hermes y en las Interfaces de Red diseñadas para las interconexiones con los distintos procesadores. Estos elementos se obviaron en la arquitectura de buses y por esta razón, tanto instanciar, como configurar dichos elementos, ya conlleva un incremento de consumo de potencia.

9.5.3. Análisis de Resultados en Estimación de Consumo Total de Potencia Estática para las Arquitecturas Interconectadas por NoC Hermes y por Buses FSL

En la Figura. 9.30 se puede observar que a medida que se incrementa el consumo de potencia estática también se incrementa la temperatura de juntura. Es así que la arquitectura interconectada por NoC Hermes consume 33mW más que la arquitectura interconectada por buses FSL. En consecuencia la

temperatura de juntura de la arquitectura interconectada por NoC Hermes es 0.1 °C más que la arquitectura interconectada por buses FSL. Este aumento en potencia y temperatura sobre la arquitectura interconectada por NoC Hermes se debe a la cantidad de recursos utilizados en el FPGA tal como se puede observar en la Figura. 9.31.

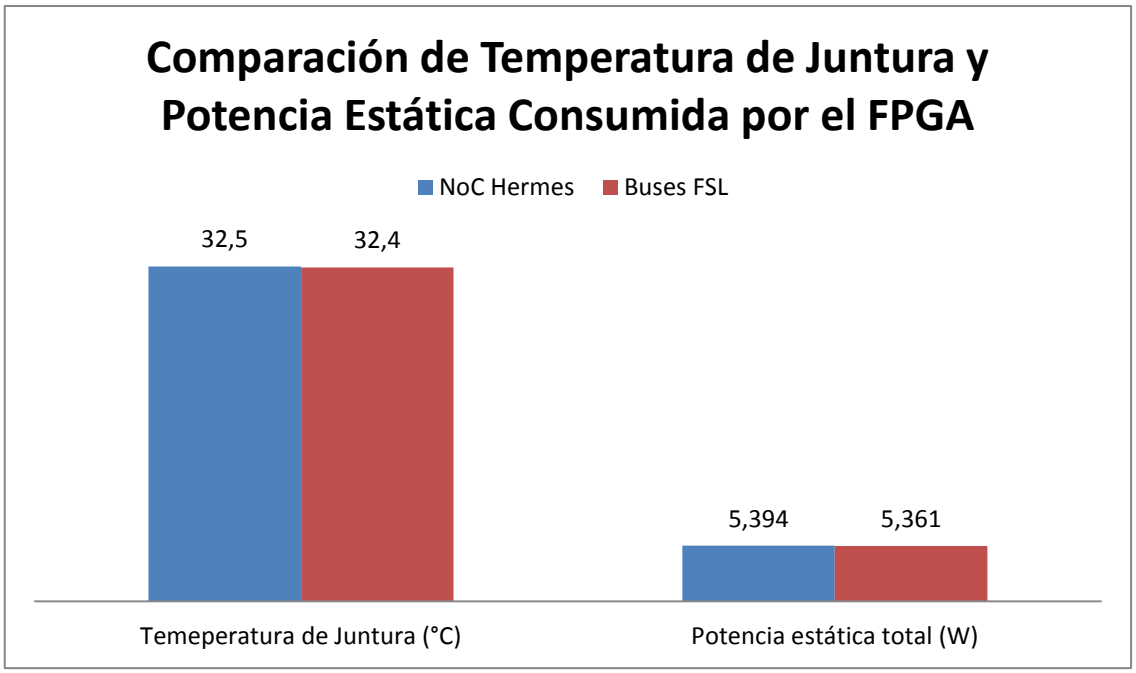


Figura. 9.30. Comparación de temperatura de juntura y de potencia estática total

En la Figura. 9.31 se observa la comparación de recursos utilizados en el FPGA para la arquitectura interconectada por NOC Hermes y para la arquitectura interconectada por buses FSL.

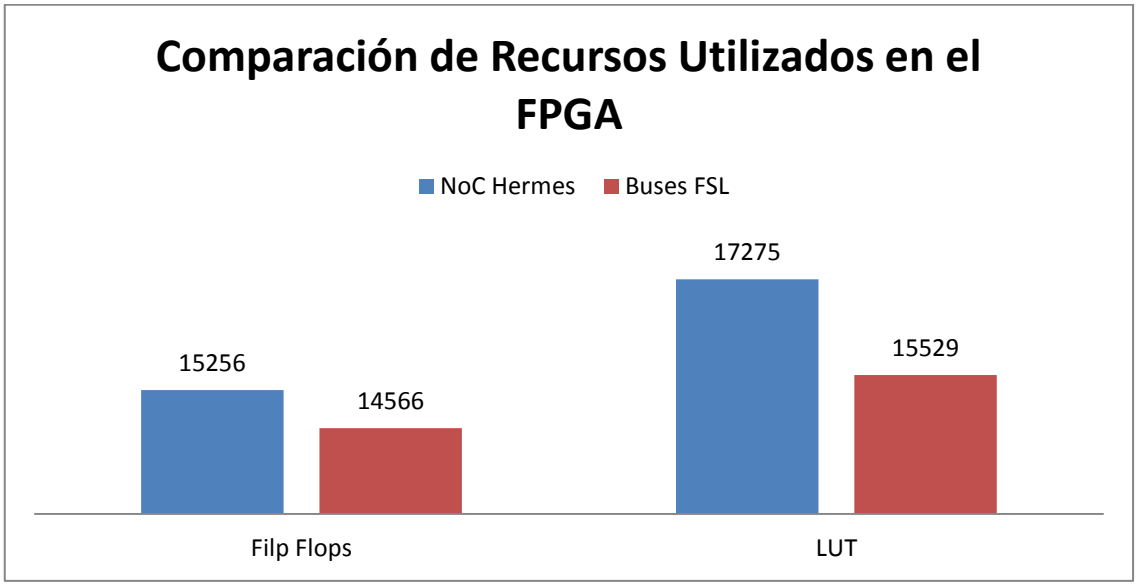


Figura. 9.31. Comparación de los recursos utilizados en el FPGA ML605 Virtex-6

En base a la herramienta Xilinx Power Estimator descrita en el **Capítulo 3** se puede realizar un análisis comparativo de consumo de potencia estática para cada una de las arquitecturas, en donde se evalúan los valores de estabilización con respecto a la fuente de alimentación Vccint.

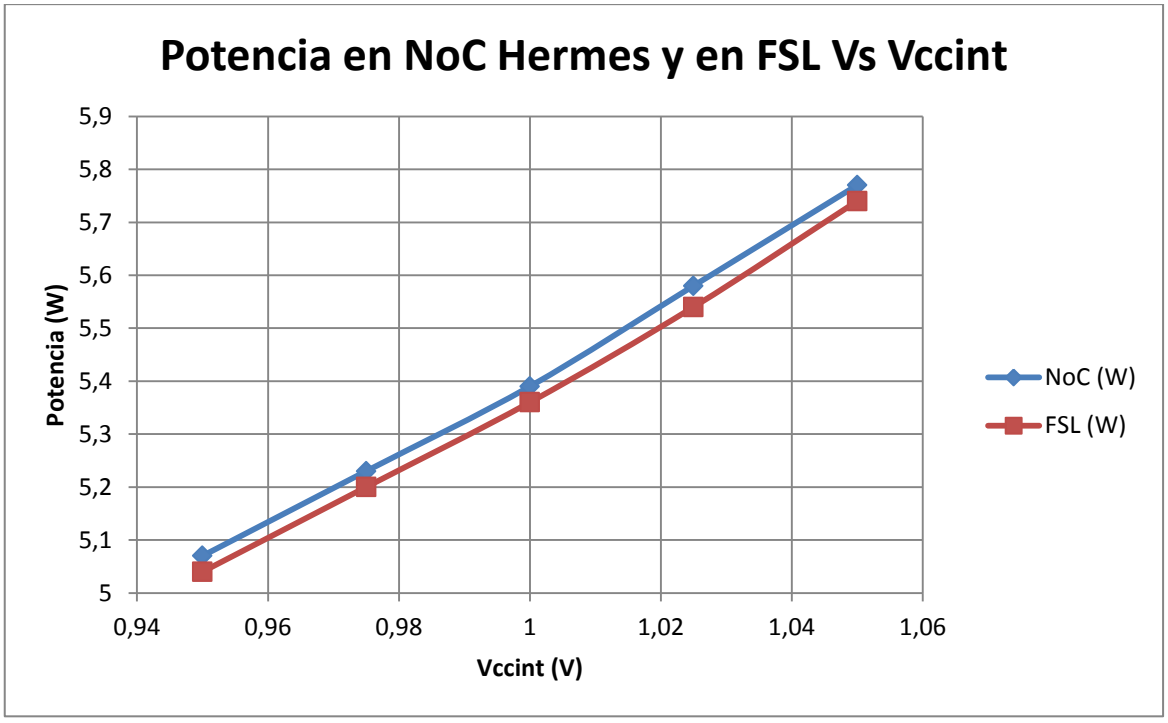


Figura. 9.32. Potencia en NoC Hermes y en FSL Vs. Vccint

En la Figura. 9.32 se puede apreciar que la potencia es mayor en una arquitectura interconectada por NoC Hermes con respecto una arquitectura interconectada por buses FSL en 32mW aproximadamente. Esto se debe a que ambas arquitecturas están conectadas en la misma topología de malla, siendo NoC Hermes la arquitectura de comunicación que más elementos emplea para su interconexión a nivel intrachip.

Finalmente, la Figura. 9.33 muestra un análisis comparativo de Temperaturas de Juntura para la arquitectura conectada por NoC y para la arquitectura conectada por buses FSL, tomando en cuenta la estimación de potencia para cada uno de los diseños.

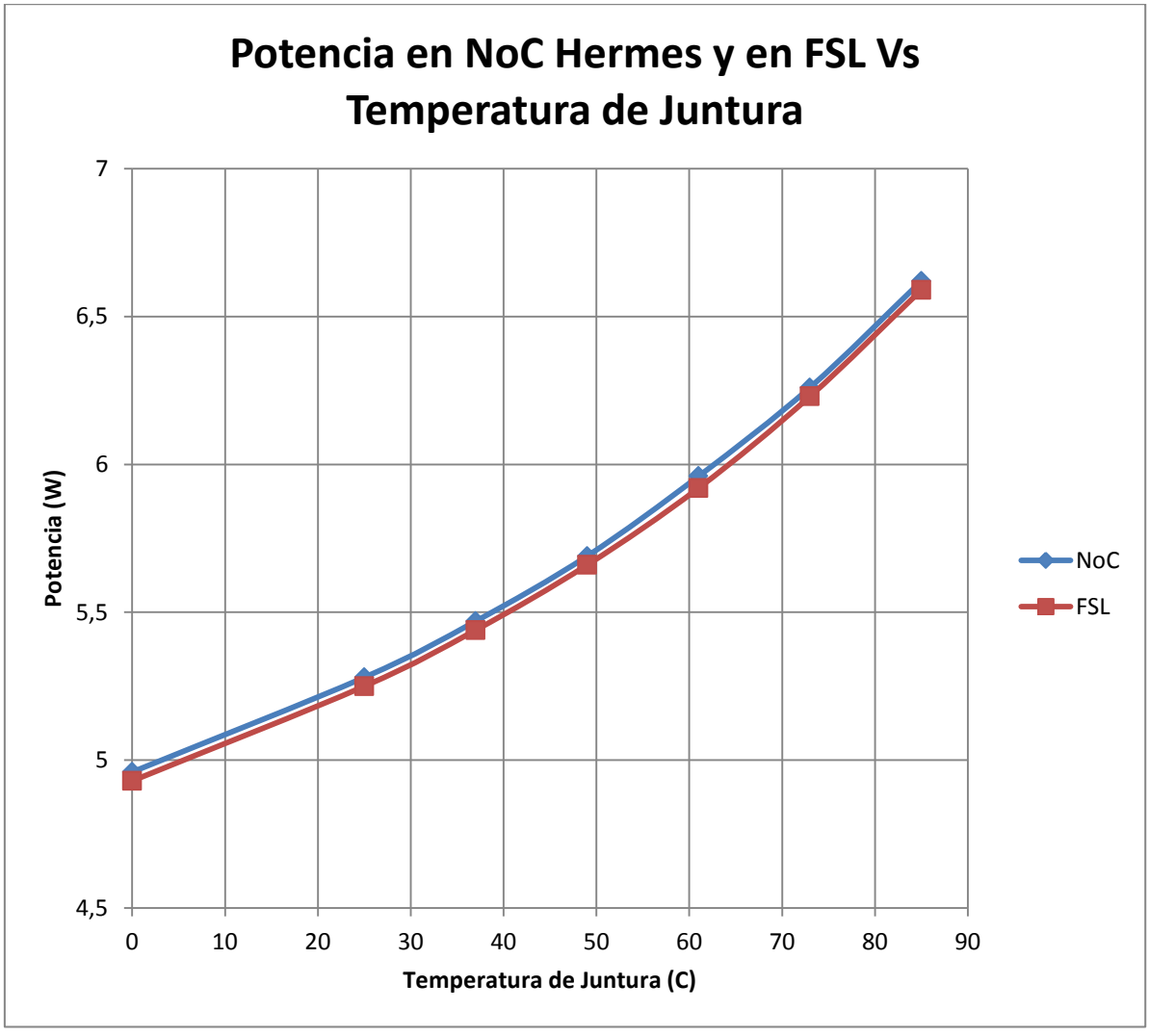


Figura. 9.33. Potencia en NoC Hermes y en FSL Vs. Temperatura de Juntura

Como se mencionó anteriormente ambas arquitecturas utilizan una topología de malla en su infraestructura de comunicación. Por lo tanto en la Figura. 9.33 se puede apreciar que la disipación de potencia por temperatura de juntura para una arquitectura interconectada por NoC Hermes es mayor que para una arquitectura interconectada por buses FSL en 31mW aproximadamente. Sin embargo es posible que conforme se realicen trabajos futuros sobre topologías en donde se incrementen más dispositivos con buses FSL se vea un aumento en la disipación de potencia por temperatura de juntura.

CAPÍTULO 10

CONCLUSIONES Y RECOMENDACIONES

10.1. CONCLUSIONES

Se implementó dos arquitecturas MPSoC de 4 procesadores MicroBlaze sobre la tarjeta de desarrollo ML605 Virtex-6. La primera tiene una arquitectura de comunicación por medio de Buses FSL y la otra está comunicada por medio de NoC. Estas arquitecturas fueron comparadas con el fin de demostrar que NoC es más eficiente en latencia que buses FSL en la transmisión de paquetes para una aplicación que demande alto tráfico de datos y mínima pérdida de información.

Después de realizar una caracterización del estado del arte de NoC se encontró que existe una amplia variedad de modelos de switches interconectados en diferentes topologías a nivel intrachip. Gracias al aporte del Área de Sistemas Digitales de la Escuela Politécnica del Ejército (ESPE) se planteó la utilización de NoC Hermes desarrollada por la Facultad de Informática de la Pontificia Universidade do Rio Grande do Sul (PUCRS, Porto Alegre, Brasil) y la implementación sobre la FPGA mediante código VHDL a través de la herramienta ATLAS desarrollada por GAPH (Hardware Design Support Group-PUCRS). Esta herramienta configura únicamente los switches de NoC Hermes en una topología 2D-Mesh en código abierto VHDL.

Con la finalidad de crear una arquitectura multiprocesada basada en MicroBlaze que emplea NoC Hermes para su comunicación fue necesario diseñar y desarrollar una interfaz de red (NI). El IP-Core resultante es de código abierto VHDL, puede ser modificado y es exclusivamente para comunicación por FSL. Este IP-Core fue desarrollado en la ESPE como contribución al presente proyecto de grado.

A pesar de que se logró implementar las dos arquitecturas MPSoC interconectadas por buses e interconectadas por NoC, no se pudo medir físicamente la latencia y el consumo de energía debido a que la plataforma de desarrollo no permite realizarlas. Por este motivo se recurrió a la estimación de estos parámetros mediante el uso de simuladores. Las simulaciones se realizaron utilizando generadores de tráfico randómico que carecen de un patrón de comportamiento que emule las características de una arquitectura multiprocesada. Por lo tanto, lo ideal para trabajos futuros es que se desarrollen generadores de tráfico que permitan simular de forma paralela diversos números de nodos dentro de una topología NoC Hermes superior a una 2D-Mesh de 2x2. Por otra parte se deberán crear generadores de tráfico que permita evaluar a NoC Hermes con patrones de tráfico de aplicaciones reales.

Mediante la utilización de generadores de tráfico se realizaron simulaciones a dos y tres nodos encontrando como resultado que la arquitectura NoC Hermes aumenta su latencia a medida que se incrementa el número de nodos involucrados en la simulación. Esto se debe principalmente al tiempo de dispersión que le toma a cada switch para establecer una ruta de comunicación mediante el algoritmo XY.

En las pruebas simuladas entre NoC Hermes a dos nodos y buses FSL, en la transmisión de paquetes de punto a punto se demostró que el bus FSL es más

rápido que la arquitectura NoC Hermes, debido a que los tiempos de transmisión son más prolongados en NoC Hermes que en FSL. En conclusión una arquitectura NoC Hermes de 2x2 en topología 2D-Mesh no es suficiente para ser evaluada. Por lo cual se requiere topologías más grandes en donde las latencias se incrementen a tal punto que se pueda demostrar la potencial mejora del desempeño al utilizar NoC Hermes frente a buses FSL.

En la arquitectura de comunicación NoC Hermes sin NI, conectada con generadores de tráfico, la latencia es de 23µs para un paquete de 5 flits. Mientras que para una arquitectura de comunicación NoC Hermes con NI, la latencia es de 25µs para el mismo tamaño de paquete. En consecuencia los NI diseñados para comunicar a los procesadores con NoC Hermes presentan una latencia de 2µs adicionales.

Con el fin de evaluar las arquitecturas multiprocesadas interconectadas por buses FSL y por NoC Hermes implementadas sobre el FPGA se desarrolló una aplicación de esteganografía en la que existe multiprocesamiento de dos procesadores trabajando simultáneamente. Esta aplicación fue útil para evaluar ambas arquitecturas dando como resultado que la implementación mediante buses FSL ocupó el 10% de los recursos del FPGA y NoC Hermes el 11%. Mientras que para las dos arquitecturas se utilizó el 100% de los recursos de memoria disponibles en el FPGA.

Para medir latencia se necesita gran cantidad de tráfico de información pero por las limitaciones de memoria en la plataforma de desarrollo se utilizó imágenes de 320x240 pixeles y con mensajes de 300 caracteres. Las limitantes en dimensión de imágenes y en tamaño de mensajes están condicionadas por las capacidades de memoria del FPGA ML605 Virtex-6 sobre el cual fueron implementados.

Para comparar el consumo de energía de cada una de las arquitecturas multiprocesadas se realizó una estimación de potencia variando las tasas de conmutación tanto para Flip Flops como para elementos I/O. El proceso de estimación da como resultado que el consumo de energía en la arquitectura interconectada por NoC Hermes en topología 2D-Mesh es 33mW mayor que el consumo de energía para la arquitectura interconectada por buses FSL. Este incremento se produce debido a que la arquitectura interconectada por NoC Hermes posee más elementos que son necesarios para su funcionamiento, los mismos que no están presentes en la arquitectura interconectada por buses FSL. Estos elementos corresponden a los Switches de Hermes y las Interfaces de Red que permiten la comunicación con los procesadores MicroBlaze.

10.2. RECOMENDACIONES

El uso de las librerías facilita el manejo de los GPIO que existen en el MPSoC, por lo tanto es importante leer y entender cómo se deben instanciar al momento de realizar un nuevo proyecto.

Se debe prestar especial atención al uso de los IP-Cores xps_sysace y xps_tft, por lo que se recomienda entender los tutoriales desarrollados para el manejo, configuración y en especial su puesta en marcha cuando se los conecta a un solo procesador o en una arquitectura de multiprocesamiento.

Para el caso del IP-Core xps_tft, se debe asegurar que tanto el reloj del módulo de reinicio (Slowest_sync_clk) y el reloj del xps_tft (sys_tft_clk) estén conectados a un reloj de frecuencia de 25MHz, caso contrario se obtienen errores en la generación del hardware o si el mismo se genera, el problema se

presentaría al momento de ejecutar el programa en el software del SDK. Para más información consultar los tutoriales.

Dentro de SDK es recomendable crear un grupo de descarga para los programas que se descargarán en el MPSoC a través del RunConfiguration. Esto permite que dichos programas se graben en el FPGA ML605 Virtex-6 de forma más sencilla.

El paquete mínimo que se puede transmitir a través de la red creada con el Switch Hermes es de tres flits, que son: cabecera, tamaño y dato, por lo tanto hay que tener cuidado al momento de armar el paquete desde el software a fin de evitar problemas con el hardware de NoC Hermes.

Para la creación de archivos de imagen en formato ppm se utilizó el software Corel PaintShop Pro X4, sin embargo los datos presentes en el formato modificado son incomprensibles para el programa creado en el SDK, debido a esto se creó otro programa utilizando el Borland C++ 5.02. La razón principal de usar dicho software es que el SDK manejar ficheros de forma limitada. Al utilizar Borland C++ 5.02 se manipuló el fichero a fin de ordenar la información contenida en la imagen y hacerla fácil de interpretar para el programa hecho en el SDK.

Es recomendable crear generadores de tráfico para pruebas en tiempo real que permitan evaluar a la arquitectura de NoC Hermes en diversos escenarios de tráfico. Estos escenarios de pruebas deberán realizarse con aplicaciones en paralelo que permitan demostrar que NoC Hermes es superior a los buses FSL.

CAPÍTULO 11

BIBLIOGRAFÍA

(n.d.). *Procesamiento de Imágenes Digitales*. Universidad de Málaga, Dpto. Lenguajes y CC. Computación E.T.S.I. Telecomunicación.

(1999, May. 13). *AMBA Specification*. Revisión 2.0, ARM IHI 0011A.

(2008, Apr.-Jul). *Procesamiento de Imáganes*. Universidad Simón Bolívar, Departamento de Comunicación y TI.

(2010). *Wishbone System-On-Chip (SoC) Interconnection Architecture for Portable IP-Cores*. Wishbone B4, OpenCores.

(2010). *Interfaz WISHBONE*. Revisión 1.6, OpenCores.

Agarwal, A., Iskander, C., & Shankar, R. (2009). Survey of Network on Chip (NoC) Architectures & Contributions. *Vol. 3, Issue 1*. Journal of Engineering, Computing and Architecture.

Al-Hadithi, B., & Muro, J. (2004). *Nuevas Tendencias en el diseño electrónico digital: Codiseño Hardware/Software, Volumen II*. Tecnología y Desarrollo, Revista de Ciencia, Tecnología y Medio Ambiente, Universidad Alfonso X El Sabio.

Anderson, J. H. (n.d.). *Power Estimation Techniques for FPGAs*. IEEE, and Farid N. Najm, Fellow, IEEE.

Andriahantenaina, A., & Charlery, H. (2003, Mar. 3). *SPIN: a Scalable, Packet Switched, On-chip Micro –network*. In: Design Automation and Test in Europe Conference and Exhibition.

Andriahantenaina, A., & Greiner, A. (2003, Mar. 3). *Micro-network for SoC: Implementation of a 32-port SPIN network*. In: Design Automation and Test in Europe Conference and Exhibition.

Antoni, R. (2007, Mar.). Procedimiento de diseño de circuitos digitales mediante FPGAs. Proyecto de Fin de Carrera, Universidad de Lleida, Escuela Politécnica Superior.

Asokan, V. (2007, Nov.). *Designing Multiprocessor Systems in Platform Studio*. Xilinx, White Paper, WP262.

Atienza, A., Angiolini, F., Murali, S., Pullini, A., Benini, A., & De Micheli, G. (2007, Nov.). *Network-on-Chip design and synthesis outlook*. INTEGRATION, the VLSI journal 41 (2008) 340-359.

Baena, A. (2010, Feb.). *Diseño sobre FPGA de una Unidad Aritmética Decimal*. Departamento de Ingeniería Electrónica y Automática, Universidad Rovira|Virgili.

Benini, L., & De Micheli, G. (n.d.). *Networks on chip: A new paradigm for component-based MPSoC desing*. DEIS Universidad de Bolognia, CSL Stanford University.

Bijlsma, B. (2005, Sep.). Asynchronous Network-on-Chip Architecture Performance Analysis. MSC Thesis, Microelectronics, Department of Electrical Engineering, Delft University of Technology.

Cadena, J., & Mollocana, G. (2012). Diseño de Hardware y Software de Systems-On-Chip Empleando Tecnología Xilinx EDK. Sangolquí, Rumiñahui, Ecuador: Proyecto de Grado para la Obtención de Titulo de Ingeniería, Escuela Politécnica del Ejército.

Carpio, F. (1997, Oct.). *Tutorial Verilog*. Ingeniería Informática, Universidad de Valencia.

Chiodo, M., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., & Sangiovanni, A. (2003, Jun.). *A Formal Specification Model for Hardware/Software Codesing*.

Chiodo, M., Giusto, P., Jurecska, A., Hsieh, H., Sangiovanni, A., & Lavagno, L. (1994, Ag.). *Hardware-Software Codesign of Embedded Systems*.

Dall'Osso, M., Biccari, G., Giovannini, L., Bertozzi, D., & Benini, L. (n.d.). *Xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs*. Italy: University of Bologna, DEIS, Viale Risorgimento.

Dally, W., & Towles, R. (2001, Jun.). *Route packets, not wires: on-chip interconnection networks*. In: 38th Design Automation Conference (DAC'01).

de la Fuente, D. (2007, Sep.). *Sistema de Comunicación HW-SW basado en objetos distribuidos*. Proyecto de fin de carrera, Universidad de Castilla-La Mancha, Escuela Superior de Informática.

Death, M. (n.d.). Introducción a la Esteganografía. Death Master, 2004 (GFDL).

Edith, B., Christian, B., Fabien, C., Yves, D., Jean, D., Didier, L., . . . Pascal, V. (n.d.). *FAUST, an Asynchronous Network-on-Chip based Architecture for Telecom Applications CEA-LETI*. France: 17, rue des Martyrs, 38054 Grenoble Cedex 9.

Ernst, R. (2003). *MPSOC Architecture Modeling*. TU Braunschweig.

Evain, S., Diguet, J., & Houzet, D. (n.d.). *uSpider: a CAD Tool for Efficient NoC Desing*. France: LESTER, IETR-INSA.

Forsell, M. (2002, Sep.-Oct.). *A Scalable High-Performance Computing Solution for Networks on Chips*. IEEE Micro.

Goossens, K., Dielissen, J., & Radulescu, A. (n.d.). *AEthereal Network on Chip: Concepts, Architectures and Implementations*. Philips Research Laboratories.

Guerrier, P., & Greiner, A. (2000, Mar.). *A generic architecture for on-chip packet-switched interconnections*. In: Design Automation and Test in Europe (DATE'00).

Guzmán, H. (2006). Investigación sobre herramientas de Codiseño Hardware-Software. Proyecto de Fin de Carrera, Escuela Superior de Ingenieros, Universidad de Sevilla.

Hemani, A., Jantsch, A., Kumar, S., Postula, A., Öberg, J., Millberg, M., & Lindqvist, D. (2000). *Network on chip: An architecture for billon transistor era*.

Hessabi, S. (n.d.). *SoC Design*. Department of Computer Engineering, Sharif University of Technology.

Huerta, P. (2009, May.). Sistemas de Multiprocesamiento Simétrico sobre FPGA. Tesis Doctoral, Departamento de Arquitectura y Tecnología de Computadoras, Ciencias de la Computación e Inteligencia Artificial, Universidad Rey Juan Carlos.

Jian, L., Swaminathan, S., & Tessier, R. (2000, Oct.). *aSOC: A Scalable, Single -Chip communications Architecture*. In: IEEE International Conference on Parallel Architectures and Compilation Techniques.

Juan, P. (n.d.). *Proyecto Encriptación de texto sobre imágenes BMP mediante software*.

Karim, F., Nguyen, A., & Dey, S. (2000, Sep-Oct). *An interconnect architecture for network systems on chips*. IEEE Micro v. 22(5).

Karim, F., Nguyen, A., Dey, S., & Rao, R. (2001, Jun.). *On-chip communication architecture for OC-768 network processors*. In: 38th Design Automation Conference (DAC'01).

Kumar, S. (2002, Apr.). *A Network on Chip Architecture and Design Methodology*. In: IEEE Computer Society Annual Symposium on VLSI. (ISVLSI'02).

Leroy, A. (2006-2007). *Optimizing the on-chip communication architecture of low power Systems-on-Chip in Deep Sub-Micron technology*. Université Libre de Bruxelles.

M., L. A. (2009). *Introducción al Lenguaje de Descripción de Hardware*. Universidad Nacional Experimental del Táchira, Departamento de Ingeniería. Electrónica, San Cristóbal, 2009.

Marescaux, T. (2007, Sep.). *Mapping and Management of Communication Services on MP-SoC Platforms*. CIP-DATA Library Technische Universiteit Eindhoven.

Marescaux, T., Bartic, A., Verkest, D., Vernalde, S., & Lauwereins, R. (2002, Sep.). *Interconnection Networks Enable Fine -Grain Dynamic Multi-Tasking on FPGAs*. In: Field-Programmable Logic and Applications (FPL'02).

Martin, G., & Chang, H. (1999). *Surviving the Soc Revolution – A Guide to Platform – Based Design*. Estados Unidos: Kluwer Academic Publisher.

Martínez, R. (n.d.). *Introducción a los IP-CORES*. Departamento de Ingeniería de Sistemas, Instituto de Microelectrónica de Barcelona.

Milica, M., & Stojcev, M. (2006, Dec.). *An Overview of On-Chip Buses, Vol. 19*. Facta Universitatis (NIS).

Miro-Panades, I., Clermidy, F., Vivet, P., & Greiner, A. (n.d.). *Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture*. Paris, France: The University of Pierre et Marie Curie, 75252.

Mirza-Aghatabar, M., Koohi, S., Hessabi, S., & Pedram, M. (n.d.). *An Emperical Investigation of Mesh and Torus NoC Topologies under Different Routing Algorithms and Traffic Models*. Sharif University of Technology, Tehran, Iran, University of Southern California, CA, USA.

Mora, A. (2008). Estudio de Arquitecturas VLSI de la Etapa de Predicción de la Compensación de Movimiento, para Compresión de Imágenes y Video con Algoritmos Full-Search. Aplicación Estándar H.264/AVC. España: Tesis Doctoral, Universidad Politécnica de Valencia.

Moraes, F., Vilar, N., Viera, A., Möller, L., & Copello, L. (2003, Oct.). *HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*.

Nollet, V. (2008, Ap.). *Run-Time management for future MPSoC Platforms*. Belgic: PROEFSCHRIFT.

Nuñez. (n.d.). *Tutorial Verilog*. Universidad de las Palmas de Gran Canaria.

Observatorio de la Seguridad de la Información. (n.d.). Esteganografía, el Arte de Ocultar Información. Cuaderno de notas del Observatorio, Instituto Nacional de Tecnologías de la Comunicación.

Ou, J., & Prasanna, V. (n.d.). *A Methodology for Energy Efficient Application Synthesis Using Platform FPGAs*. California, Los Ángeles, USA: Department of Electrical Engineering, University of Southern.

Pantalopoulos, S., & Brokalakis, A. (2012, May. 7). *Hihg Level Architecture of Parallel System*. Information and Communication Technologies (ICT) Programme.

Pasricha, S., & Dutt, N. (2008). *On-Chip Communication Architectures*. ICS 295.

Pedram, A., Zainalabedin, N., & Lombardi, F. (n.d.). *Simulating Faults of Combitional IP-Core-based SOC's in a PLI Enviroment*. Electrical and Computer Engineering Department, Northeasterm University.

Peña, J. (2008, Jun.). Diseño de módulos para el manejo de Puertos, Temporización e Interrupciones para el Núcleo KCPSM3 e Implementación en el FPGA XC3S500 de Xilinx. Proyecto de Fin de Carrera, Universidad Tecnología de la Mixteca.

Petrini, M., & Vanneschi, M. (1997). *k-ary n-trees: High Performance Networks for Massively Parallel Architectures*.

Rajesh, K. (2008, Sep.). On-chip memory architecture exploration of embedded System on chip. Tesis Doctoral, Facultad de Ingeniería, Instituto de Ciencias de la India, Supercomputer Education and Research.

Rijpkema, E., Goossens, K., & Radulescu, A. (2003, Mar.). *Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip*. In: Design, Automation and Test in Europe (DATE'03).

Rijpkema, E., Goossens, K., & Wielage, P. (2001, Nov.). *A Router Architecture for Networks on Silicon*. In: 2nd Workshop on Embedded System (PROGRESS'2001).

Rivero, J. (2005, Oct.). Desarrollo de un Elemento de Conmutación para una Arquitectura Network-on-Chip (NoC) con Garantías de QoS. Proyecto de Fin de Carrera, Universidad de las Palmas de Gran Canaria.

Sgroi, M., Sheets, M., Mihal, A., Keutzer, K., Malik, S., Rabaey, S., & Sangiovanni-Vicentelli, A. (2001, Jun.). *Addressing the System-on-Chip Interconnect Woes Through Communication-Based Design*. In: 38th Design Automation Conference (DAC'01).

Silva, L. (2010). Uso de Verilog. Sistemas Digitales, Apéndice 5.

Tan, C. C. (2010, Jun. 14). *Module: Multiprocessor and Multi-Port Memory Controller*. EDK 10.1.03, University of Toronto, ECE532 Digital System Design.

TIMA. (n.d.). *System Level Synthesis*. Retrieved from Investigating Application-Specific, Multiprocessor System-on-Chips: http://tima-sls.imag.fr/www/research/nocgen/installation

Torres, L., Benoit, P., Sassatelli, G., & Robert, M. (n.d.). *An Introduction to Multi-Core System on Chip-Trends and Challenges*. France: University of Montpellier 2.

van der Putten, P., Voeten, J., Geilen, M., & Stevens, M. (n.d.). *System Level Desing Methodology*. Holanda: Section of Information and Communication Systems, Eindhoven University of Technology.

Vazquez, L. (n.d.). Tutorial para el uso de imágnes en Octave.

Wolf, W. (2004, Jul.). *Hardware-Software Co-Design of Embedded Systems, Vol. 82, No 7*. Proceedings of the IEEE.

Wolf, W. (2007). *High Performance Embedded Computing*. Multiprocessor Architectures, Elsevier.

Wolf, W., Amine, A., & Martin, G. (2008, Oct.). *Multiprocessor System-On-Chip (MPSoC) Technology, Vol. 27, No 10*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

Xilinx. (2006, Mar.). LogiCORE Multi-Channel OPB External Memory Controller. Product Specification, DS500.

Xilinx. (2010, Mar. 4). LogicCORE XPS SYSACE (System ACE) Interface Controller (v1.01a). DS583.

Xilinx. (2010, Sep.). LogiCORE IP Processor Local Bus. PLB v4.6, DS531.

Xilinx. (2010, Jan.). ML401/ ML402/ ML403 Evaluation Platform. User Guide, UG080.

Xilinx. (2010, Jan.). PowerPC 405 Processor Block Reference Guide. Embedded Development Kit, UG018.

Xilinx. (2010, May. 3). XPS Thin Film Transistor (TFT) Controller (v2.01a). EDK, DS695.

Xilinx. (2011, Jul. 6). AXI Interface Based ML605/SP605 MicroBlaze Processor Subsystem. EDK Software Tutorial, UG670 (v4.0).

Xilinx. (2011). EDK Concepts, Tools, and Techniques. A Hands-On Guide to Effective Embedded System Design, UG683, EDK 13.2.

Xilinx. (2011, Oct.). Getting Started with the Xilinx Virtex 6 FPGA ML605 Evaluation Kit. Virtex 6 Getting Started Guide, UG533.

Xilinx. (2011, Jul. 6). ISE Simulator (ISim), In-Depht Tutorial. Guide UG682 (v13.2).

Xilinx. (2011, Jun. 22). LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11e). EDK, DS449.

Xilinx. (2011, Jul 6). LogiCore IP Multi-Port Memory Controller (MPMC) (v6.04.a). EDK, DS643.

Xilinx. (2011, Mar. 1). MicroBlaze Debug Module (MDM) (v2.00b). EDK, DS641.

Xilinx. (2011, Jul. 6). OS and Libraries Document Collection. EDK, UG643.

Xilinx. (2011, Mar. 1). PlanAhead User Guide. UG632 (v13.1).

Xilinx. (2011, Jul.). Platform Specification Format Reference Manual. Embedded Development Kit (EDK) 13.2, UG642.

Xilinx. (2011, Mar. 1). Power Methodology. Guide UG786 (v13.1).

Xilinx. (2011, Feb.). Virtex 6 CXT Family Data Sheet. Product Specification, DS153.

Xilinx. (2012, Jan.). LogiCORE IP MicroBlaze. Micro Controller System, DS865.

Xilinx. (2012, Jan. 18). XPower Estimator User Guide. UG440 (v13.4).

Xilinx. (n.d.). EDK Concepts, tools and techniques. UG638 EDK 12.2.

ANEXOS

ANEXO 1

TEORÍA DE CIRCUITOS SECUENCIALES

ANEXO 2

TUTORIAL DE CONTROL DE INTERFAZ XPS SYSACE Y LIBRERÍA LIBXIL FATFILE SYSTEM

ANEXO 3

TUTORIAL DE CONTROL TFT PARA SALIDA DE VIDEO DIGITAL (DVI) Y LIBRERÍA XTFT.H

ANEXO 4

TUTORIAL DE GENERACIÓN NOC HERMES EN LENGUAJE VHDL MEDIANTE LA HERRAMIENTA ATLAS

ANEXO 5

TUTORIAL DE HERRAMIENTA PARA CREAR O IMPORTAR NUEVOS DISEÑOS DESDE LA PLATAFORMA ISE HACIA EL XPS

ANEXO 6

TUTORIAL DE GENERACIÓN DE UNA ARQUITECTUA MPSOC SOBRE FPGA ML605 VIRTEX-6

ANEXO 7

CÓDIGO VHDL DE NOC HERMES

ANEXO 8

CÓDIGO VHDL DE LA INTERFAZ DE RED

ANEXO 9

CÓDIGO VHDL DE NOC HERMES CON NI

ANEXO 10

CÓDIGO DE LA APLICACIÓN DE ESTEGANOGRAFÍA

ANEXO 11

ESTIMACIÓN DE POTENCIA ESTÁTICA PARA UNA ARQUITECTURA INTERCONECTADA POR NOC HERMES

ANEXO 12

ESTIMACIÓN DE POTENCIA ESTÁTICA PARA UNA ARQUITECTURA INTERCONECTADA POR BUSES FSL

ANEXO 13

DIAGRAMA DE LA ARQUITECTURA MPSOC CONECTADA POR BUSES FSL

ANEXO 14

DIAGRAMA DE LA ARQUITECTURA MPSOC CONECTADA POR NOC HERMES

INDICE DE FIGURAS

INDICE DE TABLAS

GLOSARIO DE TÉRMINOS

A

ACK: Mensaje que el destino de la comunicación envía al origen de ésta para confirmar la recepción de un mensaje.

AMBA: *Advanced Microcontroller Bus Architecture*

ARM: Arquitectura RISC de 32 bits ideal para aplicaciones de baja potencia.

ASIC: *Application Specific Integrated Circuit,* Circuito Integrado para Aplicaciones Específicas.

AXI: *Advanced eXtensible Interface,* Es una familia de buses para micro controladores parte de ARM AMBA que proporciona beneficios de productividad, flexibilidad y disponibilidad en comparación con buses anteriores de AMBA.

B

BBD: *Block-Based Design*, Diseño Basado en Bloques

BIST: *Built-in self-test,* Mecanismo de prueba propia de varios dispositivos electrónicos, implementado para cumplir con requisitos de alta fidelidad y reducir tiempos en ciclos de reparación.

BIT: Es la unidad más pequeña de información

BITSTREAM: Secuencia continúa de bits.

BIT File: Archivo Bitstream del Ambiente Integrado de Software de Xilinx® (ISE™).

BRAM: *Block Random Access-Memmory,* Bloque de memoria de acceso aleatorio integrado en un dispositivo, a diferencia de la memoria distribuida.

BSB: *Base System Builder* (Constructor Base del Sistema). Es un asistente que facilita la creación de diseños completos en la herramienta Xilinx

Platform Studio (XPS). Este asistente guía paso a paso al desarrollador para seleccionar el procesador que se usará y algunas características de este como memoria cache, frecuencia de reloj, uso de depurador, entre otras.

BSP: *Board Support Package,* Entorno de programación para la implementación de código específico de soporte para un sistema operativo.

C

CAD: *Computer Aided Design*, Diseño Asistido por Computador

CORE CONNECT: Es una arquitectura de comunicación bus-microprocesador de IBM para sistemas embebidos en un chip. Fue diseñado para facilitar la integración y la reutilización del procesador, el sistema y los periféricos dentro del estándar de diseño de SoCs personalizados.

D

DCR: *Device Control Register,* Registro de Control de Dispositivo.

DDR3: *Double Data Rate 3,* Memoria de doble tasa de trasferencia de datos 3

DLMB: *Data-side Local Memory Bus,* Bus de Memoria Local, lado de Datos. Véase también: LMB.

DMA: *Direct Memory Access*, Acceso Directo a Memoria.

DSM: *Distributed Shared Memory,* memoria distribuida compartida, es un tipo de implementación hardware y software, en la que cada nodo tiene acceso a una amplia memoria compartida que se añade a la memoria privada de cada nodo.

DSP: *Digital Signal Processor*, Procesador Digital de Señales.

DTL: *Diode-Transistor Logic*, es una categoría de circuitos digitales en la cual la función de la puerta lógica la realiza una red de diodos, mientras que la función de amplificación es realizada por un transistor.

DVI: *Digital Video Interface,* Interface de Video Digital

E

ECLIPSE: Programa informático compuesto por un conjunto de herramientas de programación de código abierto y multiplataforma, utilizado para desarrollar entornos de desarrollo integrados, IDE.

EDA: *Electronic Desing Automation,* Es una categoría de herramientas de software para el diseño de sistemas electrónicos como tarjetas de circuitos impresos y circuitos integrados.

EDK: *Xilinx Embedded Development Kit*, Kit de Desarrollo Embebido de Xilinx.

ELF file: *Executable and Linkable Format file*, Archivo de Formato Ejecutable y Enlazable. Teniendo el fichero *system.bit* y el ejecutable ya compilado ".elf", se combinan ambos con la herramienta "Data2Mem" y resulta el fichero *download.bit.* Este es el fichero para programar definitivamente el FPGA.

EMC: *External Memory Controller*, Controlador de Memorias Externas

ESCALABILIDAD: Propiedad deseable de un sistema o proceso que indica su habilidad para reaccionar y adaptarse a un crecimiento continuo, sin perder calidad en los servicios ofrecidos.

ESPE: Escuela Politécnica del Ejército, es la Institución de enseñanza superior para la cual se está desarrollando el presente proyecto de grado.

F

FATfs (XilFATfs): Sistema *LibXil FATFile*. El archivo *XilFATfs* sistema de acceso a la librería, proporciona acceso de lectura/escritura para los archivos almacenados en una Xilinx *SystemACE CompactFlash*.

FIFO: *First Input, First Output*

FMC: *FPGA Mezzanine Card*, Estándar ANSI/VITA que define a los módulos de entrada y salida para la conexión entre un FPGA o con cualquier otro dispositivo con capacidades de entradas y salidas reconfigurables.

FPGA: *Field Programmable Gate Array*, Arreglos de Compuertas Programable en Campo.

FSL: *MicroBlaze Fast Simplex Link*. Interfaces de datos en vivo unidireccionales punto a punto, ideal para aceleración de *hardware*.

G

GALS: *Globally Asynchronous Locally Synchronous,* Modelo de computación basado en la interacción de comunicación de elementos síncronos con elementos asíncronos.

GPIO: *General Purpose Input and Output*, Entradas y Salidas de Propósito General. Un periférico de 32-bit que se conecta al bus de periféricos en chip.

GNU: Sistema operativo similar a Unix que es software libre. Constituido a partir de un conjunto de aplicaciones, bibliotecas y herramientas de programación utilizadas por los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y modificar su operatividad y funcionamiento.

GPOS: *General Purpose Operating Systems,* Sistemas Operativos de Propósito General

GUI: *Graphical User Interface,* Interface Gráfica de Usuario

H

HANDSHAKE: Protocolo automatizado de negociación que establece de forma dinámica los parámetros de un canal de comunicaciones establecido entre dos entidades antes de que comience la comunicación normal por el canal

HDL: *Hardware Description Language*, Lenguaje de Descripción de Hardware.

HEAP: Estructura de datos del tipo árbol, con información perteneciente a un conjunto ordenado de datos.

I

IDE: *Integrated Design Environment*, Ambiente de Diseño Integrado.

ILMB: *Instruction-side Local Memory Bus*, Bus de Memoria Local, lado de Instrucciones. Véase también: LMB.

INTERBLOQUEO: Bloqueo permanente de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros.

IOPB: *Instruction-side On-chip Peripheral Bus*, Bus de Periféricos On-Chip, lado de Instrucciones. Véase también: OPB.

IP-CORE: *Intellectual Property*, Núcleos o bloques de propiedad Intelectual

IPLB: *Instruction Processor Local Bus*

ISE: *Integrated Software Environment*, Es una herramienta de software producido por Xilinx para la síntesis y el análisis de los diseños de HDL, lo que permite al desarrollador sintetizar sus diseños, realizar análisis de simulaciones, examinar diagramas RTL, simular la reacción de un diseño a diferentes estimulos, y configurar el dispositivo de destino con el programador.

ISS: *Instruction Set Simulator*, Simulador del Conjunto de instrucciones.

J

JERARQUÍA DE BUSES: Estructura de interconexión de buses en la cual se prioriza la comunicación acorde a la velocidad del elemento de procesamiento.

JTAG: *Joint Test Action Group*, Prueba de Grupo de Acción Conjunta, es el nombre común utilizado para la norma IEEE 1149.1 titulada *Standard Test Access Port and Boundary-Scan Architecture*, utilizada para testear puertos de acceso en PCB(*Printed Circuit Board*) utilizando escaneo de límites.

L

LAYOUT: Esquema de distribución dentro de los elementos de un diseño.

LATENCIA: Es la suma de retardos temporales en una comunicación, producido por la demora en la propagación y transmisión de paquetes.

LIBGEN: Generador de Librerías y subcomponentes de la tecnología Xilinx Platform Studio.

LibXil Standard C Libraries: Las librerías de EDK y controladores de dispositivos proporcionan funciones de las librerías estándar de C, así como

funciones para acceder a los periféricos. Libgen, automáticamente configure las librerías de EDK para cada proyecto basado en los archivos MSS.

LMB: *Local Memory Bus*, Bus de Memoria Local. Un bus síncrono de baja latencia utilizado principalmente para acceder a la BRAM. El procesador *MicroBlaze* contiene un bus LMB de instrucciones y un bus LMB de datos.

LUT: *Look-up Table,* Tabla de Búsqueda para FPGA que contienen información de lógica combinacional

M

MDD File: *Microprocessor Driver Description file*, Archivo de Descripción del controlador por microprocesador.

MDM: *Microprocessor Debug Module*, Modulo de Depuración del Microprocesador.

MFS File: *LibXil Memory File System*, Sistema de Archivos de Memoria *LibXil*. El MFS proporciona la capacidad del usuario para administrar la memoria del programa en forma de identificadores de archivo.

MHS file: *Microprocessor Hardware Specification file*, Archivo de Especificación de Hardware de Microprocesador.

MLD file: *Microprocessor Library Definition file*, Archivo de Definición de Librerías de Microprocesador.

MPD file: *Microprocessor Peripheral Definition file*, Archivo de Definición de Periféricos del Microprocesador. El archivo MPD contiene todos los puertos disponibles y los parámetros de hardware para un periférico.

N

NACK: Mensaje de protocolo que se envía para informar de que en la recepción de una trama de datos ha habido un error.

NETLIST: Lista de conexiones que describe la conectividad de un diseño electrónico.

NCF file: *Netlist Constraints file*, Archivo de Restricciones del *Netlist*.

NI: *Network Interface,* Interface de Red.

NGC file: El archivo NGC, es un archivo *netlist* que contiene tanto los datos de diseños lógicos como los de limitaciones. Este archivo reemplaza tanto los archivos EDIF y NCF.

NGD file: *Native Generic Database file*, Archivo de Base de Datos Genéricos Nativos. El archivo NGD es un archivo *netlist* que representa a todo el diseño.

NGO File: Un formato específico de Xilinx, archivo binario que contiene una descripción lógica del diseño en términos de sus componentes originales y la jerarquía.

NOC: Network on Chip

O

OCP: *Open Core Protocol,* Es un protocolo de licencia abierta, que define una interfaz independiente del bus, siendo configurable y escalable para las comunicaciones de un chip.

OPB: *On-chip Peripheral Bus*, Bus de Periféricos en chip.

OVERHEAD: Tiempo extra que determinada operación puede requerir si se le añade una funcionalidad o mecanismo de medida de rendimiento.

P

PARALELISMO: Función computacional que realiza un procesador para ejecutar varias tareas al mismo tiempo.

PARALELISMO INTRÍNSECO: es una forma de computación en la cual varios cálculos pueden realizarse simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo.

PBD: *Platform Based Design*, Diseño Basado en Plataforma

PCB: *Printed Circuit Board*, Tarjeta de Circuitos Impresos

PCI: Interconexión de Componentes Periféricos, es un bus estándar para conectar dispositivos periféricos de forma dinámica.

PLB: *Processor Local Bus*.

PLD: *Programmable Logic Devices*

Plug and Play: Tecnología que permite a un dispositivo computacional ser integrado a otro dispositivo, sin tener que ser configurado mediante jumpers o software específico.

Q

QoS: *Quality of Service,* Calidad de Servicio, Tecnologías que permiten aplicar un tratamiento específico a un determinado tipo de trafico computacional.

R

RISC: *Reduced Instruction Set Computer*, Computadora con Set de Instrucciones Reducidas

RAM: *Random-Access Memmory,* Memoria de Acceso Aleatorio

ROM: *Read-Only Memmory,* Memoria de sólo Lectura

RTL: *Register Transfer Level*, Transferencia entre registros

RTOS: *Real Time Operating System*, Sistema Operativo en Tiempo Real

S

SCP: *Soft Core Proccessor,* Microprocesador implementado a través de lenguaje de descripción de hardware sobre un dispositivo lógico programable o FPGA.

SDK: Software Development Kit.

SDRAM: *Synchronous Dynamic Random-Access Memory,* Familia de memorias dinámicas de acceso aleatorio que tiene una interfaz síncrona.

SIMD: *Single Instruction, Multiple Data,* Una instrucción, múltiples datos, técnica empleada para conseguir paralelismo a nivel de datos.

SMP: *Symmetric Multi-Processing,* Arquitectura computacional en la que dos o más unidades de procesamiento comparten el acceso a una única memoria central y permite que cualquier procesador trabaje en cualquier tarea sin importar su localización en memoria.

SWITCH BEST-EFFORT: El switch realiza su mejor esfuerzo para entregar los paquetes pero no garantiza que estos lleguen y si en el proceso algo resulta mal y el paquete se pierde, la red no hace nada por recuperarlo. Únicamente su mejor esfuerzo.

SWITCH GARANTY-SERVICES: la comunicación del switch es redundante, por lo cual se garantiza la entrega de paquetes a sus destinos correspondientes en la red.

SOC: *System on Chip*

STACK: Estructura de datos en forma de lista, en la que el modo de acceso a sus elementos es de tipo LIFO (ultimo en entrar, primero en salir) que permite almacenar y recuperar datos.

T

TABLAS DE RUTEO INDEXADAS: Documento electrónico que almacena las rutas a los diferentes nodos de una red de comunicaciones.

TDD: *Timing-Driven Design*, Diseño Guiado por Tiempo

THROUGHPUT: Volumen de trabajo o de información que fluye a través de un sistema.

TIEMPO DE DISPERSIÓN: Corresponde al tiempo que tarda cualquier Switch de Hermes en establecer comunicación con el destino mediante el algoritmo de ruteo.

TOKENPASS: Protocolo de acceso al medio en el cual los nodos están conectados a un bus o canal para comunicarse con el resto. En todo momento hay un testigo (token) que los nodos de la red se van pasando, y únicamente el nodo que tiene el testigo tiene permiso para transmitir.

TLM: *Transaction-Level Modeling,* Es un enfoque de alto nivel para el modelado de sistemas digitales donde los detalles de comunicación entre los módulos se separan de los detalles de la implementación de las unidades funcionales o de la arquitectura de comunicación.

TTM: *Time to Market,* Tiempo que transcurre entre que un producto es concebido y está disponible para la venta

U

UART: Universal Asynchronous Receiver-Transmitter.

UCF: *User Constraints File***,** es un archivo ASCII, que contiene las limitaciones (*constraints*) de diseño lógico, este UCF se combina con un EDIF netlist, y se crea un archivo NGD el cual es leído y producido por un NGDbuild durante este proceso

V

VGA: *Video Graphic Adapter*

VHDL: VHSIC *Hardware Description Language*.

VLSI: *Very Large Scale of Integration*

W

WRAPPERS: Bloques de propiedad intelectual, que contienen estructuras de comunicación programable mediante VHDL para interconectar diversos segmentos o elementos de un sistema específico.

X

XCL: *Xilinx Cache Link*. Una interfaz de memoria cache externa de alto rendimiento disponible en procesador MicroBlaze.

Xilkernel: Un pequeño, extremadamente modular y configurable RTOS para las plataformas embebidas de Xilinx.

XMD: *Xilinx Microprocessor Debugger*.

XMP File: *Xilinx Microprocessor Project file*. Este es el archive de proyecto de alto nivel para un diseño EDK.

XPS: Xilinx Platform Studio. El ambiente GUI en el cual usted se puede desarrollar un sistema embebido.

FICHA DE ENTREGA

El proyecto fue entregado al Departamento de Eléctrica y Electrónica y reposa en la Escuela Politécnica del Ejército desde:

Sangolquí, a _____

ELABORADO POR:

Chicaiza Casa Wilson Mauricio	Verdezoto Dávalos Daniel Gonzalo
CI: 1716304751	CI: 1716909625

AUTORIDADES:

Ing. LUIS OROZCO

DIRECTOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL