ESCUELA POLITECNICA DEL EJÉRCITO

DEPARTAMENTO DE ELECTRICA Y ELECTRONICA

CARRERA DE INGENIERIA EN ELECTRONICA, AUTOMATIZACION Y CONTROL

PROYECTO DE GRADO PARA LA OBTENCION DEL TITULO EN INGENIERIA

DISEÑO DE HARDWARE Y SOFTWARE DE SYSTEMS-ON-CHIP EMPLEANDO TECNOLOGIA XILINX EDK.

JULIO CESAR CADENA SALAZAR

JUAN GABRIEL MOLLOCANA LARA

SANGOLQUI – ECUADOR

2012

CERTIFICACIÓN

Certificamos que el presente proyecto de grado: "DISEÑO DE HARDWARE Y SOFTWARE DE SYSTEMS-ON-CHIP EMPLEANDO TECNOLOGIA XILINX EDK", fue desarrollado en su totalidad por los señores Julio César Cadena Salazar y Juan Gabriel Mollocana Lara, bajo nuestra dirección.

Atentamente,

Ing. Hugo Ortiz	Ing. Vanessa Vargas
DIRECTOR	**CODIRECTOR**

AGRADECIMIENTOS

Principalmente a Dios y a mis padres por ser los guias que forjan mi camino dia a dia, inculcándome valores como la honestidad y el respeto a los demás. En ellos siempre podré confiar.

A mi hermana Shirley y amigos por haberme brindado su apoyo y confianza incondicional durante el desarrollo de este proyecto.

A la Escuela Politecnica del Ejército, gracias por haber contribuido eficientemente en mi educación, en sus aulas y laboratorios he pasado los mejores y más bellos momentos de mi vida, este establecimiento ha constituido mi segundo hogar.

Mis más sinceros agradecimientos a todos mis profesores que han aportado con sus conocimientos en mi formación profesional en especial a los Ingenieros Byron Navas, Vanessa Vargas, y Hugo Ortiz, ya que han contribuido en la elaboración de este proyecto.

A todos ellos mi gratitud.

Julio César Cadena Salazar

AGRADECIMIENTOS

A mis padres por su incondicional amor y confianza a lo largo de cada etapa de mi vida.

A mis hermanos por su apoyo y compañía durante el desarrollo de este proyecto.

A la Escuela Politécnica del Ejército y sus profesores por las enseñanzas y oportunidades brindadas, y en especial, a los ingenieros Vanessa Vargas, Pablo Ramos, y Hugo Ortiz por toda su ayuda, tiempo y ejemplo.

Juan Gabriel Mollocana Lara

DEDICATORIA

A mis queridos padres Julio Cadena y Rosa Salazar que me dieron la vida y me prodigaron sus cuidados, a sus esfuerzos, los cuales han contribuido para la culminación de mis estudios, dedico este proyecto de grado, fruto de mis desvelos y mi decisión por conseguir un futuro prometedor.

Julio César Cadena Salazar

A mi familia, Mirian, Homero, Paulo, Alvaro y Evelyn. A ustedes dedico todo mi esfuerzo y trabajo.

Juan Gabriel Mollocana Lara

Ademas, este proyecto de esfuerzo, dedicación y constancia, va dedicado a los futuros investigadores, diseñadores, y desarrolladores de chips del Departamento de Eléctrica y Electrónica de la ESPE, ya que junto a este trabajo, forjaran un mejor mañana colocando el nombre del Ecuador muy en alto, puesto que en este país habrá hombres y mujeres con capacidad de manufacturar productos de alta tecnología.

Julio Cadena y Gabriel Mollocana

PRÓLOGO

La tendencia de la tecnología actual está basada en dispositivos electrónicos o sistemas embebidos que posean más funciones, un mayor rendimiento, consuman menos potencia, tengan un menor tamaño y menor precio. Además, que estos sistemas estén disponibles lo antes posible en el mercado de consumidores. Estas características entre otras, fue lo que motivó a la industria electrónica a crear una nueva metodología en el diseño de circuitos integrados, de esta manera es como aparecen los *System on Chip* (SoC).

En este proyecto se realizó el estudio completo de la tecnología SoC; estado del arte, sus componentes, arquitectura, proceso y metodología de diseño, así como las herramientas principales con las que se desarrolla.

El objetivo principal de este proyecto fue crear un sistema embebido empleando en el Spartan 6 FPGA Embedded Kit, que permita controlar la temperatura interna de una planta de calefacción. Para ello, se realizó el co-diseño de *hardware* y *software* del SoC (circuito integrado - elemento controlador) empleando las herramientas y procedimiento de diseño propuesto por Xilinx.

El resultado que se obtuvo fue un SoC diseñado desde su fase conceptual, en base a especificaciones técnicas de la planta. En el hardware consta la integración de componentes gobernados por un procesador y el software permite la selección entre dos sistemas de control (ON-OFF y PID) desarrollados sobre un Sistema Operativo en Tiempo Real (RTOS).

En el CAPITULO 1 se realiza la introducción al diseño de System on Chip sobre FPGAs y porque realizar aplicaciones embebidas en el Ecuador.

En el CAPITULO 2 se presenta la investigación realizada acerca de los principales temas que conforman el diseño SoC. Estos temas son: estado del arte, definición de SoC, historia de los SoC, IP Cores, tipos de IP Cores, arquitectura, proceso de diseño, co-diseño de *hardware* y *software*, sistemas embebidos, sistemas embebidos en tiempo real, y RTOS.

En el CAPITULO 3 se estudia las herramientas de diseño de Xilinx tanto de *hardware* como de *software* sobre las cuales se va a diseñar un *System on Chip*. En este capitulo se incluye también, las configuraciones que se deben realizar a varios *IP Cores*.

En el CAPITULO 4 se realiza un resumen de tutoriales de *hardware* y *software* embebidos con el fin de obtener un procedimiento de diseño de SoCs. Además, se muestra el resumen de tres guias de estudio que servirán para el aprendizaje del diseño SoC.

En el CAPITULO 5 se plantea la propuesta de las especificaciones de diseño que deberá cumplir un SoC enfocado a una aplicación de automatización y control que controle una planta de temperatura.

En el CAPITULO 6 se presenta el procedimiento de diseño utilizado para el SoC controlador de la planta de temperatura.

En el CAPITULO 7 se trata acerca de los resultados obtenidos en cada capa del SoC.

En el CAPITULO 8 se dan a conocer las conclusiones a las que se llegaron después de haber obtenido los resultados deseados, asi como las recomendaciones necesarias para facilitar el diseño de *System on Chip*. Ademas, se indican los trabajos futuros que se podrían realizar tomando como base este proyecto.

En el CAPITULO 9 se muestra la bibliografía empleada para el análisis y estudio de *System on Chip*.

INDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

System on chip es una definición que ha tomado mucho énfasis en países que basan su economía y desarrollo en la fabricación de productos de alta tecnología. La empresa XILINX de Estados Unidos, que es el principal fabricante de FPGAs[1] en el mundo, ha facilitado el desarrollo de nuevos productos a través de estos dispositivos.

Empíricamente los FPGAs son chips vírgenes, es decir, chips que se pueden transformar en cualquier circuito deseado. Estos FPGAs aparecen con la finalidad de permitir a los diseñadores de circuitos integrados plasmar sus ideas, en un menor tiempo, realizando constantes pruebas y cambios, hasta llegar al objetivo deseado. La disminución en el tiempo de diseño, así como en gastos de fabricación, hace de los FPGAs la opción preferida para el desarrollo de nuevas propuestas en comparación a los otros tipos de ASICs[2] (Figura. 1.1.).

Los FPGAs, son la base para el diseño de los SoC (*System on Chip*, o sistema en chip). De acuerdo a Martin y Chang 2003, SoC es un circuito integrado complejo que integra la mayoría de elementos funcionales de un producto final completo dentro de un simple chip [1]. El uso de SoCs permite crear sistemas embebidos[3] de menor tamaño y que incorporen mayor tecnología.

[1] **FPGA:** Field Programmable Gate Array, Arreglo de Compuertas Programables en Campo.
[2] **ASIC:** Application Specific Integrated Circuit, Circuitos Integrados de Aplicación Especifica.
[3] **Sistemas Embebidos:** Son sistemas altamente integrados de aplicación especifica.

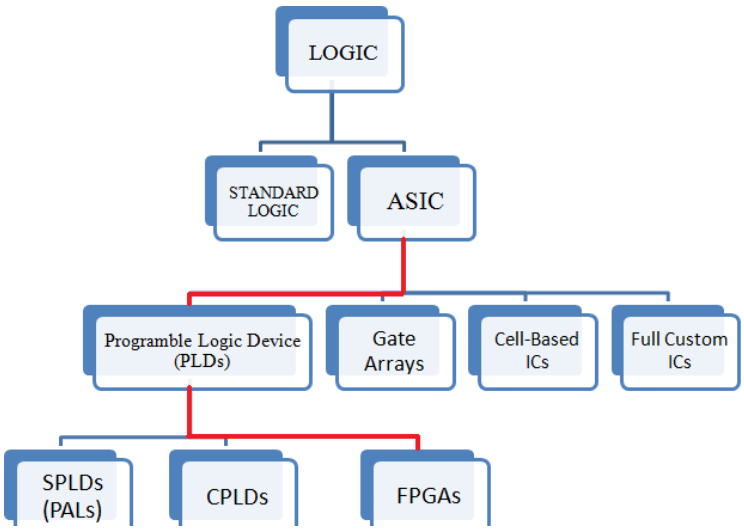

Figura. 1.1. Tipos de ASIC

Fuente: Dataquest

Por otra parte, el Ecuador, considerado un país en vías de desarrollo, ha sido limitado en el diseño de tecnología debido a costos y a la falta de profesionales capacitados en este campo. Sin embargo con el uso de FPGAs se puede iniciar con el estudio, diseño e implementación de SoCs de forma económicamente fiable.

En este marco, el propósito de este proyecto es implementar un SoC (*System-On-Chip*) orientado a una aplicación de automatización y control, realizando el co-diseño de *hardware* y *software*, empleando tecnologías y herramientas Xilinx. Cabe recalcar que la experiencia de este proyecto será trasferible a las otras carreras del DEEE: Telecomunicaciones, y Redes de Datos.

El diseño de *hardware* embebido se realizará empleando la herramienta Xilinx Platform Studio (XPS) y constará de al menos un procesador, memorias internas, externas, y cache de datos e instrucciones. Además, incluirá los IP-Cores necesarios para cumplir las especificaciones de la aplicación.

El *software* embebido se realizará empleando la herramienta de Xilinx Software Development Kit (SDK), y implementará un Sistema Operativo en Tiempo Real (RTOS), drivers y librerías. Así como, las APIs (Interfaz de Programación de Aplicaciones) necesarias para cumplir las especificaciones de la aplicación, y asi crear la arquitectura de *software* embebido del SoC.

Tecnológicamente, este proyecto se desarrollara de acuerdo al flujo de diseño embebido sugerido por Xilinx. Cabe recalcar que lo mas critico e importante del proyecto es diseñar y crear un System on Chip, lo cual es sumamente complejo, siendo la primera vez que se utiliza esta tecnología en el país y el diseño de la aplicación constituye solamente un ejemplo de utilización de un SoC en la carrera de automatización y control.

Por otro lado la realización de este proyecto pretende dar el primer paso para cambiar nuestro modelo de negocios, dejando de ser un país consumidor de tecnología y pasando a ser creadores de la misma. Además se dejará una base con tutoriales y guías de laboratorio para futuros proyectos con la finalidad de que se desarrolle el campo de diseño de chips en el país.

Cabe mencionar que esta investigación está aprobada como proyecto de Iniciación Científica ya que la Escuela Politécnica del Ejército (ESPE), lo consideró como un tema de vanguardia y de alta tecnología.

CAPÍTULO 2

ESTADO DEL ARTE

La evolución de la tecnología electrónica, ha dado pasos muy grandes durante el siglo pasado, desde que aparecieron las válvulas al vacio, pasando por los transistores, hasta llegar a los circuitos integrados o chips. La complejidad de los circuitos integrados incrementa en base al número de componentes electrónicos que llevan en su interior y a la tecnología de fabricación [74].

Para diseñar e implementar un dispositivo electrónico como un circuito digital, era necesario hacerlo con componentes discretos y, la metodología de diseño se denominaba diseño aleatorio o no estructurado, siendo el principal método de simplificación del sistema el de mapas de *Karnaugh*. Esta metodología de diseño tomaba mucho tiempo, su fabricación era costosa y ralentizaba el crecimiento tecnológico. Frente a esta necesidad surge el diseño estructurado basado en bloques funcionales (IP Cores)[4], y con ello el uso de una nueva metodología de diseño fundamentada en el uso de estos bloques. Adicionalmente, gracias al avance tecnológico y a la aparición de los Dispositivos Lógicos Programables (PLD) se ha logrado agilitar el diseño e implementación de nuevos productos.

El PLD más utilizado en la actualidad es el FPGA, gracias a su característica de reprogramabilidad en campo, siendo una ventaja frente a otras soluciones y abaratando costos, inclusive después de la manufactura inicial mediante actualizaciones.

El estado del arte o nivel más avanzado de la tecnología en el campo de diseño de chips, se ha logrado mediante la implementación de sistemas embebidos basados en

[4] **IP Cores:** Bloques de hardware con funciones especificas, pre probadas, y verificadas.

Systems on Chip (SoC) sobre FPGA. Para esto, los fabricantes de FPGA se esmeran en promover el diseño de herramientas que faciliten el diseño de plataformas *hardware*, así como también, al desarrollo de herramientas para la construcción del software (SW) que se ejecutará sobre esta plataforma. Esto brinda la ventaja de adaptar el diseño a la necesidad concreta del sistema a implementarse [38].

Es importante señalar que uno de los principales fabricantes de FPGAs en el mundo es Xilinx, que ocupa cerca del 50% en ventas de FPGAs, seguidas por ALTERA y ATMEL. (Figura. 2.1).

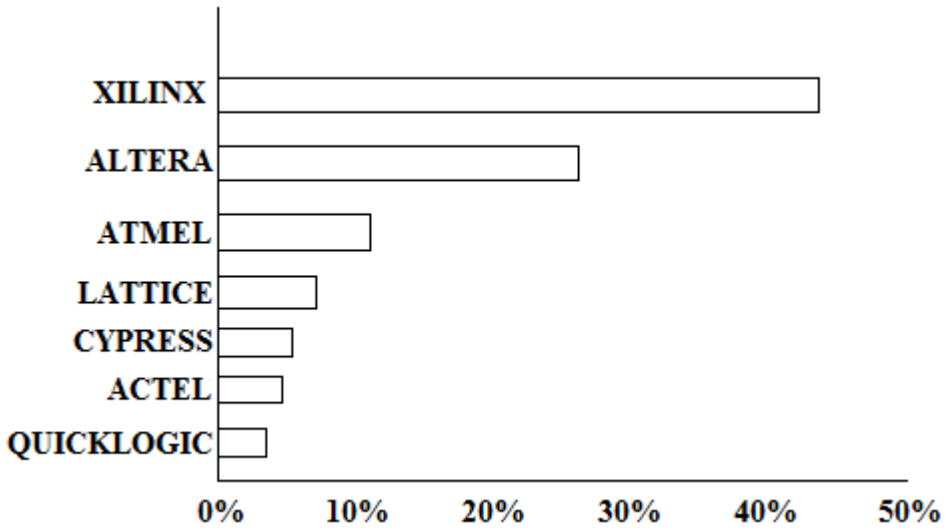

Figura. 2.1. Gráfica procentual de principales fabricantes de FPGAs en el mundo. Basado en: BOEMO, eduardo, 2005

Lo que actualmente buscan los diseñadores de dispositivos electrónicos es integrar un mayor número de elementos en un simple chip, siempre y cuando no se incremente el tamaño del mismo, disminuya el tiempo de salida del producto al mercado (time-to-market) y aumente el tiempo del producto en el mercado (time-in-market).

El time-to-market se reduce con el uso de SoCs desarrollados bajo la tecnología de diseño estructural, e implementados sobre FPGAs. Mientras que el time-in-market aumenta ya que permite al sistema ser actualizado y adaptado a nuevas necesidades.

Los productos basados en SoCs se encuentran abundantemente en el mercado, y van desde dispositivos portátiles, como son los relojes digitales, celulares, reproductores de

MP3, hasta grandes instalaciones estacionarias, como son las luces de tráfico, los controladores de fábrica y los sistemas de control de las centrales eléctricas.

En general los sistemas embebidos basados en SoCs están diseñados para hacer alguna tarea específica, en lugar de ser un computador de propósito general para múltiples tareas [10].

2.1 SOC (SYSTEM ON CHIP)

Se define a un SoC como un circuito integrado complejo, o conjunto de chips integrados, el cual agrupa los principales elementos funcionales o subsistemas de un producto final completo en una sola entidad [4]. En la Figura. 2.2, se observa varios elementos que conforman un SoC, entre los que se destacan un procesador programable, memorias on chip, unidades de aceleración implementadas en *hardware*, interfaces con dispositivos periféricos, y aunque no consta en el gráfico, en un futuro podrían incluir componentes analógicos y *opto/microelectronic mechanical system* (O/MEMS) [1].

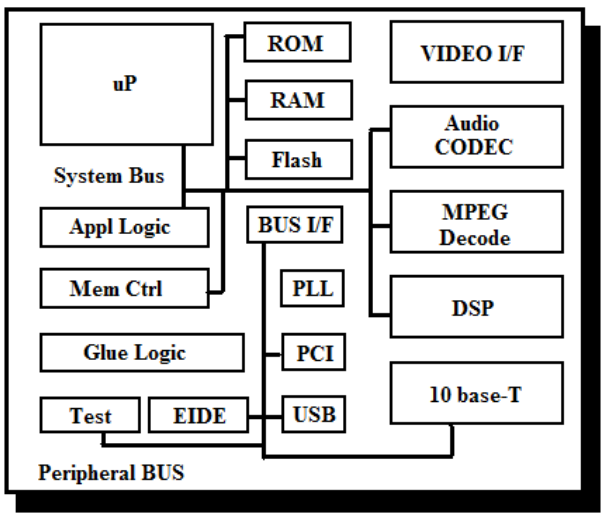

Figura. 2.2. ***System on Chip***

Fuente: MARTIN, grant y CHANG, henry, 2003

Los elementos mencionados anteriormente junto a muchos otros, son la razón de ser de los SoC ya que estos se basan en el diseño y reutilización de los bloques de propiedad intelectual *IP Cores* [2]. A fin de lograr la flexibilidad y escalabilidad deseadas en el

diseño de SoCs, se utiliza una metodología de co-diseño de *hardware* y software, en la que los IP Cores constituyen el HW del sistema.

La flexibilidad está asociada principalmente a la diversidad de *IP-Cores* que se pueden integrar en un sistema a ser implementado. Mientras que, la escalabilidad está referida al número de *IP-Cores* que pueden coexistir sin alterar el sistema al ser añadidos posteriormente [8].

2.1.1 HISTORIA

El diseño de SoCs inició una revolución hace varios años atrás. "Esta revolución comenzó a mediados de los 90, cuando la tecnología de semiconductores alcanzó la tecnología de fabricación de los 350 y 250 nanómetros (o comúnmente 0,35 y 0,25 micrones)"[5].

Desde el inicio del uso del transistor en el diseño de circuitos integrados, se ha cumplido la ley de Moore (Figura. 2.4), formulada hace 40 años, que expresa que "el numero de transistores en un chip de silicio se duplica aproximadamente cada dos años; al igual que su velocidad de funcionamiento"[6].

.

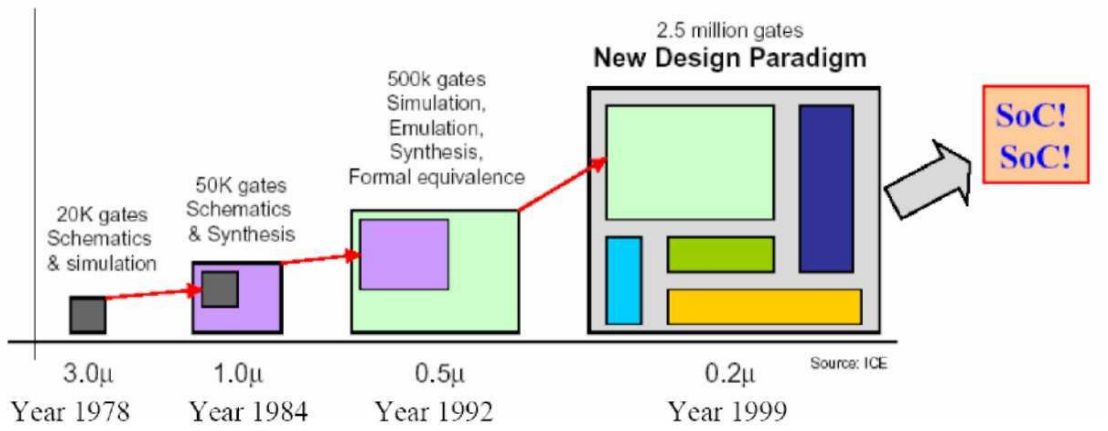

Figura. 2.3. Avances en la Integración

Fuente: ZHENGM, li-rong, 2007

[5] MARTIN, grant y CHANG, henry, *Winning the SoC Revolution - Experiences in Real Design*, Kluwer Academic Publisher, Estados Unidos 2003, 311 páginas

[6] GARCIA, almudever, *Evolución de la Metodología de Diseño y la Tecnología*, 2011 [74]

Como se muestra en la Figura. 2.3, para el año 1978 todos los transistores ocupaban la totalidad del espacio del chip con un proceso de la tecnología de silicio de 3 micrones[7].

Desde el año 1984, año de introducción de los FPGAs al mercado, hasta el año 1992, se logró disminuir la tecnología de fabricación de silicio a 1 micrón y 0.5 micrones respectivamente, con lo cual se ganó espacio dentro del chip. A mediados de los 90s con la tecnología de proceso de 350 y 250 nanómetros nace una nueva expectativa, un nuevo paradigma de diseño. Gracias a la reducción de la tecnología se pudo incluir mayor cantidad de transistores en un mismo espacio, lo que permitió que un chip contenga internamente un sistema completo son sus respectivos bloques funcionales o *IP Cores*. A partir de ese momento, nacen los SoCs que actualmente son fabricados con un proceso de tecnología de 22 nanómetros. La idea fundamental es convertir un PCB[8] con componentes discretos en un simple SoC integrado.

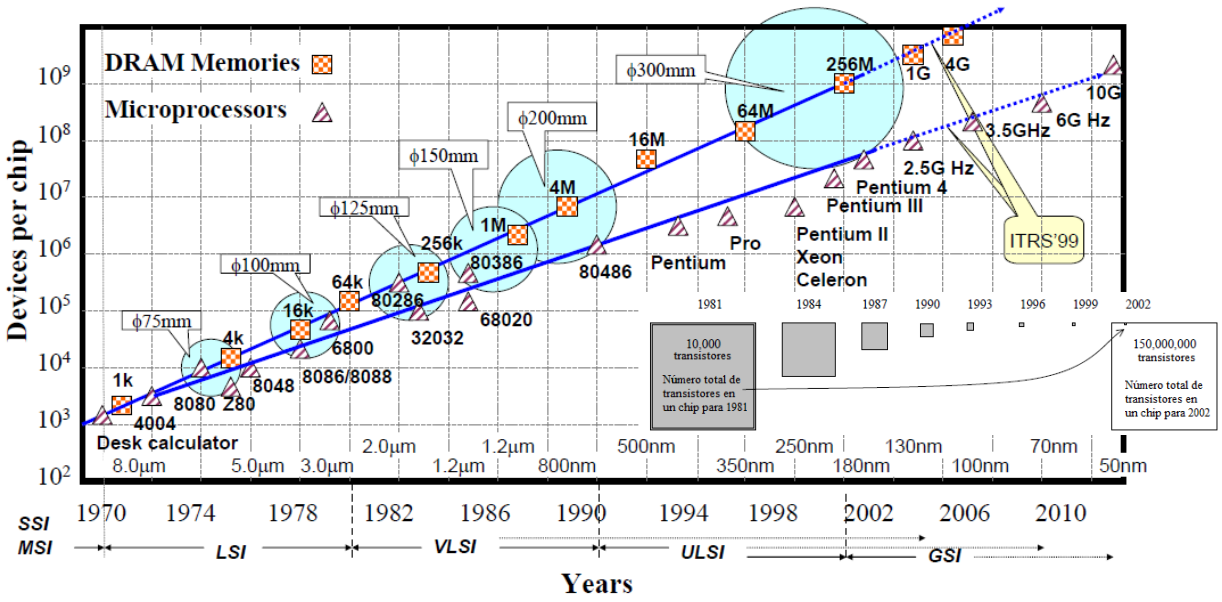

Figura. 2.4. Ley de Moore

Fuente: ZHENGM, li-rong, 2007 - LOCKWOOD, 2002

La reutilización de *IP Cores* ha sido un elemento clave en el diseño de circuitos integrados, de tal manera que el ITRS[9] ha manifestado que gracias a este concepto se disminuirá la "brecha de diseño o brecha de productividad", que se muestra en la Figura.

[7] El área donde se alojan los transistores dentro de la oblea de silicio se media en micrones y actualmente en nanómetros

[8] ***Printed Circuit Board***: Tarjeta de Circuitos Impresos que contiene varios componentes discretos interconectados a través de rutas o pistas de material conductor.

[9] International Technology Roadmap for Semiconductors (ITRS)

2.5. Esta brecha es la diferencia entre la capacidad de fabricación y la capacidad de diseño. Entendiéndose como capacidad de fabricación lo que las empresas son capaces de fabricar tecnológicamente. En tanto que, la capacidad de diseño se refiere al trabajo mensual entregado por un equipo de diseño. Esta brecha empeoró con la necesidad de bajar los costos de fabricación, lo que obliga a integrar más y más los diseños dentro de un simple circuito. Por otra parte, las limitaciones del "time-to-market" obligan a diseñar productos más rápidamente.

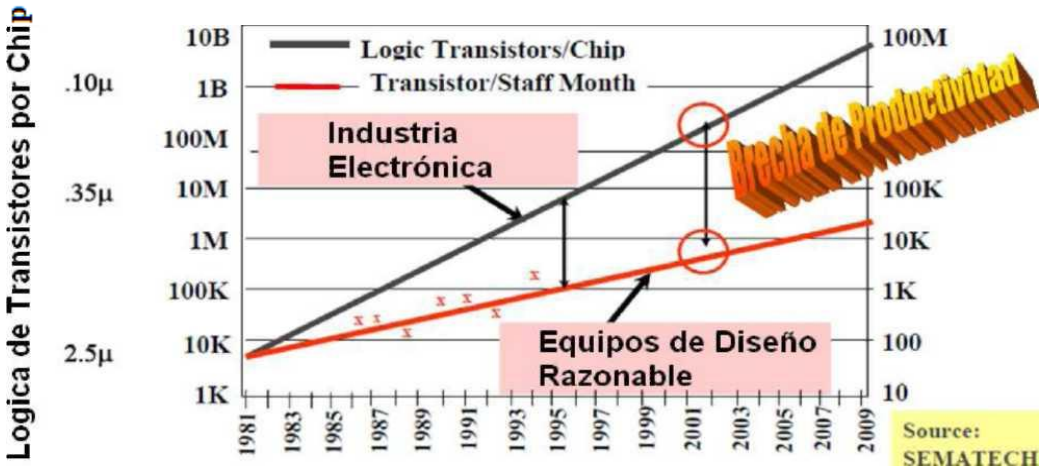

Figura. 2.5. Brecha de Diseño o de Productividad

Basado en: ZHENGM, li-rong, SEMATECH, 2007

Según Martin y Chang 2003, ha habido muchas sugerencias para atenuar esta crisis de productividad del diseño. Sin embargo, se percibe que la solución con el mayor impacto potencial en el diseño de productividad, es la reutilización de los bloques de Propiedad Intelectual (*IP Cores*), que se inició a mediados de los 90s.

Según los autores antes citados, la reutilización de los IP *Cores*, tuvo que trasladarse del 0 al 30% a mediados de los 90s, a unos extraordinarios altos niveles de 90 a 99%, ya que la industria requería:

- Nuevos métodos de diseño que permitan la creación de diseños de IP Cores reutilizables y verificables.
- Nuevos estándares de *IP Cores* para intercambio.
- La emergencia de una industria de propiedad intelectual.
- Resolución para los negocios con *IP Cores*.
- Una industria fuerte que lidere y haga posible para que todo esto suceda [1].

Como refuerzo al tema, los mismos autores señalan que hubo tres eventos formativos, que ocurrieron a mediados de los 90s para que suceda la revolución de los SoCs.

1. El primero y más significativo, fue establecer al **VSIA**[10], como la organización que se enfocaría en la reutilización de *IP Cores* y en todos los requerimientos de la industria listados anteriormente.
2. El segundo evento fue la creación de la ***Metodología de Reutilización*** para *soft IP Cores*[11], por equipos de *Mentor Graphics* y *Synopsys*, culminando en el Manual de la Metodología de Reutilización disponible desde 1997.
3. El tercer evento fue la creación, en Escocia, del proyecto ***ALBA***, que duro desde 1997 hasta 1999. Este proyecto incluía iniciativa gubernamental, industrial y académica, con el fin de centrar la economía de Escocia, en el diseño de productos de alta tecnología basados en SoCs y la reutilización de *IP Cores*[1].

2.1.2 IP CORES

Los *IP Cores* (*Intellectual Property Cores*) o Núcleos de Propiedad Intelectual son bloques con funciones preestablecidas, previamente probadas y verificadas por empresas desarrolladoras, para que posteriormente puedan ser integrados en sistemas SoC.

Una ventaja adicional a las anteriormente nombradas de la reutilización de *IP Cores*, es que ofrecen una gran reducción en el riesgo de diseño de nuevos dispositivos, ya que utilizan módulos pre-probados.

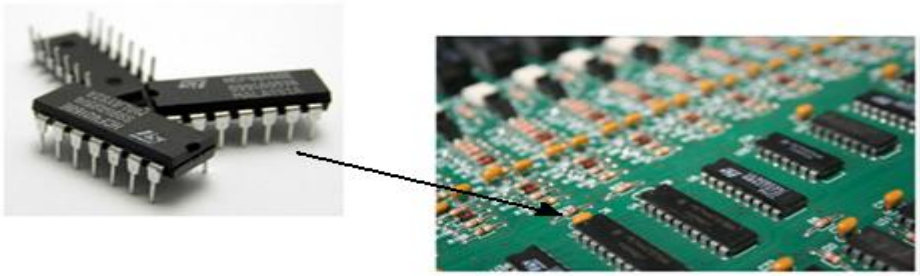

Figura. 2.6. Componentes Reales
Basado en: STMICROELECTRONICS, 2011

[10] **VSIA:** *Virtual Socket Interface Alliance*, fundado y públicamente anunciado en Septiembre de 1996.
[11] ***Soft IP Cores*:** Son un tipo de *IP Cores* descritos en código RTL (*Register Transfer Level*, Transferencia entre registros) sintetizable.

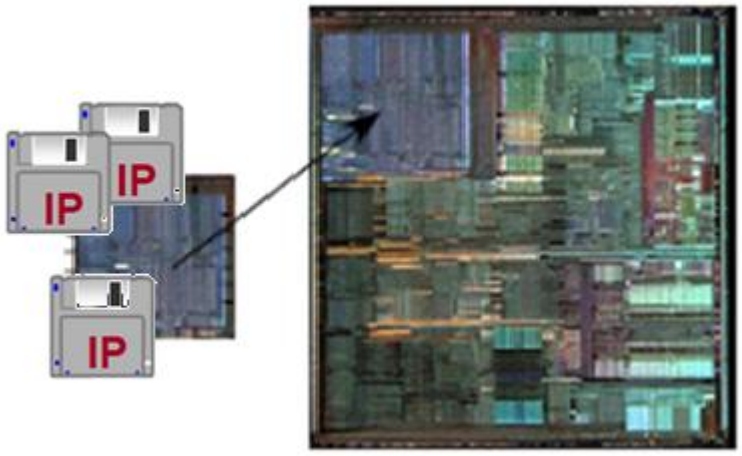

Figura. 2.7. Componentes Virtuales
Fuente: ZHENGM, li-rong, 2007

Tradicionalmente, diferentes componentes eran colocados e interconectados sobre una tarjeta (PCB) con la finalidad de cumplir una función específica (Figura. 2.6). Con la utilización de *IP Cores*, los chips individuales que conformaban los componentes en *hardware* fueron reemplazados por componentes virtuales, que cumplen las mismas funciones (Figura. 2.7). Esto brinda la ventaja de agrupar los componentes dentro de un mismo chip y disminuir notablemente el tamaño y consumo de potencia de los productos ofrecidos.

2.1.2.1 Tipos de IP Cores

Para describir un *IP Core* se usan distintos niveles de abstracción[12] (Figura. 2.8). Los usados comúnmente son:

- RTL (*Register Transfer Level*).- Descripción de operaciones y transferencias entre registros de *hardware*.
- *Gate Level*.- Descripción de operación a través de compuertas lógicas.
- *Layout*.- Descripción de operación a través de transistores.

Estos niveles definen su portabilidad, flexibilidad y previsibilidad. Entendiéndose por portabilidad como la capacidad de ser implementados sobre diversas tecnologías,

[12] Una capa de abstracción o nivel de abstracción es una forma de ocultar los detalles de implementación de ciertas funcionalidades [19]

mientras que la flexibilidad determina el grado de personalización para una aplicación específica, y la previsibilidad define su rendimiento en términos de *timing* y área dentro del futuro chip.

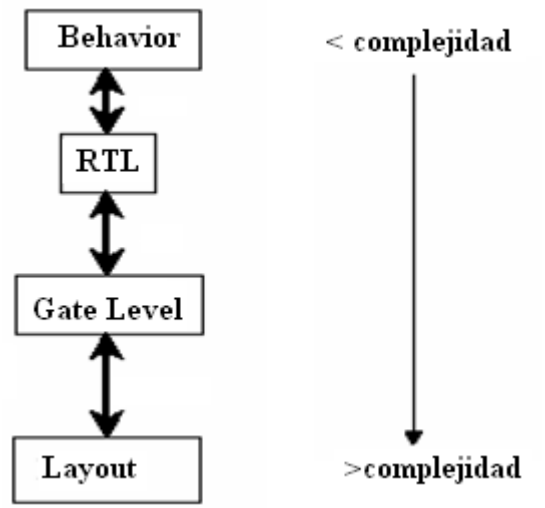

Figura. 2.8. Niveles de Abstracción
Basado en: MARTÍNEZ, Ricardo y TERÉS, lluis, 2010, Introducción a los IP Cores

Ricardo Martínez y Lluis Terés 2010, trabajadores del Centro Nacional de Microelectrónica de España, (CNM) definen los siguientes tres tipos de IP Cores:

- *Soft Cores.*
- *Firm Cores*
- *Hard Cores.*

La Tabla. 2.1 muestra un resumen de las características de cada tipo de *IP Core*.

TIPO	SOFT CORE	FIRM CORE	HARD CORE
NIVEL DE ABSTRACCION	RTL, *gate level*	*Gate level, layout*	*Layout*
DESCRIPCION	VHDL, *Verilog*	*Netlist*[13]	Descripción de transistores
PORTABILIDAD	A todas las	Limitada a tecnologías	Optimizada a una

[13] ***Netlist:*** Representación en lenguaje de descripción de hardware de la conectividad de un circuito

TIPO	SOFT CORE	FIRM CORE	HARD CORE
	tecnologías	probadas	tecnología especifica
FLEXIBILIDAD	Alta	Limitada	Muy poca
PREVISIBILIDAD	Baja	Buena	Alta y definida por la tecnología
PROTECCION PROPIEDAD INTELECTUAL	Difícil	Fácil	Fácil

Tabla. 2.1. Resumen de las Características de *IP Cores*

A continuación se observa el diseño de un SoC (Figura. 2.9) desarrollado en un FPGA *Virtex* de *Xilinx*. Tiene un procesador *PowerPC* que es un *Hard Core*, mientras que el resto de módulos que componen este diseño son *Soft Cores*. La figura muestra que un diseño puede contener ambos tipos de *IP Cores* dentro de un mismo SoC.

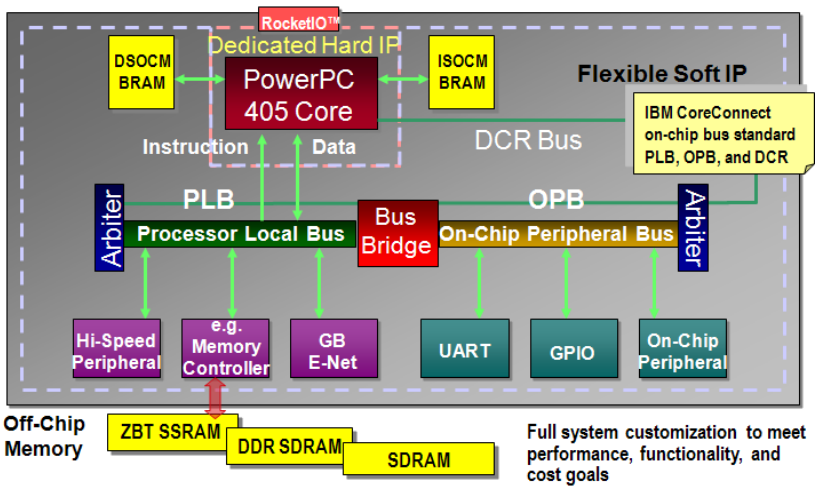

Figura. 2.9. Ejemplo de Combinación Hard IP y Soft IP en un SoC diseñado en la Tarjeta Virtex
Fuente: XILINX, Inc, 2007

2.1.3 ARQUITECTURA SYSTEM ON CHIP

Una arquitectura SoC es un conjunto organizado de elementos capaces de compartir información entre ellos. Esta arquitectura debe permitir a los fabricantes de *IP Cores* integrar sus diseños a un sistema, garantizando su funcionamiento y la comunicación con otros *IP Cores* (Figura. 2.11).

.

"Las actuales arquitecturas SoC, se basan principalmente en estándares como AMBA de ARM, *CoreConnect* de IBM, y el tradicional bus maestro-esclavo"[14]. Estos estándares ajustan los diferentes componentes, a través de buses específicos de procesador, buses del sistema de alta velocidad, y buses periféricos de alta velocidad.

Martin 2006, define al estándar AMBA como dos arquitecturas de bus segmentadas, una de alta velocidad utilizada para el sistema y la otra para periféricos de propósito general, conectadas mediante un *bridge* (puente). Por otro lado, el estándar IBM[15] *CoreConnect*, utilizado por Xilinx, implementa un *Processor Local Bus* (PLB) para conectar el CPU a los periféricos, y un *Local Memory Bus* (LMB) para conectarlo a las memorias del sistema. Cabe mencionar que actualmente las comunicaciones arquitecturales NoC (Network-On-Chip), han tomado gran importancia en el diseño de sistemas embebidos, y se basan en la implementación de redes de comunicación para *IP Cores* dentro de un FPGA.

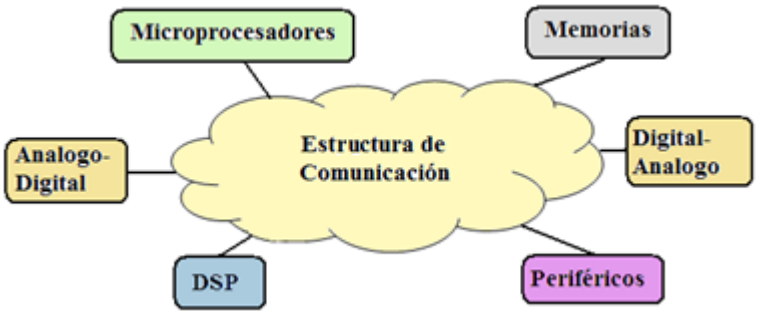

Figura. 2.10. Estructura de la Comunicación en un System on Chip
Basado en: SANDER, ingo, 2006

Como se mencionó anteriormente, tanto el estándar AMBA como el *CoreConnect* utilizan topologías tipo bus. Lo que significa que los *IP Cores* comparten una misma línea y protocolo de comunicación. La ventaja de utilizar esta topología es que si falla un elemento no genera el fallo de todo el sistema (Figura. 2.11).

[14] MARTIN, grant y CHANG, henry, *Winning the SoC Revolution - Experiences in Real Design*, Kluwer Academic Publisher, Estados Unidos 2003, 311 páginas
[15] IBM: *International Businness Machines Corporation,1999*

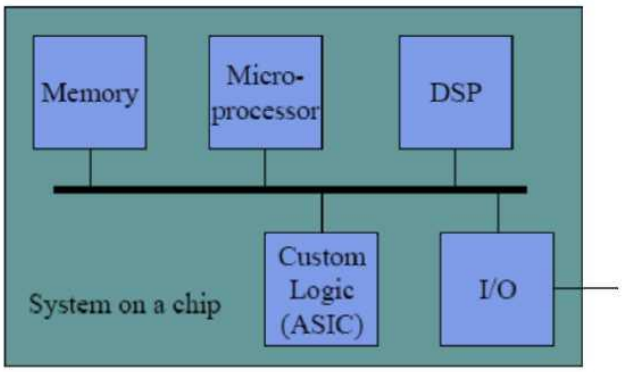

Figura. 2.11. SoC basado en topología BUS

Fuente: SANDER, ingo, 2006

2.1.4 PROCESO DE DISEÑO DE UN SYSTEM ON CHIP

Un proceso óptimo de diseño de un SoC debe cumplir los siguientes requerimientos del mercado:

- Menos tiempo de diseño.
- Que el diseño mejore la esperanza de vida de un producto.
- Buen diseño inicial, ya que un error en la implementación significaría grandes pérdidas de dinero o la muerte del producto,
- Que los grandes diseños se integren en un solo chip.
- Desarrollar paralelamente *hardware* y software[16].

La mayoría de SoCs son desarrollados a partir de *IP Cores* y sus controladores de software, que proporcionan las instrucciones para su manejo. Los *IP Cores*, se colocan sobre el FPGA de la manera más óptima, compactando en el espacio disponible la mayor cantidad posible de componentes. Para esto, se utilizan herramientas CAD que permitan elaborar un diseño de la arquitectura [12]. A su vez, el software se implementa sobre la arquitectura diseñada usando herramientas de desarrollo como IDE[17].

Cabe indicar que un paso importante en la construcción de un SoC es la emulación.

[16] ALEXANDROV, oleg, *System-on-a-Chip*, 2010 [12] - MARTIN, grant y CHANG, henry, Surviving the SoC Revolution - A Guide to Platform – Based Design

[17] **IDE (Entorno de desarrollo integrado o *Integrated Development Environment*).** Conjunto de herramientas de programación (editor de código, compilador, depurador e Interfaz Gráfica de Usuario.

El *hardware* se mapea en una plataforma de emulación basada en FPGA tal y como será fabricado, y el *software* es cargado en la memoria volátil del emulador.

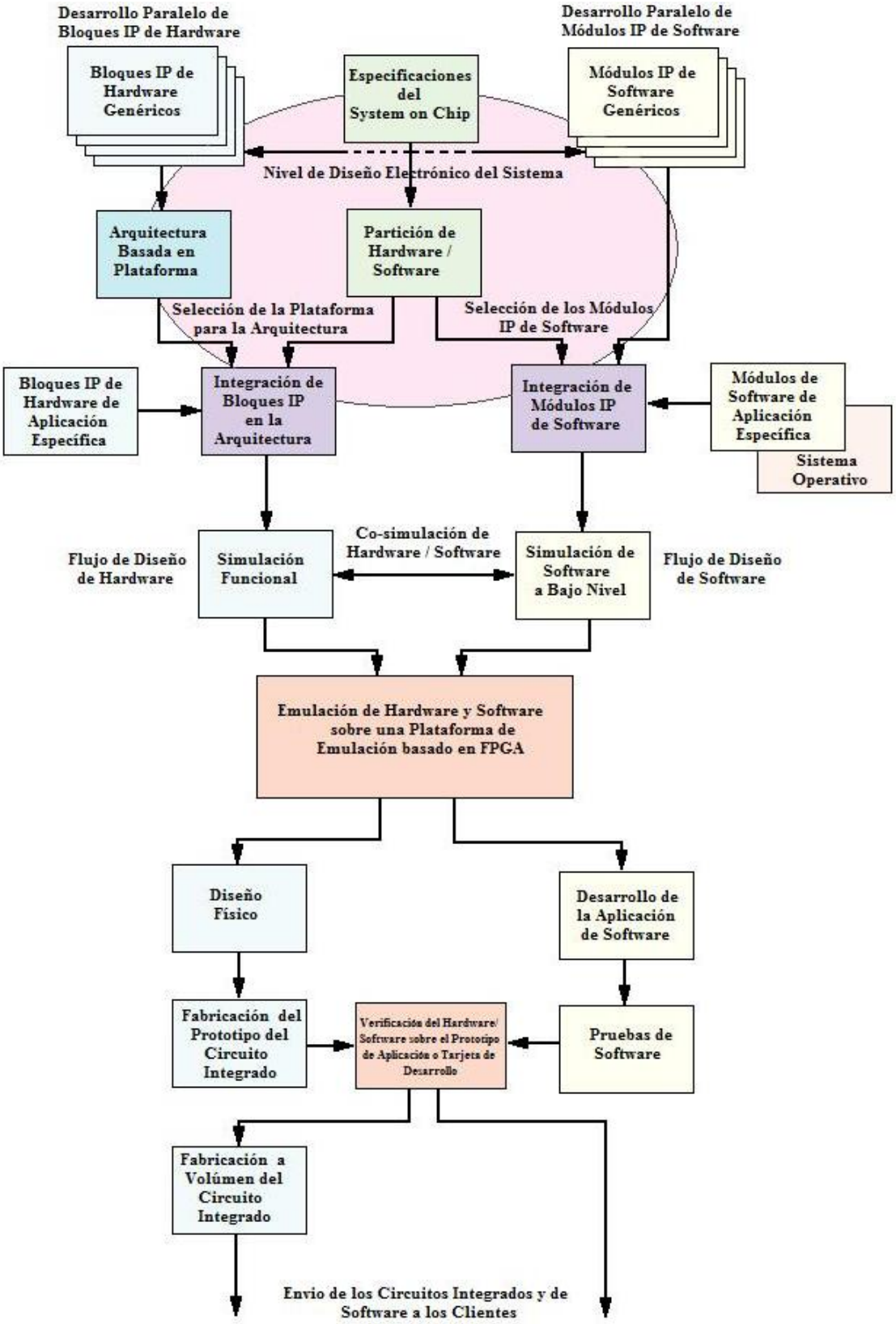

Figura. 2.12. Proceso de Diseño de System on Chip
Basado en: STANNERED, 2007

En este punto, la plataforma es puesta en funcionamiento y reproduce fielmente el comportamiento del futuro SoC. Este procedimiento se realiza con el fin de testear y depurar tanto *hardware* como *software*, bajo condiciones estrictas y a la máxima velocidad de trabajo. La emulación va generalmente precedida de la simulación tanto de *hardware* como de *software*, proceso conocido con el nombre de ***Co-Simulación***.

A continuación de una emulación satisfactoria del *hardware* del SoC, se procede a la fase de posicionamiento y ruteo (*Place and Route* o P&R) de la circuitería, con el fin de obtener un prototipo del diseño para fabricación en serie. Previo a la implementación definitiva, los prototipos son testeados y verificados para realizar posibles correcciones lógicas. Esta tarea se denomina ***verificación funcional***, y garantiza un correcto funcionamiento, adecuado tiempo de operación y consumo de energía (Figura. 2.12) [12].

2.1.5 CO - DISEÑO DE HARDWARE Y SOFTWARE

En el flujo o proceso de diseño convencional, grupos independientes de expertos diseñan el *hardware* y el *software* de un *chip*, sin que exista necesariamente cooperación entre ellos (Figura. 2.13). Por ejemplo, si se diseña un sistema con un *hardware* predefinido y construido, como un microcontrolador (PIC, ATMEL, etc.), el grupo de diseño de *software* trabajará bajo las limitaciones del *hardware* adquirido. Lo que implica, que no se podrá realizar cambios en el *hardware* perdiendo oportunidades de optimizar el sistema.

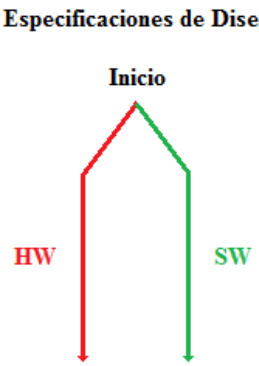

Figura. 2.13. Flujo de Diseño Tradicional

Sin embargo en el diseño de SoCs se plantea un nuevo concepto, llamado "Co - Diseño", en el cual el chip es diseñado por grupos de expertos en cooperación (Figura. 2.15).

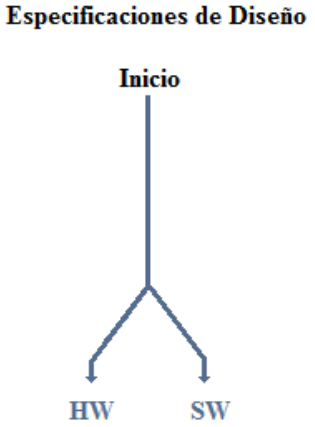

Figura. 2.14. Flujo del Co-Diseño

En el Co–diseño, el *hardware* y el *software* de un sistema embebido se desarrollan en paralelo, realizando constantes realimentaciones entre los equipos de diseño. El resultado es que cada parte puede tomar ventaja de lo que la otra puede hacer. El componente de *software* puede aprovechar las características especiales de *hardware* para obtener un mayor rendimiento. En tanto que, el componente de *hardware* puede simplificar el diseño de un módulo, si su funcionalidad puede lograrse por *software*. De esta manera se reduce la complejidad y el costo del sistema diseñado. "A menudo, los defectos de diseño, tanto en el *hardware* y como en el *software*, son descubiertos en esta estrecha colaboración"[18].

2.1.5.1 Fases del Co-Diseño HW/SW

Las fases principales del Co- diseño propuestas por Martin y Chang en el libro *Surviving the SoC Revolution* de 1999, se detallan a continuación (Figura. 2.15):

[18] LI, Qing, *Real-Time Concepts for Embedded Systems*, CMP Books, 2003, 294 páginas.

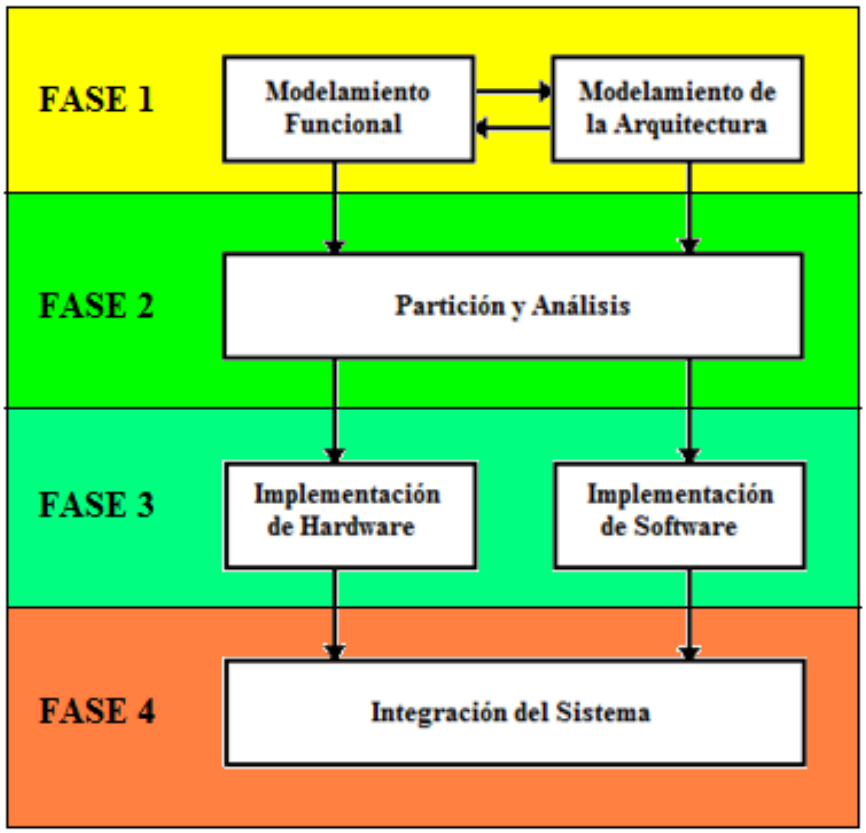

Figura. 2.15. Fases del Co-Diseño HW/SW

Basado en: MARTIN y CHANG 1999

FASE 1

***Modelamiento Funcional*.-** En esta fase se establece los requerimientos del producto, y se verifica las especificaciones del funcionamiento del sistema.

***Modelamiento de la Arquitectura*.-** Una vez que las especificaciones funcionales están definidas se procede a escoger una arquitectura que ejecute las funciones del sistema. Generalmente esta arquitectura queda defina por la plataforma que se vaya a emplear.

FASE 2

***Partición y Análisis*.-** En esta fase se realiza una partición del modelo funcional sobre el modelo de arquitectura. Se asigna las tareas del sistema a un recurso específico de *hardware*, como bloques de aceleración dedicada (decodificadores de Video y Audio), o a un recurso de *software*, como rutinas ejecutadas sobre un procesador de propósito general o especializado.

FASE 3

Implementación de Hardware.- Esta fase abarca el diseño de nuevos bloques de *hardware* y la integración de bloques reusables o IP Cores. Finaliza con la síntesis del código VHDL de la plataforma de *hardware* resultante.

Implementación de Software.- En esta fase se realiza la programación de la aplicación de software a través de un IDE, utilizando los drivers y librerías necesarios. Finaliza con la compilación del código del programa, dado en lenguajes como C o C++, y su almacenamiento en el núcleo del procesador.

FASE 4

Integración del Sistema.- Con el *hardware y software* desarrollados, se procede a ensamblar el sistema completo para la realización de pruebas de laboratorio. La implementación del producto puede incluir emuladores y prototipos rápidos para verificar las funciones de *hardware* y *software*.

El concepto de Co-Diseño de *hardware* y software enfatiza una de las características fundamentales de los sistemas embebidos, que es su personalización para cada aplicación específica. Puesto que los diseños finales pueden llegar a ser tan variados como los productos en sí. Sin embargo, para lograr que estos diseños sean económicamente fiables y cumplan con las limitaciones de tiempo, se los ha desarrollado uniendo el concepto de Co-diseño y la tecnología SoC.

2.1.5.2 Flujo de Co-Diseño de un SoC

Para comenzar el Co-Diseño de un SoC, se debe escoger una arquitectura. Esta arquitectura debe cumplir con los requerimientos del sistema a diseñar y depende de los recursos de propiedad intelectual disponibles. Después de verificar que la arquitectura escogida sea capaz de cumplir con las especificaciones del diseño, se realiza la partición del sistema. Luego, se comprueba a través de estimadores que el resultado previsto cumpla con las especificaciones del diseño. En caso de cumplir se procede al desarrollo de

hardware y *software*, caso contrario se realiza correcciones en la partición y se vuelve a estimar Figura. 2.16.

El *hardware* del sistema, se describe mediante lenguajes como VHDL[19] o *Verilog*[20] . Por otro lado el *software* es desarrollado en lenguajes de programación como C o C++. Estos procesos se llevan a cabo dentro de una estructura de cooperación y constantes realimentaciones.

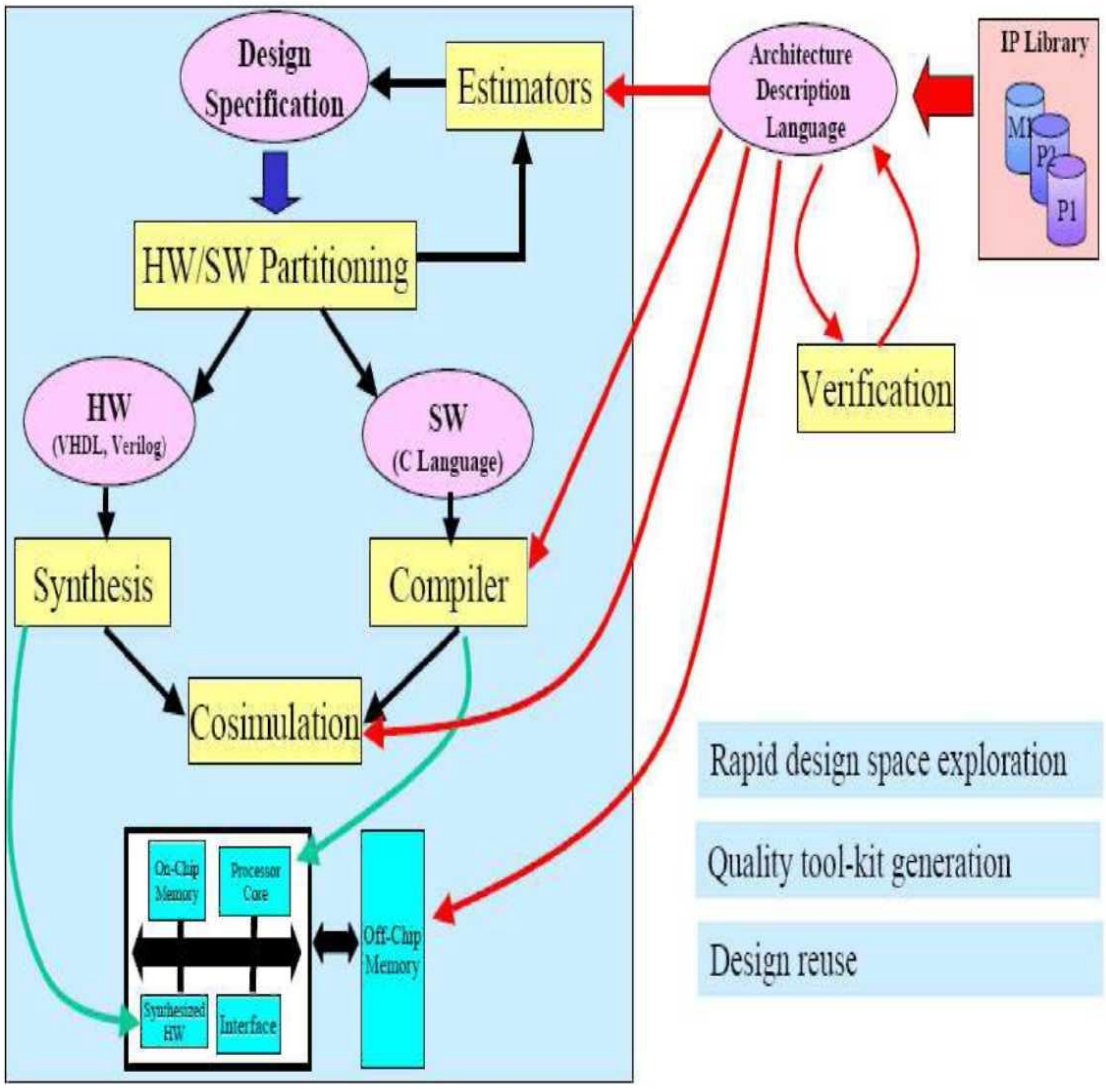

Figura. 2.16. Flujo del Co-Diseño HW/SW de un SoC

Fuente: ZHENGM, li-rong, 2007

[19] VHDL: HDL significa *Hardware Description Language*, y V es de VHSIC que es una abreviatura de muy alta velocidad de circuitos integrados

[20] *Verilog* se destina a ser utilizado para la verificación a través de la simulación, para la prueba de análisis y para la síntesis lógica.

Tras la síntesis del *hardware* y la compilación del *software* se realiza una co-simulación del sistema donde se corrigen posibles errores. Finalmente se realiza la descargar de la síntesis y el programa compilado a la plataforma de emulación (Figura. 2.16).

Cabe recalcar que en *hardware* las sentencias se ejecutan de manera concurrente (paralelo), es decir varios procesos se ejecutan simultáneamente. Mientras que las instrucciones de *software* se ejecutan de manera secuencial, es decir línea por línea. Razón por la cual, una función implementada en *hardware* será más rápida que aquella implementada por *software*.

2.1.6 METODOLOGÍAS DE DISEÑO

"La transición del diseño basado en transistor, al diseño basado en compuertas marcó el comienzo de los ASICs, lo que ocasionó un gran crecimiento de la productividad"[21]. Este hecho impulsó la reestructuración de las organizaciones de ingeniería, creando nuevas industrias, alterando la relación entre el diseñador y el diseño, e introduciendo un nuevo nivel de abstracción [23]. Las metodologías de diseño primarias se dividen en tres segmentos (Figura. 2.17):

- 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 [23]. 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, que se basa en gran medida en la experiencia adquirida con BBD (Figura. 2.17).

[21] MARTIN, grant y CHANG, henry, *Surviving the SoC Revolution - A Guide to Platform – Based Design,* Kluwer Academic Publisher, Estados Unidos 1999, 235 páginas.

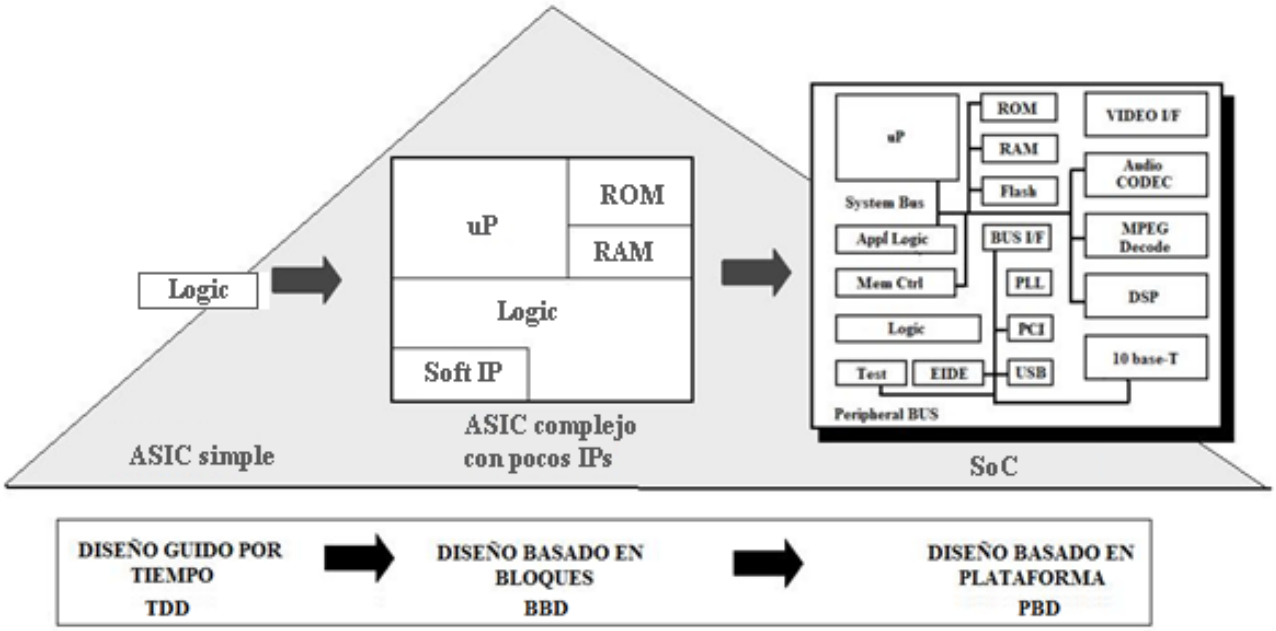

Figura. 2.17. Metodologías de Diseño Primarias

Basado en: MARTIN, grant y CHANG, henry, 1999

La Tabla 2.2 resume las características de diseño de las tres metodologías antes citadas.

CARACTERÍSTICAS DE DISEÑO	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 VC (Componentes Virtuales)
EQUIPO DE DISEÑO	Pequeño, enfocado	Multidisciplinario	Multigrupo y multidisciplinario
DISEÑO PRIMARIO	Lógica Personalizada	Bloques en Contextos, interfaces personalizadas	Interconexión con el sistema y el Bus
REUTILIZACION DEL DISEÑO	NINGUNO	Oportunista, firme y persistente	Planificada y persistente
ENFOQUE DE OPTIMIZACION PRIMARIA	Síntesis, arquitectura gate-level	Floor planning, arquitectura en bloques	Arquitectura del Sistema Silicon-compatible
COMPONENTE BASE DEL DISEÑO	Compuertas y memorias	Grupos Funcionales, núcleos	Componentes Virtuales (VCs)
ARQUITECTURA DEL BUS	Ninguna/personalizada	Personalizada	Estandarizada y de aplicación especifica

CARACTERÍSTICAS DE DISEÑO	TDD	BBD	PBD
ARQUITECTURA DE PRUEBA	Ninguna / Escaneo	Escaneo /JTAG/BIST/personalizada	Jerárquica/Escaneo Paralelo/JTAG/BIST/personalizada
SEÑAL MIXTA	Ninguna	Análoga/Digital, PLLs	Funciones, Interfaces
LIMITACIONES / ESPECIFICACIONES META	Lógica Limitada	Bloque presupuestado a limitaciones	Limitaciones de Interface
NIVEL DE VERIFICACION	RTL/compuerta	Bus funcional a ciclo exacto/RTL/compuerta	Mezcla (ISS a RTL con *hardware* y software)
HARDWARE/SOFTWARE CO-VERIFICACION	Ninguno	Funcionalidades de *Hardware*/Software e interfaces	Interfaces de *Hardware*/Software solamente
ENFOQUE EN EL PARTICIONAMIENTO	limitaciones de Síntesis (Jerarquía)	Funciones (Jerarquía)	Funciones/Comunicaciones (Jerarquía)
PLACEMENT	Completamente	Jerárquico	Jerárquico
ROUTINNG	Completamente	Completamente	Jerárquico
ANALISIS DE TIEMPO	Completamente	Completamente con limitaciones de Jerarquía	Jerárquico
CALCULO DE RETARDO	Completamente	Completamente	Jerárquico
VERIFICACION FISICA	Completamente	Completamente con limitaciones de Jerarquía	Jerárquico

Tabla. 2.2. Resumen de la Características de Diseño

Fuente: MARTIN, grant y CHANG, henry, 1999

En resumen, en la metodología TDD no existe reutilización, y sus principales inconvenientes son las limitaciones del tiempo de diseño y de la tecnología de fabricación, que en esos momentos alcanzaba pocos micrones. Por otro lado en la metodología BBD, se comienza a utilizar bloques de funciones lógicas preestablecidas (*IP Cores*), los equipos de diseño estaban orientados a los ASIC, y el interés se enfocaba en los FPGA. Esto fue el inicio del concepto PBD.

2.1.6.1 Diseño Basado en Plataforma (PBD, *Platform Based Design*)

Según el grupo de trabajo VSIA, una plataforma es "una gestión integrada, con un conjunto de características comunes, en el que un grupo o familia de productos se pueden construir. Una plataforma es un componente virtual (VC)"[22]. En definitiva, la plataforma es "un conjunto de equipos y *software* básico sobre el cual va a funcionar uno o varios sistemas a diseñar"[23] (Figura. 2.18).

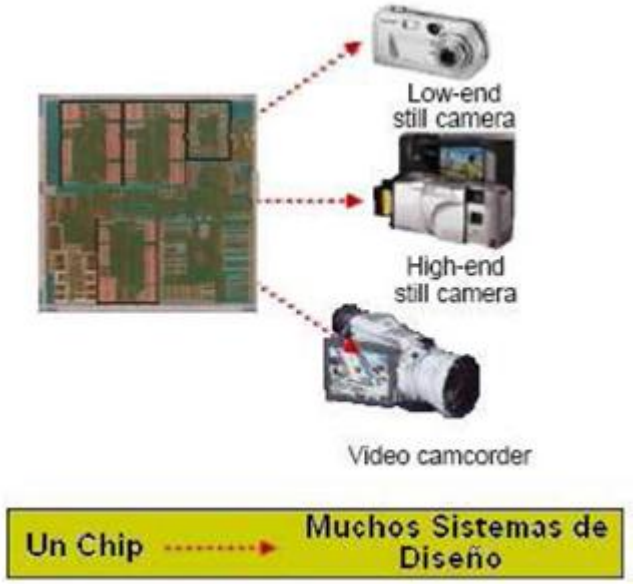

Figura. 2.18. Ejemplo de Plataforma para diferentes equipos de electrónica de consumo

Fuente: LEIBSON, 2004

El diseño basado en Plataforma (PBD), incluye la totalidad de capacidades de las metodologías TDD y BBD. PBD. Permite disminuir el *time to market* expandiendo las oportunidades y la velocidad de distribución de sus productos derivados. Además, reduce varios riesgos involucrados en el diseño, facilitando la verificación de un SoC complejo debido a la gran reutilización de combinaciones de *IP Cores*. Es decir, en lugar de mirar a la reutilización de *IP Cores* bloque por bloque, el diseño basado en plataforma agrega la reutilización de grupos de *IP Cores* en una arquitectura [23]. La idea principal de la plataforma es simplificar el proceso de diseño. Un ejemplo de esto es el *MicroBlaze Processor Subsystem*, que se explicará en el Capitulo 3.

[22] MARTIN, grant y CHANG, henry, *Winning the SoC Revolution - Experiences in Real Design*, Kluwer Academic Publisher, Estados Unidos 2003, 311 páginas

[23] Universidad de Concepción, *Plataforma*, 2010

La industria ha tomado el diseño basado en plataforma para muchos proyectos de SoC, por varias razones. Según Martin y Chang, las más importantes son:

- Es el siguiente paso lógico en la reutilización de *IP Cores*.
- Los resultados de un diseño se obtienen rápido, permitiendo la optimización y adaptación de productos a necesidades muy específicas, lo que ayuda a la personalización de los mismos.
- Las plataformas capturan y reutilizan las mejores arquitecturas y enfoques de diseño establecidos, para determinados tipos de productos.

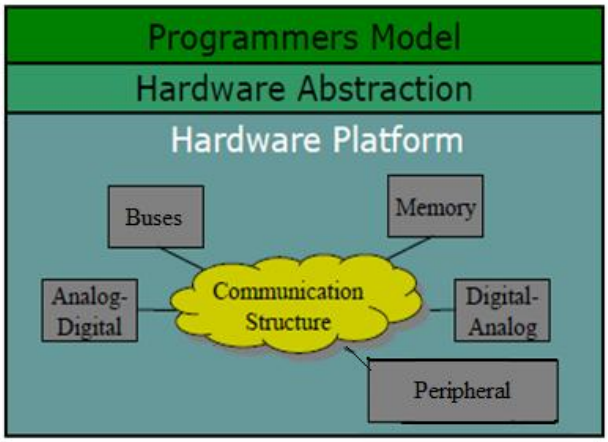

Figura. 2.19. Plataforma de *Hardware*

Fuente: SANDER, ingo, 2006

En un diseño PBD se realiza una abstracción de la plataforma de *Hardware* y en la capa superior se implementa el modelo de programación de acuerdo a las especificaciones del diseño. Este concepto de capas permite cambiar la arquitectura física del SoC sin afectar la aplicación, y añadir nuevos servicios en la parte superior de una arquitectura existente. Los cambios que se realicen en una capa "afectan solamente a la capa misma y sus interfaces"[24]. Las plataformas pueden ser consideradas en su más abstracto sentido como "una familia coordinada de arquitecturas de *hardware-software*, que satisfacen un conjunto de limitaciones arquitecturales, que son impuestas para permitir la reutilización de componentes de *hardware y software*"[25].

[24] SANDER, ingo, *SoC Architectures*, 2006 [13]

[25] MARTIN, grant y CHANG, henry, *Winning the SoC Revolution - Experiences in Real Design*, Kluwer Academic Publisher, Estados Unidos 2003, 311 páginas

2.2 SISTEMAS EMBEBIDOS

Como definición general, un "sistema embebido es un sistema computacional con un alto grado de integración de *hardware* y *software*, diseñado para desempeñar una función específica"[26]. Los Sistemas Embebidos pueden encontrarse en:

- Electrónica de consumo: Cámaras digitales, celulares, reproductores mp3.
- En el hogar: Televisores, microondas, sistemas de cine en casa.
- En el trabajo: Copiadoras, sistemas de seguridad, switches, routers.
- Equipos médicos: Marcapasos, desfibriladores, electrocardiógrafos.
- Aeronáutica y aeroespacial: Satélites, sistemas de guía, vehículos espaciales.
- Industria: Robots industriales, osciloscopios, multímetros, etc.

Figura. 2.20. Sistemas Embebidos en el entorno del Hogar

[26] LI, Qing, *Real-Time Concepts for Embedded Systems*, CMP Books, 2003, 294 páginas.

2.2.1 CARACTERÍSTICAS DE LOS SISTEMAS EMBEBIDOS

Algunas características distintivas de un sistema embebido son:

- Están dedicados a tareas específicas
- Tienen restricciones de tiempo real
- Concurrencia de procesos
- Bajo Consumo de energía
- Bajo Precio
- Bajo Peso
- Pequeñas Dimensiones
- Generalmente emplean un Sistema Operativo de Tiempo Real (RTOS) [21]

Adicionalmente, los sistemas embebidos no siempre trabajan en condiciones óptimas de temperatura, humedad, limpieza, etc., por lo que su robustez debe adaptarse a sus condiciones de trabajo. Por otro lado, usan diversos dispositivos para interactuar con su entorno, como son:

- Convertidores A/D (Análogo/Digital) y D/A (Digital/Análogo)
- PWM (Modulación por Ancho de Pulso)
- Entradas y salidas digitales para sensores, actuadores, periféricos especiales, etc.

Los SoC son muy usados en la implementación de sistemas embebidos, debido a que integran gran cantidad de componentes en su interior. Esto permite construir una gran variedad de aplicaciones embebidas, sin la necesidad de añadir dispositivos periféricos externos, reduciendo el costo total, el tamaño del producto final, y su tiempo de fabricación [20].

2.2.2 SISTEMAS EMBEBIDOS DE TIEMPO REAL

Los sistemas embebidos de tiempo real tienen la principal característica de responder a eventos externos de una manera oportuna y garantizada. Los acontecimientos externos pueden ser de carácter sincrónico o asincrónico (Figura. 2.21). Las limitaciones en el tiempo de respuesta a estos acontecimientos incluyen [20]:

- Reconocer cuando ocurre un evento.
- Realizar las operaciones requeridas como resultado del evento.
- Llegar a los resultados necesarios dentro de un límite de tiempo determinado.

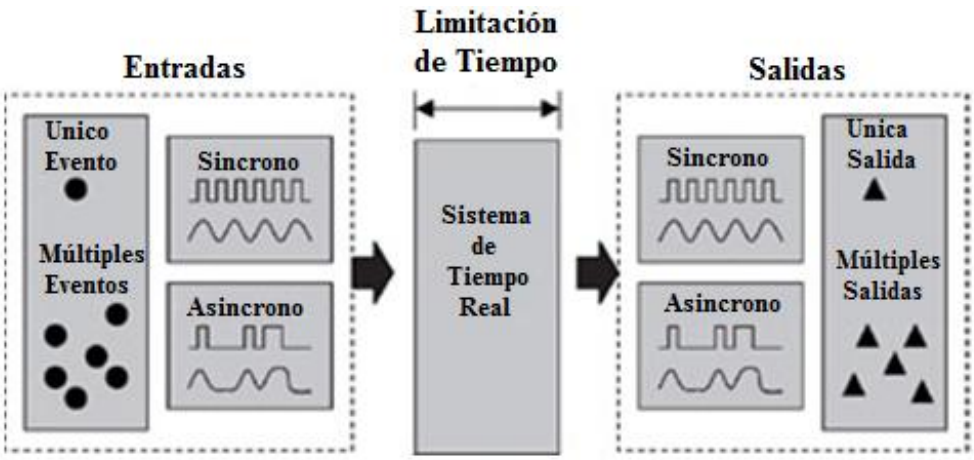

Figura. 2.21. Vista Simple de un Sistema de Tiempo Real

Basado en: Li, qing, 2003

No todos los sistemas embebidos presentan comportamientos en tiempo real, ni todos los sistemas en tiempo real son embebidos. Sin embargo, los dos sistemas no son mutuamente excluyentes, y la zona en que se superponen crea la combinación de los sistemas conocidos como sistemas embebidos en tiempo real [23] (Figura. 2.22).

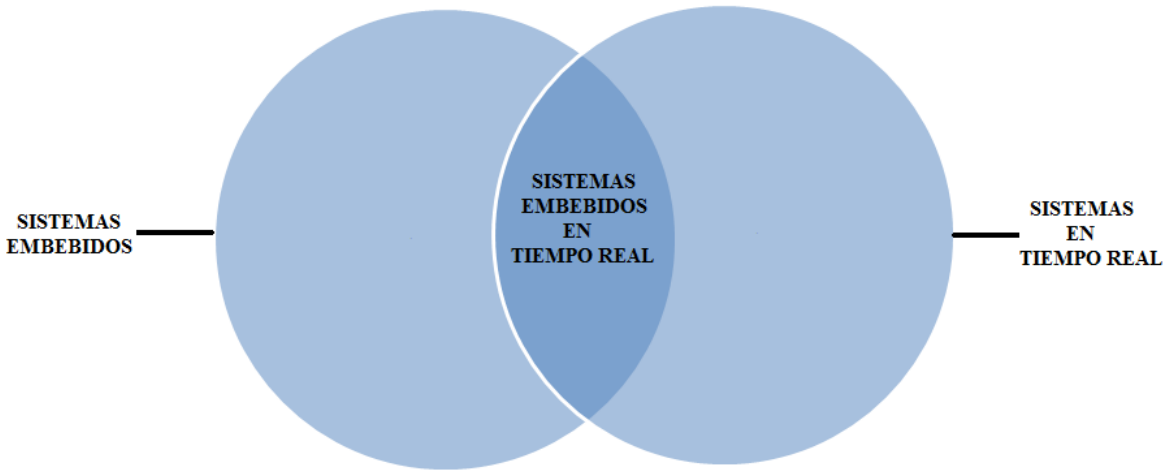

Figura. 2.22. Sistemas Embebidos en Tiempo Real

Basado en: LI, qing, 2003

Como se mencionó anteriormente, muchos sistemas cumplen con limitaciones de tiempo y plazos bien determinados en la ejecución de tareas y obtención de resultados. Los

sistemas se clasifican en dos tipos de acuerdo al grado de tolerancia al incumplimiento de plazos:

- Sistemas *Hard* de Tiempo Real
- Sistemas *Soft* de Tiempo Real

2.2.2.1 Sistemas *Hard* de Tiempo Real

Un Sistema *Hard* de Tiempo Real debe cumplir sus plazos con un grado casi nulo de tolerancia. Los plazos no cumplidos producen catástrofes, y su coste puede implicar incluso vidas humanas. Los resultados de los cálculos obtenidos después del plazo establecido, tienen prácticamente un nivel de utilidad cero [20].

Las armas de defensa, los sistemas de guía para misiles y los controladores de vuelo constituyen Sistemas *Hard* de Tiempo Real. Ya que por ejemplo, si se desea cambiar el rumbo de la trayectoria de un misil y el tiempo de respuesta del sistema no es determinista, los posibles retardos no deseados causarían daños y pérdidas incalculables, al impactar el misil en lugares no previstos [20].

2.2.2.2 Sistemas *Soft* de Tiempo Real

Un Sistema *Soft* de Tiempo Real debe cumplir sus plazos con un grado aceptable de tolerancia. Los plazos pueden contener diversos niveles de tolerancia, plazos promedio de tiempo, e inclusive una distribución estadística de tiempos de respuesta con diferentes grados de aceptabilidad. En estos sistemas un plazo vencido, no resulta en el fallo del sistema, pero dependiendo de la aplicación puede implicar costes que aumentan en proporción a la demora [20].

Los reproductores de DVD son Sistemas *Soft* de Tiempo Real. Esto se debe a que, decodifican audio y video mientras responden a los comandos de los usuarios en tiempo real. Sin embargo, si el usuario envía una serie de comandos muy rápido y causa que el decodificador pierda información, el retardo no ocasiona más que una visible distorsión momentánea. El reproductor de DVD continúa funcionando después de un momento y los datos ingresados siguen siendo útiles, por lo es un sistema con un nivel alto de tolerancia [20].

2.3 SISTEMA OPERATIVO EN TIEMPO REAL (RTOS)

"Un RTOS[27], es un programa que realiza la ejecución de rutinas de *software* en forma oportuna, administra los recursos del sistema, y proporciona una base coherente para el desarrollo del código de una aplicación"[28]. Contiene una combinación de varios módulos que pueden incluir: un *kernel*, un sistema de archivos, protocolos de red, soporte POSIX[29], *drivers* y otros componentes necesarios para una aplicación específica (Figura. 2.23).

2.3.1 KERNEL

La parte principal de un RTOS, es el *kernel* o núcleo del Sistema Operativo. Este proporciona la lógica básica, la programación, y los algoritmos para la gestión de los recursos del sistema (centro de la Figura. 2.24).

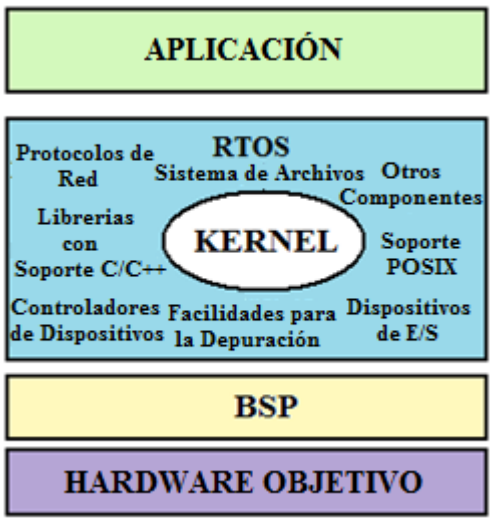

Figura. 2.23. Vista de alto nivel de un RTOS, su núcleo, y otros componentes que se encuentran en sistemas embebidos

Basado en: LI, Qing, 2003

Según el autor Li Qing, en su libro *Real-Time Concepts for Embedded Systems*, 2003, el *kernel* de un RTOS debe contener los siguientes componentes (Figura. 2.24):

[27] RTOS: *Real-Time-Operating-System*

[28] LI, Qing, *Real-Time Concepts for Embedded Systems*, CMP Books, 2003, 294 páginas.

[29] POSIX (Portable Operating System Interface): Estándar que persigue generalizar las interface de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas.

Scheduler

El *scheduler* contiene un conjunto de algoritmos que determina cuando se ejecuta cada tarea o hilo. Algunos ejemplos comunes de "*schedulling algorithms*" incluyen *round-robin*, donde procesos cíclicos se ejecutan turnándose una y otra vez, o *Preemptive Scheduling*, en el cual el hilo con la prioridad más alta, continua ejecutándose hasta que finaliza, entra en espera, o es sustituido por un hilo de mayor prioridad.

Objetos

Son estructuras especiales del *kernel* que ayudan a los desarrolladores a crear aplicaciones para sistemas embebidos en tiempo real, como: hilos, semáforos y *messages queue*[30], o identificadores de interrupciones.

Servicios

Son operaciones que el núcleo realiza sobre un objeto, o en general las operaciones tales como *timing*, manejo de interrupciones, y gestión de recursos.

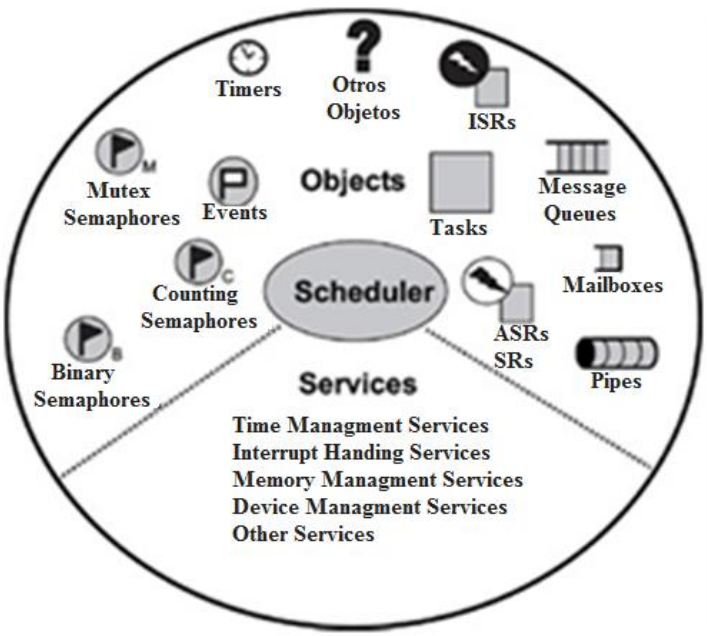

Figura. 2.24. Componentes comunes en un *kernel* de RTOS.

Basado en: LI, qing, 2003

[30] ***Messages queue:*** Colas de mensajes

En un sistema embebido, un RTOS proporciona una plataforma de *software* estable sobre la cual se puede ejecutar aplicaciones. Sin embargo, no todos los sistemas embebidos están diseñados con un RTOS. Los sistemas que requieren *hardware* relativamente sencillo, o una pequeña cantidad de código de aplicación no exigen un RTOS. Mientras que, aquellos con aplicaciones de *software* relativamente complejas si lo requieren.

2.3.3 RTOS COMERCIALES

En el mercado hay abundantes RTOS, la mayoría de ellos proveen mecanismos adecuados para habilitar el desarrollo de sistemas de tiempo real, algunos ejemplos son *Tornado/VxWorks, Lynx, OSE, QNX, RT-Linux, y ThreadX, Petalinux, Standalone, Xilkernel*. Estos dos últimos vienen incluidos en las herramientas de Xilinx como se verá en el siguiente capítulo. En la Tabla. 2.3 se puede encontrar un listado de las compañías que producen RTOS y los procesadores que los soportan.

Compañía	Producto	Procesador
ENEA Embedded Technology	OSE	PowerPC® 405
eSOL Co., Ltd	PrKernel (µITRON4.0)	PowerPC 405 / MicroBlaze
Express Logic	ThreadX®	PowerPC 405, 440 / MicroBlaze
Green Hills Software	Integrity®	PowerPC 405, 440
LynuxWorks	BlueCat Linux	PowerPC 405, 440
LynuxWorks	LynuxOS	PowerPC 405
Mentor Graphics ESD	Nucleus Plus	PowerPC 405, 440 / Microblaze
Micriµm	µC/OS-II	PowerPC 405 / MicroBlaze
MiSPO	NORTi/ulTRON	PowerPC 405 / MicroBlaze
MontaVista Software	MontaVista Linux	PowerPC 405, 440
PetaLogix	uClinux and Petalinux 2.6	MicroBlaze
QNX	Neutrino®	PowerPC 405
Wind River Systems	VxWorks®	PowerPC 405, 440
Wind River Systems	Wind River GPP Linux	PowerPC 405, 440
Timesys	LinuxLink	PowerPC 405, 440

Tabla. 2.3. RTOS Disponibles en el mercado

CAPÍTULO 3

PLATAFORMA DE DESARROLLO XILINX SPARTAN-6 FPGA EMBEDDED KIT

3.1 INTRODUCCIÓN

El Xilinx Spartan-6 FPGA Embedded Kit es un conjunto de varios elementos entre los que destaca la tarjeta de desarrollo SP605. Esta tarjeta permite a los diseñadores de *hardware* y *software* emular sus diseños sobre el FPGA Spartan 6 LX45T. Este FPGA está construido con tecnología de 45 nanómetros, 9 capas de metal, y tecnología de proceso *oxido-dual*. Cabe mencionar, que los FPGA Spartan 6 se comercializaron en 2009 como una solución de bajo costo para la automoción, soluciones inalámbricas, pantallas planas y aplicaciones de video vigilancia.

El Xilinx Spartan-6 FPGA Embedded Kit (Figura. 3.1) contiene las herramientas y los *IP Cores* necesarios para disminuir el tiempo de desarrollo de un SoC. El kit completo incluye:

- Tarjeta de Evaluación SP605 basada en el FPGA Spartan-6 LX45T(XC6SLX45T-3FGG484), acompañada de:
    - Fuente de Alimentación
    - 2 cables USB Tipo-A a Mini-B de 5 pines (para la descarga de Hardware y la depuración de software)
    - 1 Cable de Ethernet
    - 1Adaptador VGA a DVI
    - 1 Compact Flash Card – 2GB (con demos de Sistemas Embebidos para emplearlos con el Kit)

- DVD con Software Xilinx ISE® *Design Suite* (IDS) version 12.1
- Licencia para ISE *Design Suite Embedded Edition* que incluye:
    - Embedded Development Kit (EDK)
        - Xilinx Platform Studio (XPS)
        - Software Development Kit (SDK)
    - ISE Design Suite Logic Edition (Device-locked to Spartan-6 LX45T)
        - ISE Foundation™ con Simulador ISE (ISim)
        - PlanAhead™ Herramienta de Análisis y Diseño
        - Analizador ChipScope™ Pro logic y Toolkit Serial I/O
- Documentación - Impresa:
    - SP605 - Guía de Hardware Setup
    - UG727 - Introducción con el Kit Embebido Spartan-6.
- Memoria de Almacenamiento USB contiene:
    - Documentación:
        - DS757 - SP605 Hoja Técnica de MicroBlaze™ Processor Sub-System
        - UG728 - SP605 Tutorial de Hardware
        - UG729 - SP605 Tutorial de Software
    - Diseño de Referencia – *MicroBlaze Processor Subsystem*
    - Demos con Spartan-6 Embedded Kit
    - Driver para la comunicación serial y Herramientas Adicionales.

Figura. 3.1. Xilinx SPARTAN-6 FPGA EMBEDDED KIT

Fuente: XILINX, Inc., 2011

3.2 DESCRIPCIÓN DE LA PLATAFORMA DE HARDWARE

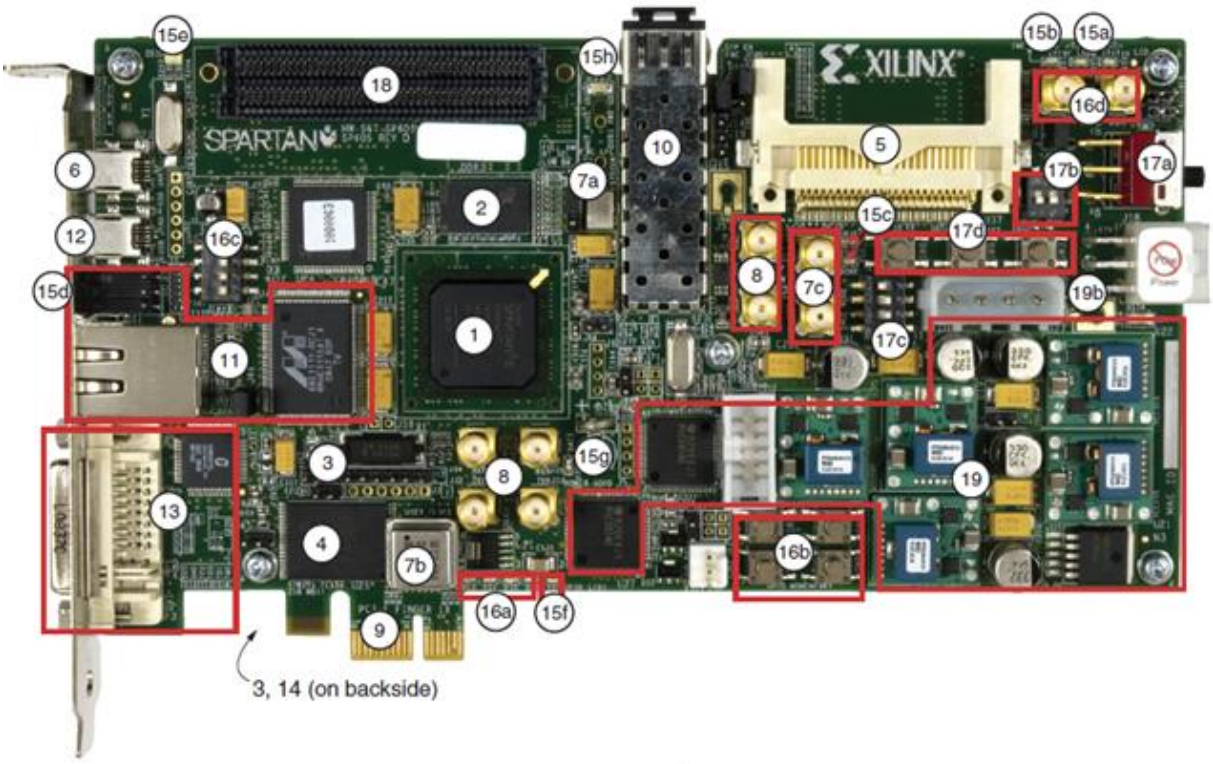

Figura. 3.2. Características de la Tarjeta

Fuente: XILINX, Inc., 2010

3.2.1 COMPONENTES DE LA TARJETA SP605

Los componentes listados a continuación corresponden a aquellos mostrados en la Figura. 3.2. Se sugiere dirigirse al glosario de términos en caso de que alguna sigla no resulte familiar:

1. Spartan-6 FPGA - XC6SLX45T-3FGG484 FPGA
2. Componente de Memoria DDR3
3. Cabecera SPI Ext. x4 - SPI Flash x4 (ubicado en la parte trasera de la tarjeta SP605)
4. Linear BPI Flash x16
5. Zócalo de System ACE CompactFlash
6. Conector USB JTAG (USB Mini-B)
7. Clock Generation
    a. Oscilador de 200 MHz.
    b. Zócalo de Oscilador, singleended – LVCMOS
    c. Conectores SMA
8. GTP puerto SMA x4 y MGT Reloj SMA (REFCLK)
9. Conector PCIe (Gen 1)
10. Módulo SFP Cage/Connector
11. Ethernet 10/100/1000
12. USB UART (Puente USB-to-UART)
13. Conector DVI Codec y Video
14. IIC EEPROM (ubicado en la parte trasera de la tarjeta SP605)
15. LEDs de estado
    a. FMC Power Good
    b. Estado de System ACE CF
    c. FPGA Inicialización y Realizado
    d. Estado de Ethernet PHY
    e. Estado de JTAG USB
    f. FPGA Despierto
    g. TI Power Good
    h. MGT AVCC, DDR3 Term Power Good
16.

a. LEDs (4) GPIO - LEDs Rojos - (activación en Alto)
b. *Pushbuttons* (4) GPIO - (activación en Alto)
c. *DIP Switch* (4-polos) - (activación en Alto)
d. SMA (2)

17. Interruptores de Energía, Configuración, y *Pushbuttons*
    a. SP605 Power On-Off - Interruptor de Deslice
    b. Modos del FPGA - DIP Switch
    c. Configuración del System ACE CF - DIP Switch
    d. FPGA PROG, CPU Reset, and System ACE CF Reset Pushbutton.
18. Conector FMC LPC - Samtec ASP-134603-01
19.
    a. Controlador Power Management
    b. Mini-Fit Type 6-Pin, ATX Type 4-pin – conectores de entrada de energía de 12 V

3.3 DESCRIPCIÓN DE LA PLATAFORMA DE SOFTWARE

Esta plataforma está constituida por el *ISE Design Suite Embedded Edition* 12.1. Este *software* proporciona herramientas para el diseño embebido y una serie de *IP Cores* adaptados a las necesidades comunes de los desarrolladores. Una de las herramientas principales de *ISE Desing Suite Embedded Edition* es el *Embedded Development Kit* (EDK).

Es importante conocer la manera correcta de instalar *ISE Design Suite* 12.1, razón por la cual en el Anexo A1, se detalla paso a paso su instalación y licenciamiento para que permanezca activado en su computador, y no presente ningún problema posterior.

3.3.1 EMBEDDED DEVELOPMENT KIT

El *Embedded Development Kit* (EDK) de Xilinx es un ambiente integrado para el diseño de sistemas embebidos. Este conjunto preconfigurado incluye *Xilinx Platform Studio* (XPS), para el diseño de *hardware,* y *Software Development Kit* (SDK) para el diseño de *software*. Así como, toda la documentación e la mayoría de *IP Cores* que se podrían necesitar en el diseño de SoCs con procesadores *PowerPC* y/o *MicroBlaze*.

Cabe mencionar que el documento UG683 *EDK, Concepts, tools, and Techniques* de Xilinx, proporciona la base para la definición de los conceptos que se detallan a continuación.

3.3.1.1 XILINX PLATFORM STUDIO (XPS)

Al utilizar XPS se puede crear sistemas embebidos procesados altamente personalizables, e integrar estos diseños dentro de un FPGA. XPS está compuesto de una vista grafica de diseño y varios asistentes que guían a los usuarios a través de los pasos necesarios para crear sistemas basados en procesador.

XPS posee las siguientes características:

- Permite al usuario rápidamente personalizar y configurar detalles del diseño en el *System Assembly View* (SAV).
- Consta de un amplio catálogo de *IP Cores* basados en bus PLB.
- Presenta cuadros de diálogo de configuraciones de *IP Cores* .
- Está estrechamente integrado con *ISE Project Navigator, ISIM, y ChipScope*.
- Admite un asistente para la creación e importación de *IP Cores*, el cual automatiza la creación de plantillas personalizadas, y provee mecanismos para importar *IP Cores* creados por el usuario dentro del XPS.
- Se puede exportar el proyecto de *Hardware* al *Software Development Kit* (SDK).
- Permite la creación automática de reportes del diseño.

Áreas de la Ventana Principal del XPS

1. Área de Información del Proyecto (*Project Information Area*)

   a) Ventana Project
   b) Ventana Application
   c) Ventana IP Catalog

2. Área de desarrollo de proyecto

d) Panel de Conexiones (Conectivity Panel)

e) Expansión de conexiones y buses asociados a los IPs (*View Buttons*)

f) Filtros de Panel (*Filters Panel*)

3. Área de Depuración del Proyecto

g) Ventana de Consola

h) Ventana de Advertencias (Warnings)

i) Ventana de Errores

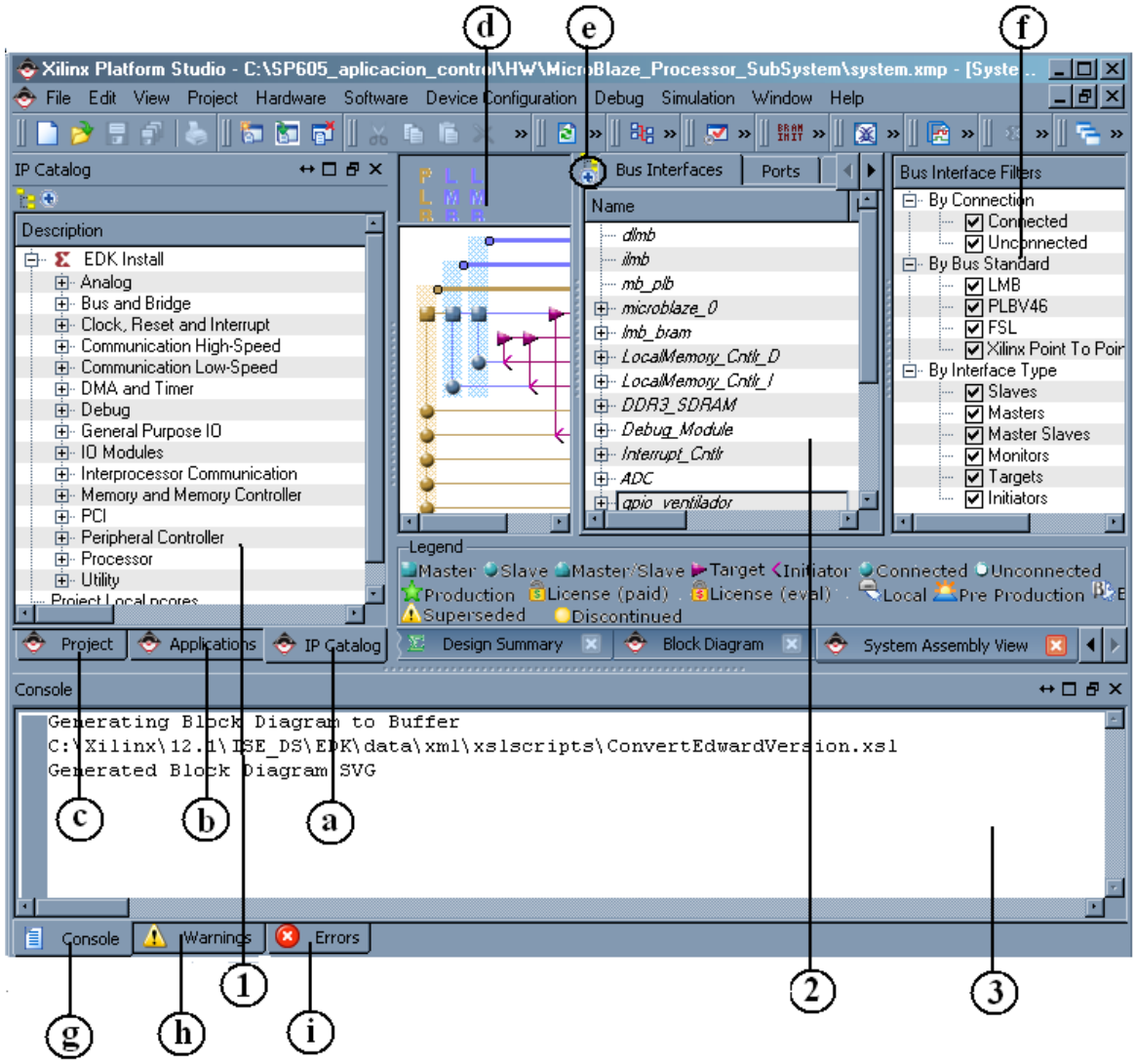

Figura. 3.3. Vista general del XPS

1. Área de Información del Proyecto (*Project Information Area*)

Esta área contiene la información necesaria del proyecto que se está realizando, incluye las pestañas: *Project, Applications,* e *IP Catalog.*

a) Ventana Project

Como se muestra en la Figura. 3.4, en esta sección se encuentra una lista de los archivos principales del proyecto.

La información esta agrupada en las siguientes categorías generales:

- Archivos del Proyecto (*Project Files*)
  Contiene los archivos: *Microprocessor Hardware Specification (MHS), Microprocessor Software Specification (MSS),* User Constraints File (UCF), *IMPACT Command, Implementation Option, y Bitgen Option.*

  Archivo MHS (*system.mhs*): Este archivo, conforma la base de *hardware*. Contiene la descripción de los elementos del sistema y sus parámetros de configuración. Además indica la conexión entre *IP Cores* y buses, todo en formato de texto.

  Archivo MSS (*system.mss*): Este archivo contiene una descripción referente a varios parámetros de *software* asociados con los periféricos, y constituye la base del proyecto de *software*.

  Los archivos MHS y MSS son los productos principales del diseño en XPS. El sistema completo de *hardware* y *software* es representado por estos dos archivos.

  ***User Constraints File* (*system.ucf*)**: En un archivo UCF, el diseñador decide a que pines del FPGA se asocian las conexiones externas del diseño en XPS.

Nota: Cuando se edite un archivo UCF, se deberá volver a sintetizar el sistema para que los cambios se vean reflejados [45].

Archivo XMP (*system.xmp*): El archivo *Xilinx Microprocessor Project (XMP)* es leído por el XPS para mostrar gráficamente su contenido en la interfaz de usuario. Toda la información del proyecto se guarda en este archivo y se lo encuentra en la carpeta principal del proyecto de *hardware*.

- Opciones del Proyecto (*Project Option*)
  Contiene opciones específicas del proyecto, tales como *Device, Netlist, Implementation, Hardware Description Language (HDL), y Sim Model.*

- Resumen del Diseño (*Design Summary*)
  El resumen del diseño es una representación gráfica de su diseño embebido y contiene tablas con los resultados finales.

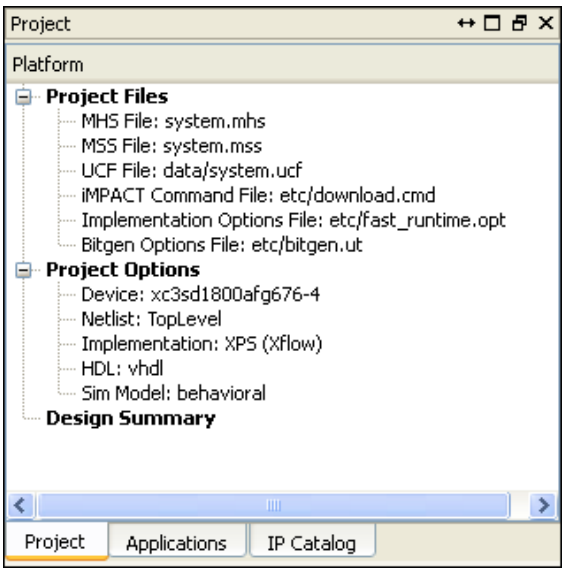

Figura. 3.4. *Project Information Area*, Ventana *Project*

b) Ventana *Aplication*

Enlista las opciones de configuración de las aplicaciones de *software*, sus archivos de cabecera, y archivos fuente, asociados con cada aplicación (Figura. 3.5). En esta opción se puede:

- Crear y añadir aplicaciones de *software*, construir proyectos, y cargarlos a la memoria RAM.
- Establecer opciones de compilación.
- Añadir archivos fuente y de cabecera al proyecto.

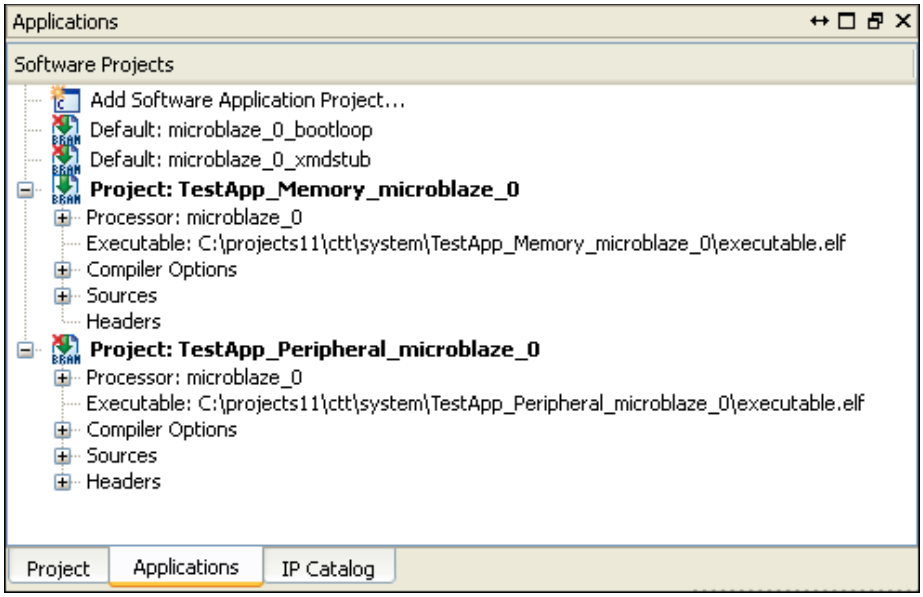

Figura. 3.5. ***Project Information Area*****, Pestaña Aplicaciones**

Nota: Es posible crear y gestionar proyectos de *software* en el XPS; sin embargo, se recomienda el uso de la herramienta SDK para desarrollar *software*.

c) Ventana de Catálogos de IP Cores (*IP Catalog Tab*)

Muestra la lista de *IP Cores*, que contiene el XPS, así como cierta información básica, que incluye:

- Nombre de los IP *Cores* y estado de su licenciamiento (no licenciado, bloqueado, no bloqueado), además consta el grupo al que pertenece cada uno de ellos.
- La versión de lanzamiento y su estado (activo, o en desuso)
- Los procesadores que soporta cada *IP Core*.
- Su clasificación, sea esta bus, periférico, o procesador

Cada *IP Core*, incluye su hoja técnica, y el archivo *Microprocessor Peripheral Description* (MPD). Solo basta con dar click derecho sobre el *IP Core*, y seleccionar la opción que se desea visualizar. A continuación se enlista todos los *IP Cores* que se puede encontrar en el IP *Catalog* de *ISE Design Suite* 12.1

GRUPO	NOMBRE	VERSION DE IP CORE	TIPO DE IP CORE	PROCESADOR QUE SOPORTA	CLASIFICACION DE IP CORE
ANALOG	XPS Delta-Sigma Analog to Digital Converter	1.01.a	xps_deltasigma_adc	PowerPC, MicroBlaze	Periférico
	XPS Delta-Sigma Digital to Analog Converter	1.01.a	xps_deltasigma_dac	PowerPC, MicroBlaze	Periférico
BUS AND BRIDGE	Fast Simple Link (FSL) Bus	2.11.c	fsl_v20	MicroBlaze	Bus
	Local Memory Bus (LMB) 1.0	1.00.a	lmb_v10	MicroBlaze	Bus
	Processor Local Bus (PLB) 4.6	1.04.a	plb_v46	PowerPC	Bus
	PLBV46 to PLBV46 Bridge	1.02.b	plbv46_plbv46_bridge	PowerPC, MicroBlaze	Periférico
CLOCK, RESET AND INTERRUPT	Clock Generator	4.00.a	clock_generator	PowerPC, MicroBlaze	Periférico
	Phase Locked Loop	2.00.a	pll_module	PowerPC	Periférico
	Processor System Reset Module	2.00.a	proc_sys_reset	PowerPC, MicroBlaze	Periférico
	XPS Interrupt Controller	2.01.a	xps_intc	PowerPC, MicroBlaze	Periférico
COMUNICATION HIGH – SPEED	Ethernet PHY MII to Reduced MII	1.01.a	mii_to_rmii	PowerPC, MicroBlaze	Periférico
	XPS CAN Controller	3.01.a	xps_can	PowerPC, MicroBlaze	Periférico
	XPS 10/100 Ethernet MAC Lite	4.00.a	xps_ethernetlite	PowerPC, MicroBlaze	Periférico
	XPS LocalLink FIFO	1.02.a	xps_II_fifo	PowerPC, MicroBlaze	Periférico
	XPS LocalLink Tri-mode Ethernet MAC	2.03.a	xps_II_temac	PowerPC, MicroBlaze	Periférico
	XPS MOST	1.01.a	xps_most_nic	PowerPC, MicroBlaze	Periférico
	XPS USB2 Peripheral	3.00.a	xps_usb2_device	PowerPC, MicroBlaze	Periférico
	XPS USB Host	1.01.a	xps_usb_host	PowerPC, MicroBlaze	Periférico
COMUNICATION	XPS IIC Interface	2.03.a	xps_iic	PowerPC, MicroBlaze	Periférico
	XPC PS2 Interface	1.01.b	xps_ps2	PowerPC, MicroBlaze	Periférico

GRUPO	NOMBRE	VERSION DE IP CORE	TIPO DE IP CORE	PROCESADOR QUE SOPORTA	CLASIFICACION DE IP CORE
LOW – SPEED	XPS SPI Interface	2.01.b	xps_spi	PowerPC, MicroBlaze	Periférico
	XPS UART (16550-style)	3.00.a	xps_uart16550	PowerPC, MicroBlaze	Periférico
	XPS UART (Lite)	1.01.a	xps_uartlite	PowerPC, MicroBlaze	Periférico
DMA AND TIMER	Fixed Interval Timer	1.01.b	fit_timer	PowerPC, MicroBlaze	Periférico
	XPS Central DMA Controller	2.01.c	xps_central_dma	PowerPC, MicroBlaze	Periférico
	XPS Watchdog Timer	1.01.a	xps_timebase_wdt	PowerPC, MicroBlaze	Periférico
	XPS Timer/Counter	1.02.a	xps_timer	PowerPC, MicroBlaze	Periférico
DEBUG	Chipscope Integrated Controller	1.04.a	chipscope_icon	PowerPC, MicroBlaze	Periférico
	Chipscope Integrated Logic Analyzer (ILA)	1.03.a	chipscope_ila	PowerPC, MicroBlaze	Periférico
	Chipscope PLBv46 Integrated Bus Analyzer (IBA)	1.03.a	chipscope_plbv46_iba	PowerPC	Periférico
	Chipscope Virtual IO (VIO)	1.03.a	chipscope_vio	PowerPC, MicroBlaze	Periférico
	MicroBlaze Debug Module (MDM)	1.00.g	mdm	MicroBlaze	Periférico
GENERAL PURPOSE IO	XPS General Purpose IO	2.00.a	xps_gpio	PowerPC, MicroBlaze	Periférico
IO MODULES	XPS TFT	2.01.a	xps_tft	PowerPC, MicroBlaze	Periférico
INTERPROCESSOR COMUNICATION	XPS Mailbox	2.00.b	xps_mailbox	PowerPC, MicroBlaze	Periférico
	XPS Mutex	1.00.d	xps_mutex	PowerPC, MicroBlaze	Periférico
MEMORY AND MEMORY CONTROLLER	Block RAM (BRAM) Block	1.00.a	bram_block	PowerPC, MicroBlaze	Periférico
	LMB BRAM Controller	2.10.b	lmb_bram_if_cntlr	MicroBlaze	Periférico
	Multi-Port Memory Controller(DDR/DDR2/SDRAM)	6.00.a	mpmc	PowerPC, MicroBlaze	Periférico
	XPS BRAM Controller	1.00.b	xps_bram_if_cntlr	PowerPC, MicroBlaze	Periférico

GRUPO	NOMBRE	VERSION DE IP CORE	TIPO DE IP CORE	PROCESADOR QUE SOPORTA	CLASIFICACION DE IP CORE
	XPS Multi-Channel External Memory Controller (SRAM/Flash)	3.01.a	xps_mch_emc	PowerPC, MicroBlaze	Periférico
	XPS System ACE Interface Controller (Compact Flash)	1.01.a	xps_sysace	PowerPC, MicroBlaze	Periférico
PCI	PLBv46 IP Interface (IPIF) to LogicCORE PCI Express Bridge	4.04.a	plbv46_pcie	PowerPC, MicroBlaze	Periférico
PERIPHERAL CONTROLLER	XPS External Peripherical Controller	1.02.a	xps_epc	PowerPC, MicroBlaze	Periférico
PROCESSOR	MicroBlaze	7.30.a	microblaze	MicroBlaze	Procesador
UTILITY	Utility Bus Split	1.00.a	util_bus_split	PowerPC, MicroBlaze	Periférico
	Differential Signaling IO Buffer	1.01.a	util_ds_buf	PowerPC, MicroBlaze	Periférico
	Utility Flip-Flop	1.10.a	util_flipflop	PowerPC, MicroBlaze	Periférico
	Utility IO Multiplexor	1.00.a	util_io_mux	PowerPC, MicroBlaze	Periférico
	Utility Reduced Logic	1.00.a	util_reduced_logic	PowerPC, MicroBlaze	Periférico
	Utility Vector Logic	1.00.a	util_vector_logic	PowerPC, MicroBlaze	Periférico

Tabla. 3.1. IP *Catalog*

MICROBLAZE SOFT PROCESSOR CORE

El *MicroBlaze* es un *Soft Core*, es decir, no viene como un diseño predefinido en la oblea de silicio del FPGA. Esta es la razón por la que se lo conoce como *Soft Processor*. Es un procesador RISC de 32 bits diseñado para trabajar con la arquitectura *Harvard* y el estándar *CoreConnect* de IBM. Gracias al *MicroBlaze Soft Processor*, se tiene gran flexibilidad para integrar gran cantidad de periféricos y de memorias, lo cual ofrece la posibilidad de realizar un sin número de diseños de sistemas embebidos sin generar gastos adicionales y obteniendo el mayor provecho de los recursos del FPGA.

Características de MicroBlaze Soft Procesador

A continuación se presenta un listado con las características más importantes de *MicroBlaze*, algunas de estas serán analizadas a continuación y las restantes podrán ser estudiadas a mayor profundidad en la hoja técnica de este *IP Core*.

- Arquitectura Pipeline
- Repertorio de Instrucciones
- Soporte de Memoria
- 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)
    - Registro de desplazamiento de *Hardware*.
    - Unidad de Gestión de Memoria (MMU)
        - Proporciona toda la funcionalidad de la MMU, con Memoria Virtual soportado por Linux 2.6
        - Permite el modo MPU (*Multiple Process Unit*) de regiones de protección para aplicaciones con RTOS seguras.
        - Controla la asignación de la dirección efectiva a la dirección física.
        - Posee 2 zonas de protección a memoria.
- Módulo para depuración.
- Controlador de Interrupciones.
- Doble *Timer / Counter* de 32 bits.
- Soporta Unidad de Punto Flotante (FPU)
- Ejecución de la aceleración de Hardware
    - *Barrel Shifter* (1 ciclo de operación)
    - División entera (32 ciclo de operación)
    - Multiplicación (1 ciclo de operación)
- 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.
    - Señales de Interrupción con activación alta o baja.
- Soporte para la lógica de depuración
    - Control de JTAG a través de un núcleo de soporte de depuración.

Arquitectura Pipeline

La ejecución de instrucciones en *MicroBlaze* es *pipelined* (segmentada), es decir, la ejecución de la mayoría de instrucciones toma un ciclo de reloj completo, pocas son las instrucciones que necesitan más de un ciclo para su ejecución.

Cuando se producen problemas relacionados con saltos de instrucciones demasiados prolongados, *pipeline* debe tratar de resolverlos concentrando mecanismos para evitar dichos inconvenientes, por lo general tratados por *hardware* en *MicroBlaze*. Cada vez que se produce un salto, el *pipeline* se vacía cuando el salto se hace efectivo.

Existe una técnica conocida con el nombre de *Delay Slots* y consiste en un cierto número de instrucciones de salto, incluidas en el repertorio, que permiten la ejecución de la instrucción que les precede, con el objetivo de reducir la penalización de vaciado.

Repertorio de Instrucciones

MicroBlaze tiene una estructura de *hardware* relativamente simple pero con una capacidad de procesamiento de datos sumamente compleja, idea principal de las arquitecturas tipo RISC [73].

En el caso de *MicroBlaze*, soporta 87 instrucciones, considerando la diferencia entre aquellas que operan con valores inmediatos y aquellas que realizan la misma operación con registros. Cabe destacar que las instrucciones que requieran un procesamiento complejo se realizarán en un *hardware* específico, utilizando otros recursos del FPGA.

Soporte de Memoria

MicroBlaze maneja los siguientes tipos de memorias:

- SDRAM - DDR3 de 128 MB, 16-bit, 333.5 MHz (667 Mb/s)
- Internal Block RAM - 32 KB
- Linear (Parallel) Flash - 32 MB
- Quad SPI Flash - 8 MB
- Compact Flash Card - 2 GB

- IIC EEPROM - 1 KB
- *Multi-Port Memory Controller* (MPMC) con un puerto disponible para la interface lógica de usuario para memoria SDRAM DDR3 externa.

ASISTENTE DE CONFIGURACIÓN DE MICROBLAZE

Xilinx ha creado el asistente de configuración *MicroBlaze* (Figura. 3.6), con un solo clic del ratón se puede seleccionar entre varias configuraciones. Este asistente proporciona información instantánea sobre la utilización de recursos.

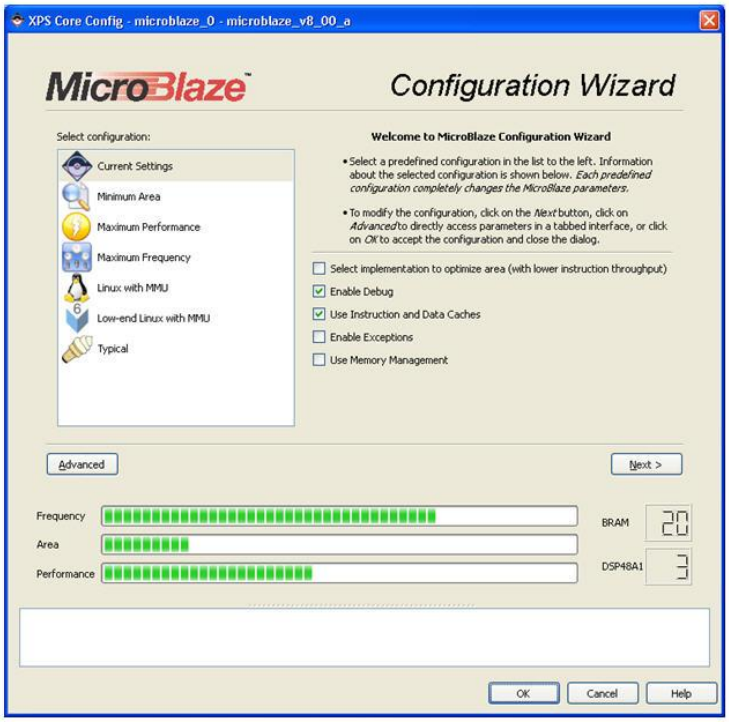

Figura. 3.6. Asistente de Configuración de MicroBlaze

BUSES DEL SISTEMA

MicroBlaze utiliza arquitectura *Harvard*, donde la memoria de datos es separada de la de instrucciones. Esta arquitectura combinada con el estándar *CoreConnect*, permiten separar las conexiones del procesador con sus memorias en un tipo de bus, y las del procesador con sus periféricos, en otro tipo de bus, disminuyendo la carga capacitiva general del sistema.

Actualmente en nuevas versiones de ISE *Design Suite*, el estándar *CoreConnect*

define tres tipos de *buses*, cada uno con diferentes características de velocidad y conectividad:

LMB: (*Local Memory Bus*): Es un bus síncrono de alta velocidad, que se lo utiliza en la arquitectura *Harvard* para la conexión directa a los bloques de memoria interna del FPGA. El bus LMB no permite la utilización de tamaños de palabras mayores o menores de 32 bits, además solo admite un maestro en su implementación y puede ser utilizado para interconectar los puertos de instrucciones y de datos del procesador *MicroBlaze* al bloque de RAM (BRAM). Este bus es compatible con el bus PLB.

Se denotan 3 características importantes:

- Eficiente, no requiere de árbitro.
- Buses de datos de lectura y escritura separados.
- Baja utilización de recursos en el FPGA. [57]

Descripción Funcional

La Figura. 3.7 muestra al procesador *MicroBlaze* utilizando dos módulos LMB v10 uno para Instrucciones I y otro para Datos D, conectados a los dos puertos del bloque de BRAM, a través de controladores de interface LMB BRAM (*Interface Controllers*).

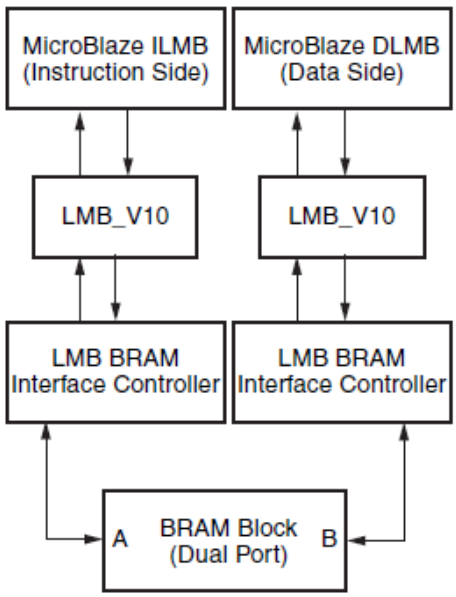

Figura. 3.7. Procesador MicroBlaze utilizando dos módulos LMB
Fuente: XILINX, Inc., 2009

PLB: (*Processor Local Bus*): El bus local de procesador de Xilinx a 128 bits, proporciona una infraestructura de bus para conectar un número opcional de maestros PLB y esclavos dentro del sistema general del 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.

Descripción Funcional

El bus PLB consiste de un árbitro central de bus, buses de control, y todas las estructuras OR/MUX necesarias. Este bus proporciona una estructura de bus PLB completa y permite conexiones directas con un número configurable de maestros y esclavos. La Figura. 3.8 aporta un ejemplo de la conexión de PLB a un sistema con tres maestros, y tres esclavos [56].

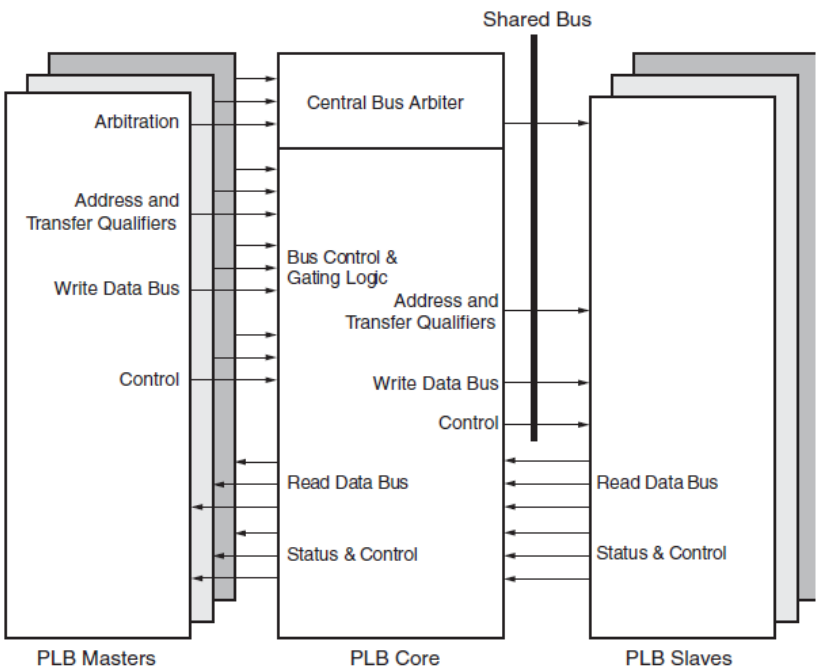

Figura. 3.8. Diagrama de Interconexión de PLB

Fuente: XILINX Inc., 2009

Diagrama de Bloques

La Figura. 3.9 proporciona una vista comprensiva acerca de los bloques que componen un bus PLB.

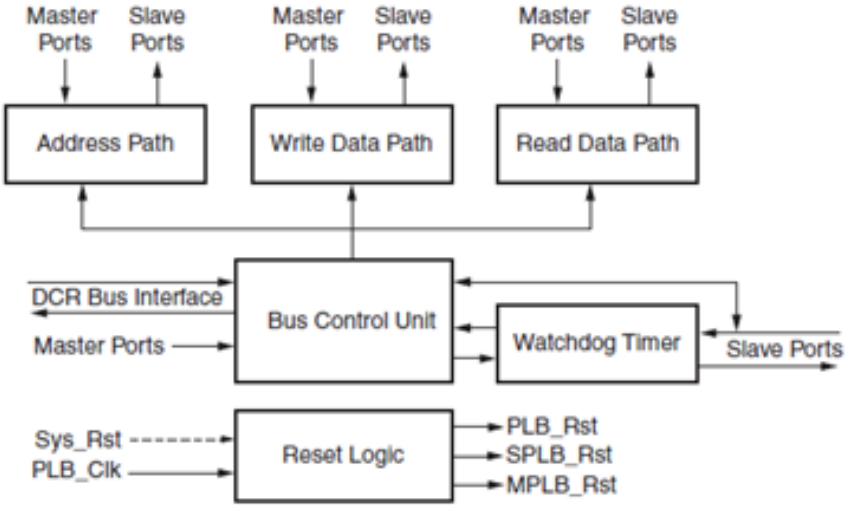

Figura. 3.9. Diagrama de Bloques de PLB

Fuente: XILINX Inc., 2009

***Address Path*:** Este bloque contiene la multiplexación necesaria para seleccionar las direcciones de maestros, que son conducidas a los dispositivos esclavos en las direcciones de salida del PLB.

***Write Data Path*:** La unidad *write data path*, contiene la lógica de direccionamiento necesaria de los datos de escritura para los maestros y esclavos.

***Read Data Path*:** La unidad *read data path*, contiene la lógica de direccionamiento necesaria de los datos de lectura para los maestros y esclavos.

Unidad de Control de Bus: Este módulo implementa un árbitro controlador de bus, que maneja las direcciones y flujo de datos hacia el PLB y DCRs, este árbitro soporta la arbitración para 16 maestros PLB.

***Watchdog Timer*:** El *watchdog timer* del PLB, es utilizado para generar las respuestas PLB_MTimeout cuando ningún esclavo responda. El *watchdog timer* está

establecido para 16 ciclos de reloj.

***Reset Logic*:** El PLB v4.6 no incluye un circuito de *reset*, se asume que el usuario colocara un *core proc_sys_reset* para asegurar el *reset* por al menos 16 ciclos de reloj.

Además de estos *buses* existe otra alternativa para comunicarse con el procesador *MicroBlaze*, se trata de un protocolo de comunicación denominado *Fast Simple Link* (FSL), que permite la comunicación con el procesador accediendo a los registros internos de este. El periférico actúa a modo de co-procesador y la conexión se realiza de manera sencilla a través de registros de desplazamiento de 32 bits. *MicroBlaze* soporta hasta 16 dispositivos conectados con este protocolo.

INTRODUCCIÓN AL USO DE IP CORES

Los *IP Cores* que a continuación se describen, son considerados de mayor interés debido a que son utilizados en la mayoría de diseños SoC. Su principal aplicación está orientada a proyectos de Automatización y Control. En este proyecto son empleados por las guías de estudio que serán analizadas más adelante.

Las definiciones de cada *IP Core*, son una recopilación de las hojas técnicas de cada uno de ellos, proporcionadas por el fabricante Xilinx. Estos *IP Cores* son:

- Utility Bus Split
- Utility Flip Flop
- XPS Delta-Sigma DAC
- XPS Delta-Sigma ADC
- XPS 16550 UART
- XPS General Purpose Input/Output (GPIO)
- XPS Timer / Counter

UTILITY BUS SPLIT

El *core Bus Split*, toma un bus de entrada y lo divide en dos buses de salida de menor tamaño y, así sirve como nexo entre periféricos (Figura. 3.10).

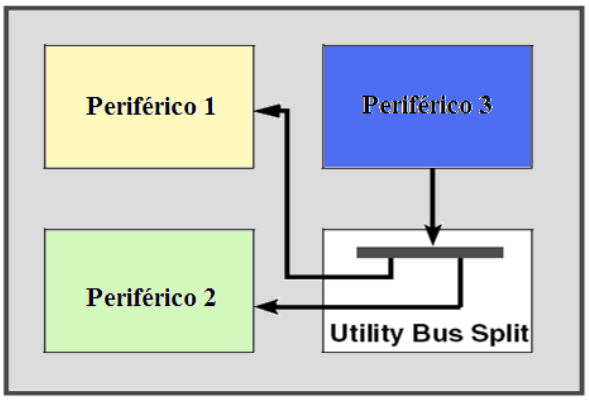

Figura. 3.10. *Bus Split* en un Sistema

Fuente: XILINX, Inc., 2009

Configuración

Se puede configurar el tamaño tanto de la entrada como de las salidas (Figura. 3.11).

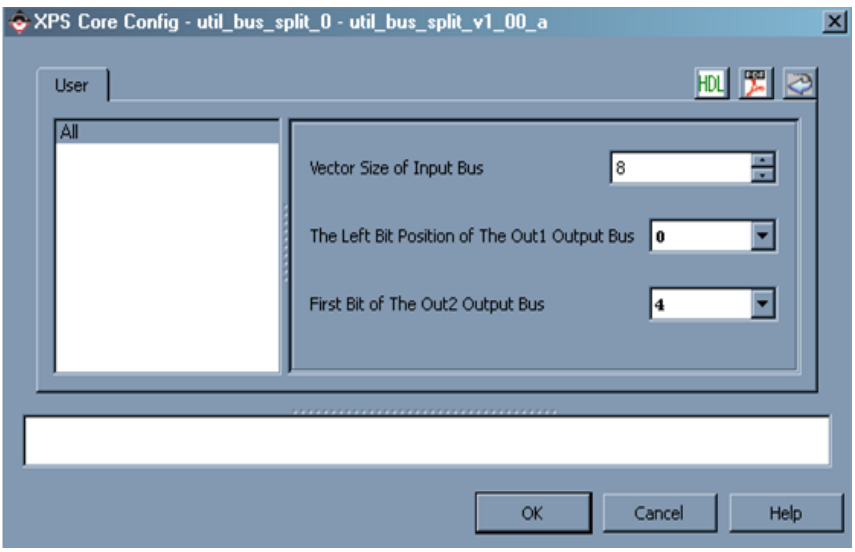

Figura. 3.11. Configuración del *Utility Bus Split*

La única restricción que tiene es que, el bus de salida 1 *Left Bit Position* producto de la división, debe de ser de menor tamaño que, el bus de salida 2 *First Bit* y ambos buses de salida deben ser de menor tamaño que, el bus de entrada. La Figura. 3.12, indica los

tamaños máximos permitidos por los buses.

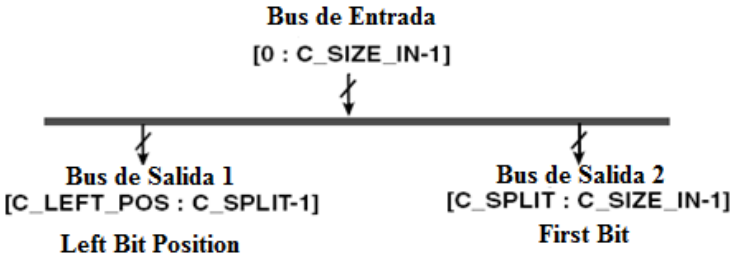

Figura. 3.12. Configuración del *Utility Bus Split*

Fuente: XILINX, Inc, 2009

Por ejemplo, si el tamaño del vector del bus de entrada es de 4 bits y en el bus de salida 2 se escoge de un tamaño 3, se deberá escoger para la salida 1 tamaño entre 0, 1, o 2 (Figura. 3.13).

C_SIZE_IN: 4

C_SPLIT: 1

C_LEFT_POS: 0

Bus de Entrada: [0 : C_SIZE_IN - 1] → [0 : 4 - 1] → [0 : 3]

Tamaño del Bus de Entrada: 4

Bus de Salida 2: [C_SPLIT : C_SIZE_IN - 1] → [1 : 4 - 1] → [1 : 3]

Tamaño del Bus de Salida 2: 3.[31]

Bus de Salida 1: [C_LEFT_POS : C_SPLIT - 1] → [0 : 1 - 1] → [0 : 0]

Tamaño del Bus de Salida 1: 1

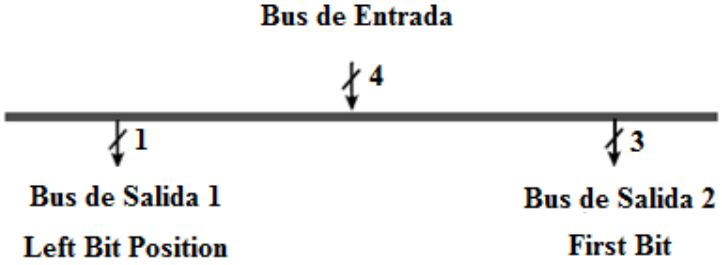

Figura. 3.13. Ejemplo de configuración del *Utility Bus Split*

[31] **Nota:** El mínimo valor que puede adquirir la salida 2 *First Bit* es de 1.

UTILITY FLIP FLOP

Los *Flip Flop´*s son elementos básicos de memoria, capaces de permanecer en uno de dos estados posibles (uno o cero) durante un tiempo indefinido en ausencia de perturbaciones. Este *core* que se encuentra en el grupo de *Utility* del *IP Catalog*, es un *Flip Flop* tipo D, el cual actúa como seguidor de señal, ya que permite mantener en la salida la misma señal provocada en la entrada. En la Figura. 3.14 se muestra la disposición de un *Utility Flip Flop* en un sistema.

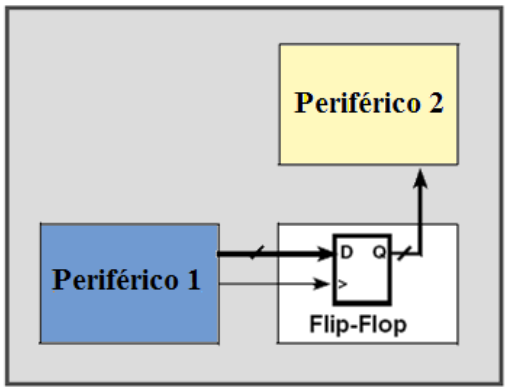

Figura. 3.14. *Utility Flip Flop* en un Sistema

Características

El *IP Core Flip Flop* tiene las siguientes características:

- El ancho del bus de entrada y salida puede ser configurable.
- Soporta *set* y *clear* sincrónico o *reset* y *preset* asincrónicos.
- Puede o no tener la entrada de *clock enable CE*.
- Valor inicial de registro programable.

Configuración

Para configurar este *core* se debe activar los casilleros que se muestran en la Figura. 3.15. Además, se debe escoger el ancho del bus que va ingresar por la entrada D del *Flip Flop* y si va a tener un valor inicial.

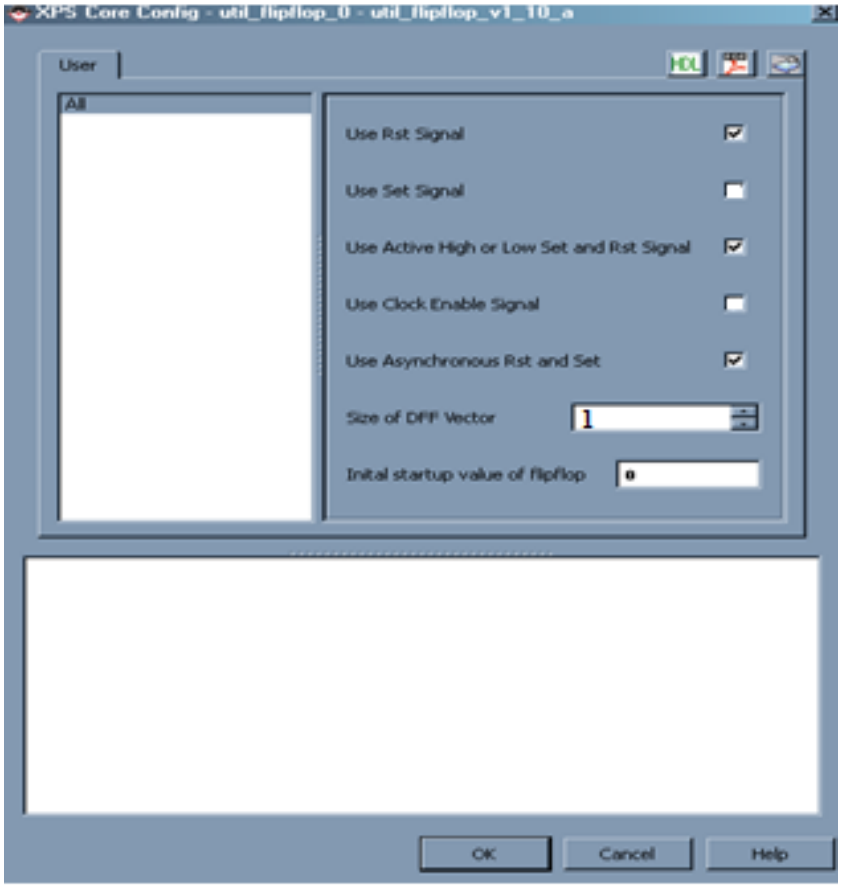

Figura. 3.15. Configuración del *Utility Flip Flop*

No se ha seleccionado la entrada CE (*Clock Enable*), ya que solo se necesitará un pulso, de duración de un flanco positivo, utilidad que se la verá más adelante en configuración del DAC (Conversor Digital – Análogo, señal Read_En). Observar la disposición de entradas y salidas del *Flip Flop* en la Figura. 3.16.

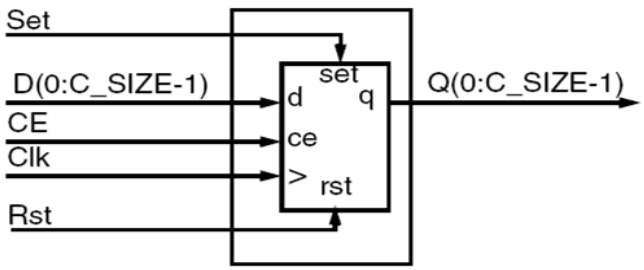

Figura. 3.16. Diagrama de Bloques del *Utility Flip Flop*

Fuente: XILINX, Inc., 2009

En resumen, si hay un 1 en la entrada D, cuando se aplica el pulso de reloj, Q toma el valor de 1 (SET) y lo almacena y si hay un 0 en la entrada D, cuando se aplica el pulso de reloj la salida toma el valor de 0 (RESET).

XPS DELTA-SIGMA DAC (CONVERSOR DIGITAL - ANALOGO)

El *IP Core XPS Delta - Sigma DAC* (Versión 1.01a), utiliza técnicas digitales para convertir un número binario en un voltaje analógico. Consecuentemente, este *IP Core* puede ser implementado en lógica programable. *Delta – Sigma DAC*, es un DAC de alta velocidad que trabaja con un solo bit. Usando una realimentación digital, se genera una cadena de pulsos, donde el promedio del ancho de ciclo de esta cadena de pulsos es proporcional al valor de la entrada binaria. La señal analógica se crea pasando la cadena de pulsos a través de un *Filtro Pasa Bajos* analógico, el cual es el único circuito externo requerido para este proceso. Este filtro está compuesto por una resistencia y un capacitor, ver la Figura. 3.17.

Una variedad de aplicaciones del uso del DAC incluyen:

- **En control y automatización**: Es la base para implementar diferentes tipos de controladores, o circuitos que utilicen técnicas adaptativas Esta información está en formato de bits “1 y 0”. Razón por lo cual se hace necesario traducir esta información digital a formato analógico, con propósitos de ilustración, seguimiento, indicación, etc.
- **En instrumentación:** De la misma forma, se lo utiliza con el objetivo de visualizar y monitorizar, el estado de las variables o alarmas que proporciona la salida analógica de un instrumento.
- **En control por computadora de procesos o en la experimentación**: Si requiere de una interface que transfiera las instrucciones digitales de la computadora al lenguaje de los actuadores del proceso que normalmente es analógico.
- Además se lo usa en varias aplicaciones como generadores de onda y fuentes de voltaje programables.

Características

Las características de este *IP Core* comprenden:

- Interface PLB
- Ancho de DAC seleccionable

- Salida a escala completa seleccionable
- Generación de interrupciones programables
- FIFO (*First Input First Output*) de datos de 16 entradas de profundidad.

Arquitectura Delta – Sigma

La Figura. 3.17, representa el diagrama de bloques de alto nivel de una implementación típica de un *Delta – Sigma DAC*. Como se indica en este diagrama las entradas incluyen las señales de *Reset* y Clock (*DAC_Clk*), además del bus de entrada del número binario (*DACin*).

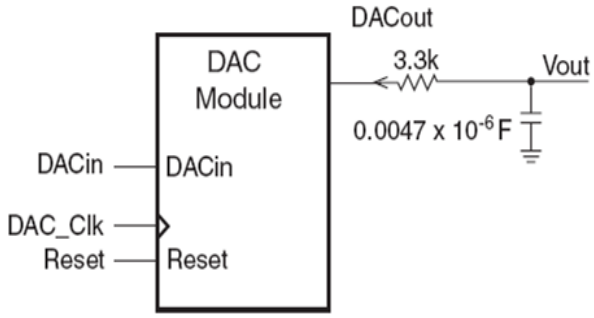

Figura. 3.17. Módulo DAC

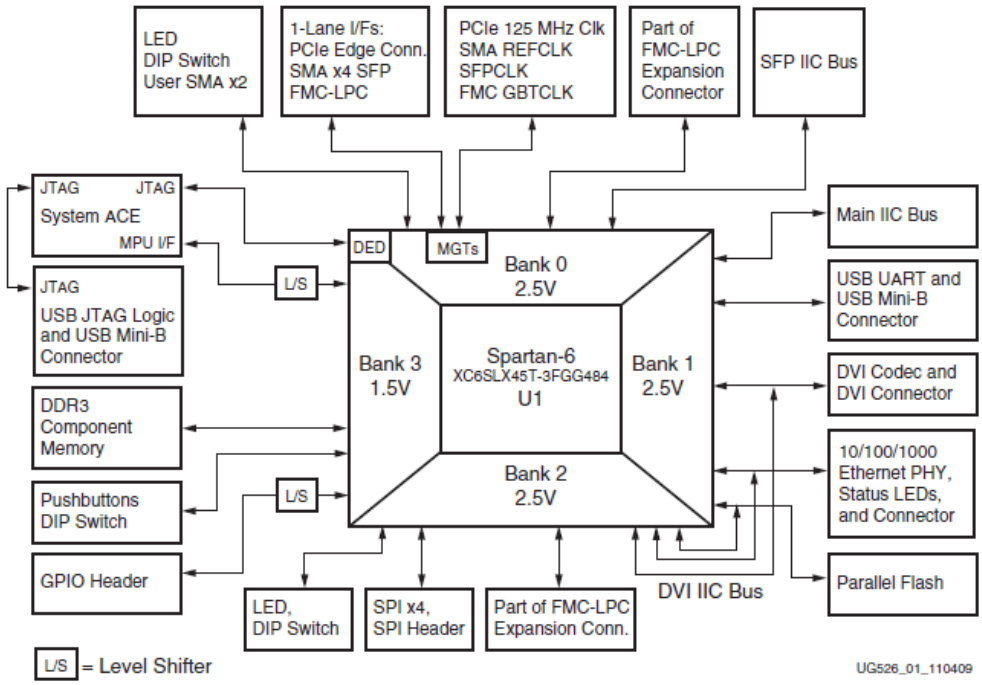

Figura. 3.18. Bancos de Voltaje del FPGA Spartan 6

Fuente: XILINX, Inc., 2010

La salida *Vout* puede ser establecida desde 0V hasta V_{CCO}, donde V_{CCO} es la tensión de voltaje aplicado a las E/S del FPGA, desde uno de sus bancos (Figura. 3.18). Este voltaje es conducido al filtro pasa bajos RC[32].

La entrada binaria del DAC en esta implementación es un número *unsigned* (sin signo), con cero representando el nivel más bajo de voltaje. El voltaje de salida es solamente positivo. Un cero en la entrada produce un voltaje de cero en la salida. Para señales AC, la desviación *bias* positiva, en la señal analógica puede ser removida con acoplamiento capacitivo a la carga.

La Figura. 3.19, es el diagrama de bloques detallado de un *XPS Delta – Sigma DAC*. El ancho de la entrada binaria en la implementación descrita es configurable, en el ejemplo que muestra la figura muestra un DAC con una entrada de 8 bits binaria (*DACIn*, parte central de la figura).

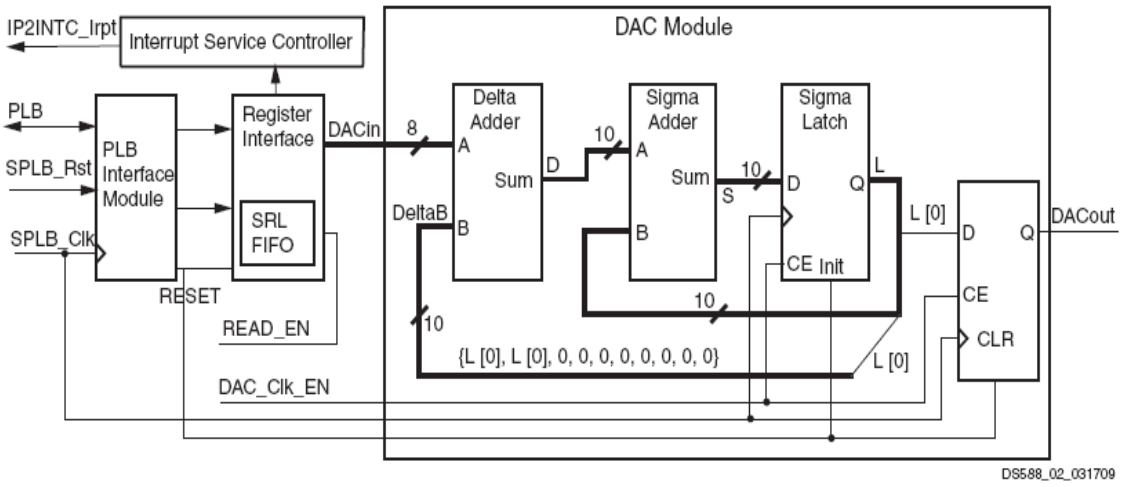

Figura. 3.19. Diagrama de Bloques del *XPS Delta – Sigma DAC core*

Fuente: XILINX, Inc., 2010

El término *delta-sigma* se refiere a la diferencia y suma aritméticas respectivamente, debido a que se utiliza sumadores (*adders*) binarios, para crearlas. A pesar de que la entrada del *Delta adder* sea *unsigned*, la salida de ambos sumadores son consideradas como números con signo.

[32] **Filtro Pasa bajos:** Es un filtro pasivo ya que está compuesto por una resistencia y un capacitor. En general los filtros permiten el paso o no de las frecuencias dependiendo el valor de estas. Este filtro permite el paso de las frecuencias desde cero hasta una frecuencia específica o frecuencia de corte.

El *Delta Adder* calcula la diferencia entre la señal de 8 bits *DACin* y la salida del valor presente en la señal *DACout*. En la Figura. 3.19 se indica la diferencia al sumar la entrada *DACin* a un valor creado por la concatenación de dos copias del bit más significativo (*L[0]*) del bloque *Sigma Latch* con todos los ceros de la señal *Delta B*. Esto compensa el hecho de que la entrada *DACin* sea *unsigned*.

El bloque *Sigma Adder* suma su salida anterior mantenida en *Sigma Latch*, con la salida actual del *Delta Adder*. Esto se hace porque todos los bits en cualquiera de las entradas A o B son iguales a cero, por lo que las entradas A y B del *Sigma Latch*, son combinadas en lugar de sumarse. Como se nota la entrada *DACin*, puede ser ampliada un bit mas, para permitir el rango analógico completo de 0V a V_{CCO}.

Para la implementación física basada en la Figura. 3.17, el voltaje de salida V_{out} se representa como una función de la entrada *DACin* y puede ser expresada con la siguiente fórmula:

$$V_{OUT} = \left(\frac{DACin}{2^{(C\_NUM\_DAC\_BITS)}}\right) x\ V_{CCO}\ \ [Volts]$$

Ecuación 3.1

Por ejemplo, para un DAC de 8 bits (C_NUM_DAC_BITS = 8), el voltaje V_{OUT} más bajo es 0V cuando *DACIn* es 0x00. El valor más alto de V_{OUT} es $\frac{255}{256} * V_{CCO}$ cuando DACIn sea 0xFF. Los valores mínimos y máximos permitidos por C_NUM_DAC_BITS son de 2 a 16 de tipo entero.

Recomendaciones para el diseño del Filtro Pasa bajos

El filtro pasa bajos RC mostrado en la Figura. 3.17 es adecuado para la mayoría de aplicaciones. Un buffer de 24mA LVTTL es utilizado a la salida para proporcionar la máxima corriente para manejar el filtro.

Hay tres recomendaciones principales a tomar en cuenta al escoger los valores de resistencia y capacitor:

1. **Fuente de salida de voltaje y corriente**: es importante que la señal *DACout*

siempre tenga el rango completo de voltaje, es decir de 0V a V_{CCO}, si el valor de R es demasiado bajo y la señal *DACout* no puede recorrer todo el rango, la salida analógica se vuelve no lineal. El peor caso de la impedancia de salida de 24mA LVTTL, es alrededor de los 25 W. Así que el valor de la Resistencia R debe ser de 2.5KΩ o mayor para asegurar el rango completo, con un error de 1% o menos.

2. **Impedancia de carga**: mantenga el valor de R relativamente bajo a la impedancia de la carga, para que el cambio de corriente hacia el capacitor durante la carga se vuelva despreciable.
3. **Constante de Tiempo**: la constante de tiempo del filtro ($\tau = RC$), debe ser lo suficientemente alta para atenuar enormemente los pulsos individuales en la cadena de pulsos.

Registros de Trabajo

La Tabla. 3.2 muestra los registros internos del XPS Delta - Sigma.

Nombre del Registro	Dirección	Acceso
Device Global Interrupt Enable Register (GIE)	C_BASEADDR + 0x01C	Lectura/Escritura
IP Interrupt Status Register (IPISR)	C_BASEADDR + 0x020	Lectura/Escritura
IP Interrupt Enable Register (IPIER)	C_BASEADDR + 0x028	Lectura/Escritura
Control Register (CR)	C_BASEADDR + 0x100	Lectura/Escritura
Data FIFO[33] (FIFO)	C_BASEADDR + 0x104	Lectura/Escritura
Data FIFO Occupancy (OCCY)	C_BASEADDR + 0x108	Lectura
Data FIFO programmable depth interrupt Register (PIRQ)	C_BASEADDR + 0x10C	Lectura/Escritura

Tabla. 3.2. Registros del XPS Delta – Sigma DAC

Fuente: XILINX, Inc, 2010

Registro de Control (CR)

La descripción de cada bit para este registro se muestra en la Tabla. 3.3 y Figura.

[33] FIFO: (First In First Out): estructura de datos tipo cola donde el primer elemento en entrar será también el primero en salir.

3.20.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0 – 29	Reservado	-	No Asignado	Reservado
30	FIFO_RST	Lectura/Escritura	"0"	**FIFO Reset** "1"=Resetea la Data FIFO "0"=Data FIFO en operación normal
31	EN	Lectura/Escritura	"0"	**Habilitación del DAC** "1"=Habilita el XPS Delta - Sigma DAC "0"=Resetea y deshabilita el XPS Delta Sigma DAC. La salida del DAC será cero

Tabla. 3.3. Definiciones de Bits del Registro de Control del XPS DAC

Fuente: XILINX, Inc., 2010

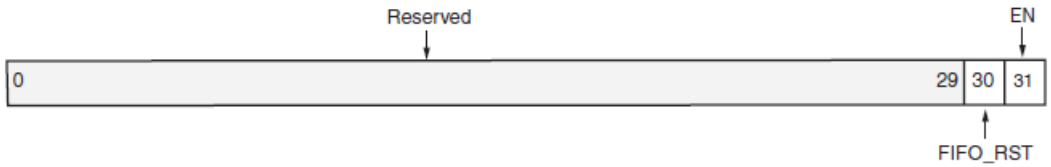

Figura. 3.20. Registro de Control de *XPS Delta – Sigma*

Fuente: XILINX, Inc., 2010

Data FIFO

Esta *SRL*[34] *FIFO* de 16 entradas de profundidad, contiene los datos de salida del *XPS Delta – Sigma DAC*. Es decir el dato que será convertido de digital a análogo. El *Data FIFO* se muestra en la Tabla. 3.4. La lectura de esta ubicación se traducirá en la lectura de la palabra de salida actual que está en la FIFO. No se recomienda intentar escribir, llenando la FIFO completamente, ya que los resultados en el byte de datos se pueden perder.

Cuando el Data FIFO está vacío y el DAC está habilitado, la salida del DAC será cero.

[34] SRL (Shift Register Lookup table): registro de desplazamiento de longitud variable.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0 to [31 - C_NUM_DAC_BITS - C_FULL_RANGE]	Reservado	-	-	Reservado
[32 - C_NUM_DAC_BITS - C_FULL_RANGE] to 31	Data	Lectura / Escritura	Indeterminado	Datos de Salida del DAC

Tabla. 3.4. Definiciones de Bits del Data FIFO del XPS Delta – Sigma DAC

Fuente: XILINX, Inc., 2010

Cabe recalcar que cuando C_FULL_RANGE es "1", la salida del DAC será a escala completa, cuando es "0" la salida del DAC será 1 LSB menos que la escala completa. En el ejemplo de la Figura. 3.21, se muestra el espacio de los datos cuando C_NUM_DAC_BITS es establecido a 8 y el C_FULL_RANGE, es establecido a "0".

Figura. 3.21. Data FIFO

Data FIFO Occupancy Register (OCCY)

Este campo tiene el valor de ocupación del Data FIFO. Al leer este registro se puede determinar si la FIFO está vacía o no. Además el *Data FIFO Empty Interrupts* transmite esa información, leer todo cero implica que ninguna locación ha sido llenada, y leer 10000 implica que las dieciséis locaciones están llenas. La Tabla. 3.5 y la Figura. 3.22 detallan la definición de los bits de este registro.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0 - 26	Reservado	NA	-	Reservado
27 - 31	Valor de Ocupación	Lectura	0x0	El bit 27 es el MSB. Un valor de "10000" implica que todos las 16 locaciones en la FIFO están llenas

Tabla. 3.5. Definiciones de Bits del Registro OCCY del XPS *Delta – Sigma DAC*

Fuente: Xilinx Inc, 2010

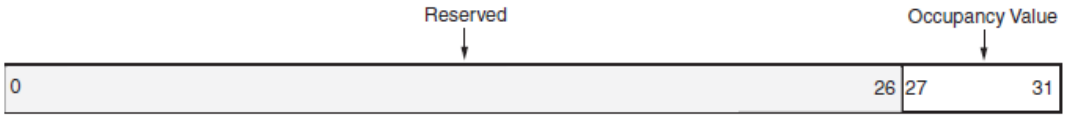

Figura. 3.22. Registro OCCY

Fuente: XILINX, Inc., 2010

Data FIFO Programmable Depth Interrupt Register (PIRQ)

Este campo contiene el valor que causará que se establezca la interrupción PIRQ, siempre y cuando este valor sea mayor o igual al valor OCCY y quedara así hasta el valor en mención disminuya. La definición de bits es expresada en la Tabla. 3.6 y la Figura. 3.23.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0 - 26	Reservado	-	-	Reservado
27 - 31	Valor de Comparación	Lectura/Escritura	0x0	El bit 27 es el MSB. Un valor de "00101" implica que a lo menos 5 o menos de 5 locaciones en la DATA FIFO están llenas, La interrupción FIFO PIRQ será establecido.

Tabla. 3.6. Definiciones de Bits del Registro *Data FIFO Programmable Depth Interrupt*

Fuente: XILINX, Inc., 2010

Figura. 3.23. Registro *Data FIFO Programmable Depth Interrupt Register*

Fuente: XILINX, Inc., 2010

Procedimiento para implementación de un IP Core DAC

Los pasos subsecuentes son requeridos, con el fin de establecer los registros del DAC para inicializar una conversión D/A:

- Inicialice los registros de interrupción GIE (“1”, para habilitar, y “0” para deshabilitar, en el bit de este registro),
- Escriba el dato a ser convertido en el *Data FIFO*.

- Establezca el *PIRQ*, para generar una interrupción antes de que la *FIFO* esté vacía.
- Habilite el *DAC* escribiendo un "1" en su registro de control (*CR*).
- Active la entrada Read_en poniéndola en alto solamente por un ciclo del *SPLB_CLK*, el valor escrito en la *Data FIFO* comenzará a ser convertido. La Figura. 3.24, indica el diagrama de tiempo requerido para la conversión D/A. Si la señal *Read_en* está en alto y el *Data FIFO* esta vacio, la salida *DACout* será cero "0". La salida *DACout* permanecerá en cero hasta que un dato sea escrito en la *Data FIFO* y *Read_en* sea activado correctamente.
- Escriba el nuevo valor en la *Data FIFO*, si un nuevo valor es requerido, considerando el paso anterior.

Cabe recalcar que la señal *DAC_CLK_En* (*DAC Clock Enable*), permite que el SPLB_Clk sincronice el *SIGMA LATCH* y el flip flop tipo D (Figura. 3.19).

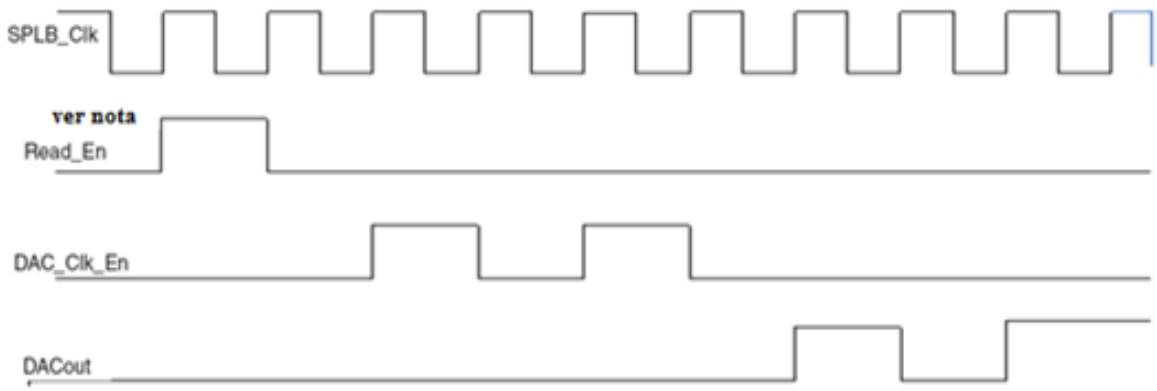

Figura. 3.24. Diagrama de Tiempo de DAC core[35].

Fuente: XILINX, Inc., 2010

El IP Core delta-sigma DAC, necesita un pulso que dure exactamente un ciclo de reloj en su entrada *Read_En* (Figura. 3.26), para enviar un dato de su FIFO de 16 registros a los bloques de modulación *delta-sigma*, de esta manera tener a la salida de este *IP Core* la información modulada y lista para ser reconstruida en el filtro pasa bajos.

Este pulso no puede ser generado solamente por *software* ya que la ejecución de las instrucciones necesarias para *setear* y *resetear* un bit toman varios ciclos de reloj. Por esta

[35] **Nota:** El accionamiento en alto de *Read_En* por un *SPLB_CLK*. El primer valor escrito en la Data FIFO comenzará a ser convertido dos *DAC_CLK_En* mas tarde. *Read_En* debería ser generado si la FIFO no está vacía.

razón se implementa un circuito basado en *flip flops* del grupo *utility* del *IP Catalog*, analizados con anterioridad, con el fin de generar el pulso requerido y asegurar que su duración sea exactamente de un ciclo de reloj[36]. Estas conexiones internas se muestran en la Figura. 3.25:

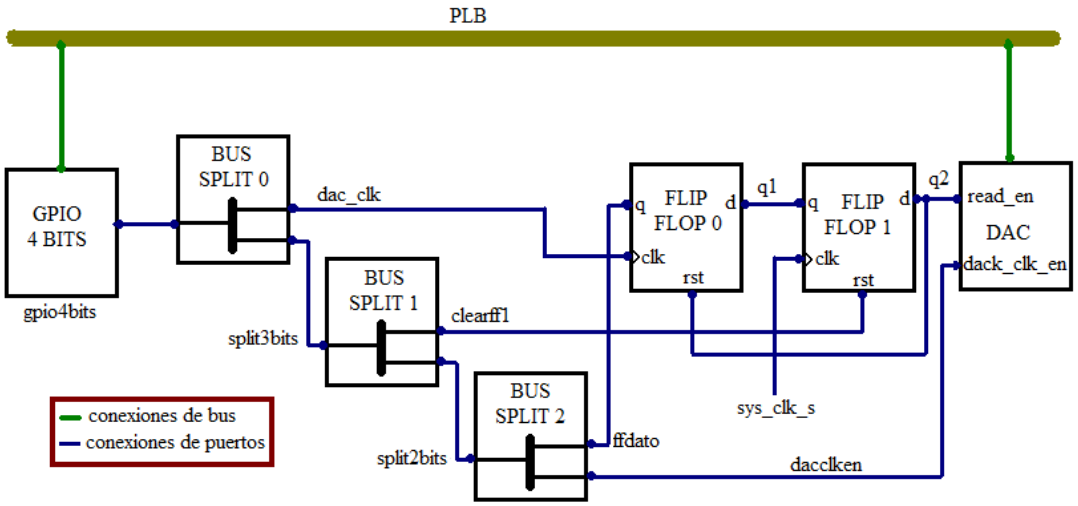

Figura. 3.25. Diagrama de conexiones para inicializar la conversión de DAC core.

El diagrama de tiempos correspondiente al esquema antes mostrado, se ilustra en la Figura. 3.26. Donde se observa el comportamiento de las salidas "q1" y "q2", correspondientes a los *flip flops*, en relación a la manipulación de los bits del GPIO.

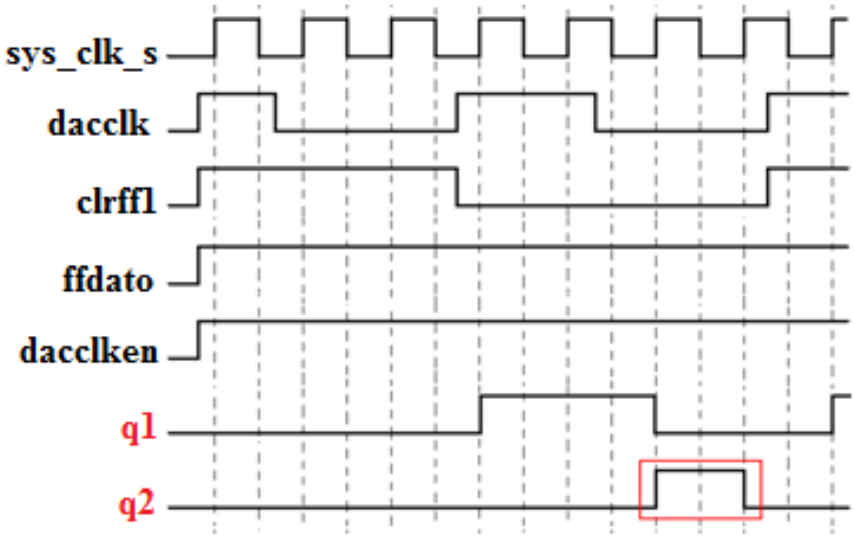

Figura. 3.26. Diagrama de tiempos de las formas de ondas de señales internas de las conexiones realizadas para iniciar conversión en un DAC Core.

[36] **Nota:** Estas conexiones deben ser realizadas en la pestaña ***Ports*** del XPS, ya que en esta sección se realizan las conexiones internas necesarias.

Se puede ver que la secuencia necesaria para generar un pulso de un ciclo de reloj de duración para la entrada *Read_en* del *XPS Delta – Sigma DAC* es: "1111" - "0111" – "0011" – "1111", donde "*dacclken*" es el bit menos significativo.

XPS DELTA-SIGMA ADC (CONVERSOR ANALOGO - DIGITAL)

En la mayoría de los sistemas electrónicos resulta conveniente efectuar las funciones de regulación y control automático de sistemas mediante técnicas digitales. Sin embargo en muchos de los casos la señal disponible normalmente es analógica. Esta medida corresponde a la señal proveniente de un sistema de audio, de video, de puntos de medición, celdas extensiométricas, termopares, etc.

El dispositivo capaz de convertir una entrada analógica de voltaje en un valor binario o señal digital es el ADC, el valor de este número es directa o inversamente proporcional al voltaje, estas señales digitales minimizan la distorsión producida por las imperfecciones del sistema que las originó. La conversión análoga a digital es realizada en el *IP Core XPS Delta – Sigma ADC* (*XPS ADC*), utilizando la técnica de conversión *Delta – Sigma*, que se detalló en el tema anterior. Este *soft IP Core*, está diseñado para comunicarse con el bus PLB.

Características:

Las características de un IP Core ADC comprenden:

- Se conecta como un esclavo de 32 bits en buses de 32, 64 o 128 bits del PLB V4.6.
- Tiene una resolución seleccionable.
- Posee una FIFO de datos de 16 entradas de profundidad.

Descripción Funcional

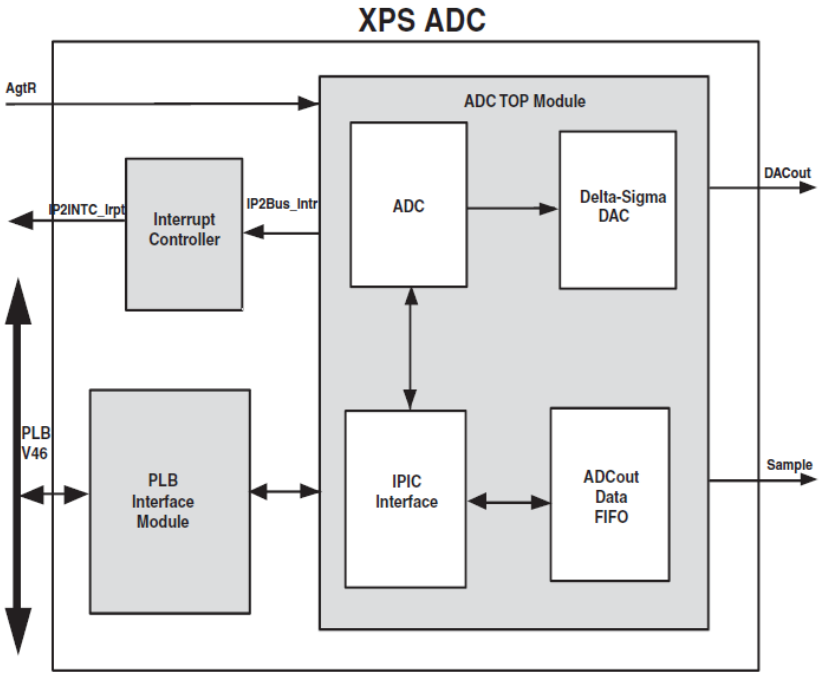

Figura. 3.27. Diagrama de Bloques del XPS ADC *core*.

Fuente: XILINX, Inc., 2010

La Figura. 3.27 muestra los bloques o módulos por los cuales está comprendido un XPS ADC.

***Interrupt Controller*:** Proporciona soporte para las interrupciones del *XPS ADC*, por medio de este modulo se solicita la atención del microprocesador.

***PLB Interface Module*:** Este modulo proporciona la comunicación bidireccional entre el *IP Core XPS ADC* y el bus PLB.

***IPIC Interface*:** Es un conjunto de señales que conectan al *XPS ADC* con el *PLB Interface Module*. Este modulo genera las señales de solicitud de lectura / escritura requeridas, utilizando las señales de la FIFO.

***ADCout Data FIFO*:** Es una pila FIFO de 16 entradas de profundidad, para el almacenamiento de los valores analógicos convertidos.

***Delta-Sigma DAC*:** Este DAC es utilizado para generar un voltaje de referencia

para la entrada negativa del comparador externo (Figura. 3.28). Al utilizar un DAC el máximo rango de voltaje analógico que puede ser convertido estará definido entre 0V y V_{CCO}. Por supuesto, que esta salida DACout puede ser amplificada exteriormente para aumentar este rango.

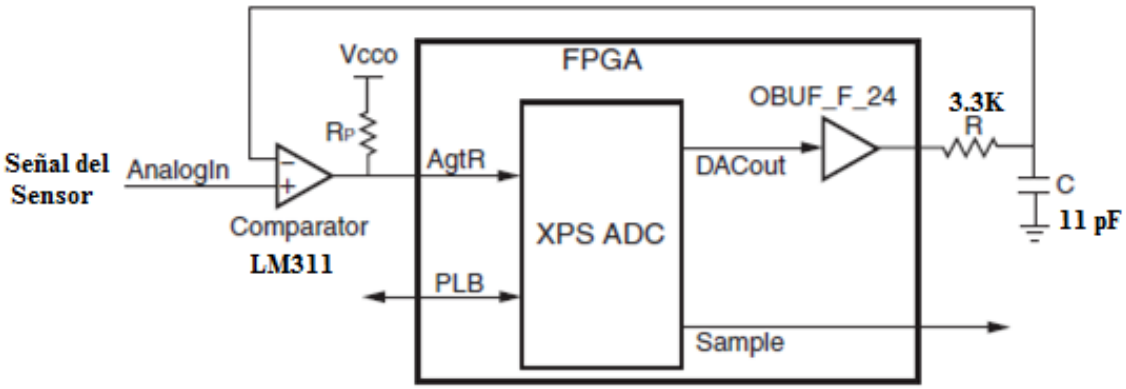

Figura. 3.28. Implementación de un Conversor Análogo a Digital utilizando XPS ADC. Fuente: XILINX, Inc., 2010

La entrada analógica es determinada mediante la decodificación del tren de pulsos en el pin AgtR (*Analog greater that Reference*) proveniente del comparador externo. Para cada bit del tren de pulsos, solamente el bit más significativo de la entrada *DAC input* es inicialmente seteado, lo que conduce al voltaje de referencia a la mitad del rango al iniciar la conversión. Dependiendo de la salida del comparador, el bit más significativo es llevado a cero o permanece seteado, y el siguiente bit más significativo del *DAC input* es seteado. Este proceso continua para cada bit de *DAC input*.

El DAC es un bit más ancho que la salida del ADC. Esto es necesario para que el bit numerado como más bajo de la salida del ADC sea significativo. Cuando todos los bits han sido muestreados los bits superiores del registro de alimentación del DAC se transfiere al registro de salida del ADC.

Tasa de Muestreo

La tasa de muestreo del XPS DAC puede ser expresado con la siguiente ecuación:

$$XPS\ ADC_{SR} = \frac{f_{Clk}}{\left(2^{(C\_DACIN\_WIDTH)} x (FSTM + 1) x (C\_DACIN\_WIDTH)\right)} \quad [^{muestras}/_{segundo}]$$

Ecuación 3.2

Los conversores análogo a digital convencionales requieren al menos dos veces la mayor frecuencia de entrada como tasa de muestreo. Se recomienda que el f_{Clk} sea igual que PLB Clock para eliminar el ruido producido en la entrada de la señal.

La Tabla. 3.7 muestra el rango de la frecuencia de la señal *AnalogIn* y el muestreo del ADC para varias frecuencias PLB Clock y valores *Filter Settle Time Multiplier* (FSTM).

Frecuencia PLB Clock	Valor de Carga FSTM	Rango de la Frecuencia de la Señal AnalogIn	Tasa de Muestreo del ADC [muestras / seg]
40 MHz	4	<868 Hz	1736
80 MHz	4	<1736 Hz	3472
100 MHz	4	<2170 Hz	4340
40 MHz	8	<482 Hz	964
80 MHz	8	<965 Hz	1929
100 MHz	8	<1205 Hz	2411

Tabla. 3.7. Calculo de la Tasa de Muestreo del XPS ADC (con C_DACIN_WIDTH = 9)

Fuente: XILINX Inc., 2010

Descripción de Registros del ADC

La Tabla. 3.8 muestra los registros del XPS ADC y sus direcciones.

Dirección Base + Offset (Hex)	Nombre del Registro	Acceso	Descripción del Registro
C_BASEADDR + 0x01C	GIE	Lectura / Escritura	Global Interrupt Enable
C_BASEADDR + 0x020	IPISR	Lectura	Registro IP de Estado de Interrupción

Dirección Base + Offset (Hex)	Nombre del Registro	Acceso	Descripción del Registro
C_BASEADDR + 0x028	IPIER	Lectura / Escritura	Registro IP de Habilitación de Interrupción
C_BASEADDR + 0x100	ADCCR	Lectura / Escritura	Registro de Control de ADC
C_BASEADDR + 0x104	FIFO	Lectura	Data FIFO ADCout
C_BASEADDR + 0x108	OCCY	Lectura	ADCout Data FIFO Occupancy

Tabla. 3.8. Registros del XPS ADC

Fuente: XILINX, Inc., 2010

Registro Global Interrupt Enable (GIE)

Este registro permite la habilitación o deshabilitación de las interrupciones globales del sistema que van al procesador y residen en el *PLB Interface Module*. La Tabla. 3.9 y la Figura. 3.29, indican la especificación de los bits de este registro.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0	Global Interrupt Enable	Lectura / Escritura	"0"	"1" = Habilitar "0" = Deshabilitar
1 al 31	No utilizado	-	0	Inutilizado

Tabla. 3.9. Definición de los Bits de GIE

Fuente: XILINX, Inc., 2010

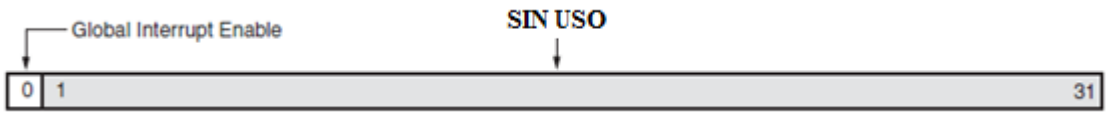

Figura. 3.29. Registro GIE

Fuente: XILINX, Inc., 2010

Registro de Control de ADC (ADCCR)

Este registro contiene el bit de habilitación de conversión (EC), y el *Filter Settle Time Multiplier* (FSTM). El bit EC habilita/deshabilita el proceso de conversión análogo – digital. El FSTM es un valor binario, que depende de las características del filtro pasa bajo RC (utilizado para la conversión del tren de pulsos de *DACout* en una señal análoga equivalente). El ancho del valor de FSTM es

configurado con el parámetro C_FSTM_WIDTH. Para la mayoría de aplicaciones el valor de 4 bits es suficiente. En la Tabla. 3.10 se indica la definición de bits del ADCCR y en la Figura. 3.30 se indican donde se sitúan los bits EC y FSTM.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0	Bit de Habilitación de Conversión (EC)	Lectura / Escritura	"0"	"1" = Habilitar "0" = Deshabilitar
1 a [(32-C_FSTM_WIDTH)-1]	Inutilizado	-	0	Inutilizado
[32-C_FSTM_WIDTH] to 31	FSTM	Lectura / Escritura	0	Estos bits mantienen el valor binario, el cual depende de las características RC del filtro pasa bajos

Tabla. 3.10. Definición de los Bits de ADCCR

Fuente: XILINX, Inc., 2010

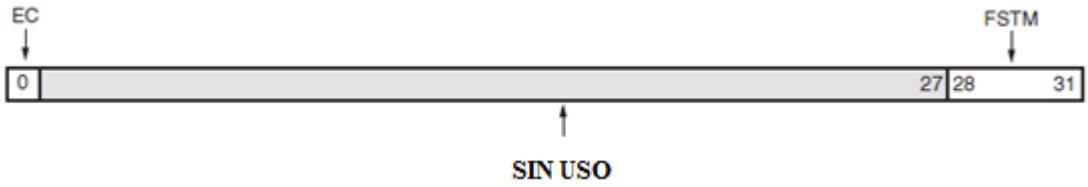

Figura. 3.30. Registro ADCCR (C_FSTM_WIDTH = 4)

Fuente: XILINX, Inc., 2010

ADCout Data FIFO (FIFO)

Esta FIFO de 16 entradas de profundidad, contiene los datos que saldrán por el XPS ADC. La definición de bits de este registro está definida en la Tabla. 3.11. Mediante *software* se deberá chequear si hay presencia de datos antes de leer este registro. Dos advertencias en cuanto al manejo eficiente de la FIFO, primero si se lee la FIFO y no existen datos almacenados en ella, se producirá un error y el resultado será indefinido, y segundo si la FIFO está llena los datos que sigan ingresando en esta FIFO se perderán. La Figura. 3.31 muestra la ubicación de los datos cuando C_DACIN_WIDTH es establecido a 9.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0 al [32-C_DACIN_WIDTH]	Inutilizado	-	0	Inutilizado
[(32-C_DACIN_WIDTH)+1] al 31	ADCout	Lectura	0	El Valor Digital equivalente de la muestra de la entrada analógica. El número de la resolución de ADCout es C_DACIN_WIDTH - 1.

Tabla. 3.11. Definición de los Bits de ADCout Data FIFO

Fuente: XILINX, Inc., 2010

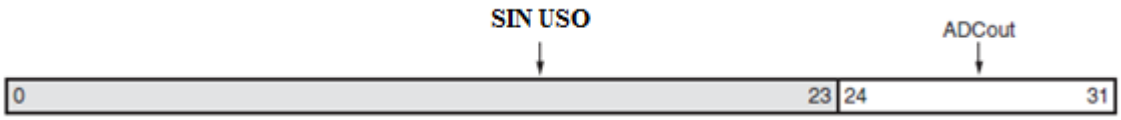

Figura. 3.31. Registro ADCout Data FIFO (C_DACIN_WIDTH = 9)

Fuente: XILINX, Inc., 2010

ADCout Data FIFO Occupancy Register (OCCY)

El registro ADCout Data FIFO Occupancy, contiene el valor de ocupación del ADCout Data FIFO. Leyendo este valor se puede determinar si la FIFO está vacía. El valor leído es el valor de la cuenta binaria, por lo tanto si se lee todo ceros, implica que ninguna locación está llena, y si se lee 10000 implica que todas las 16 locaciones están llenas. La Tabla. 3.12 y la Figura. 3.32 indican la definición de bits del registro OCCY.

Bit(s)	Nombre	Acceso al Core	Valor de Reset	Descripción
0 al 26	Inutilizado	-	0	Inutilizado
27 al 31	Data Occupancy	Lectura	0	Numero de palabra de datos que se encuentran actualmente en el ADCout Data FIFO

Tabla. 3.12. Definición de los Bits del Registro OCCY

Fuente: XILINX, Inc., 2010

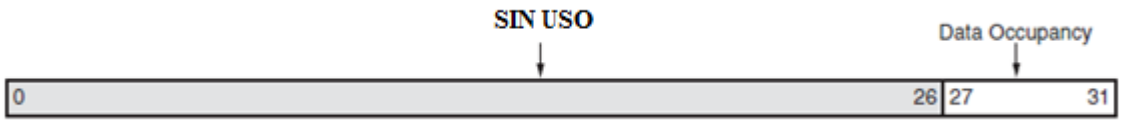

Figura. 3.32. Registro OCCY (C_DACIN_WIDTH = 9)
Fuente: XILINX, Inc., 2010

XPS 16550 UART

UART, son las siglas de "*Universal Asynchronous Receiver-Transmitter*". Este *IP Core* controla el puerto serial de la tarjeta SP605. Su función es la de tomar bytes de datos y transmitir los bits individuales de forma secuencial, desde el CPU, y cuando recibe los datos tomar estos bits individuales y formar el byte ordenadamente desde un modem.

Características

Los IP Cores UART poseen las siguientes características:

- Se conecta como un esclavo de 32 bits en buses de 32, 64 o 128 bits del PLB V4.6.
- Presenta registros de hardware y software que son compatibles con todos los estándares UARTs 16450 y 16550.
- Implementa todos los protocolos estándar de interfaz serial.
    - 5, 6, 7, u 8 bits por carácter
    - Detección y generación de paridad impar, par, o ninguna de ellas
    - Detección y generación de 1, 1.5, o 2 bits de stop.
    - Generador interno del *baud rate*
    - Funciones de control del módem
    - Priorización de la transmisión, recepción, línea de estado, y control de las interrupciones del módem
    - Salida en falso de la detección de bit y su recuperación
    - Detección y generación de la línea de ruptura.
    - Funcionalidad de diagnóstico de bucle interno
    - FIFO de 16 bytes para la transmisión y recepción.

La configuración básicamente de este *IP Core* consiste en:

- En XPS:

  - Conecte el *IP Core UART*, al bus PLB, en el SAV (*System Assembly View*).
  - Diríjase a la pestaña *Ports* del SAV, despliegue las señales del IP Core UART, encuentre las señales "*sin*" y "*sout*" y en cada una de estas señales seleccione "*Make External"* al desplegar su lista de conexiones[37].
  - Vaya a la parte superior de la pestaña *Ports*, es decir en *External Ports*, seleccione la dirección apropiada de la señal "*sin*" y "*sout*". Señal "*sin*" como entrada I y "*sout*" como salida O.
  - Tome en cuenta los nombres que se colocaron en la sección *External Ports*, para la declaración de los pines del FPGA en el archivo UCF. Observe cuales son los pines del FPGA conectados en la tarjeta SP605, para el puerto RS232 UART.
  - Añada en el archivo UCF las siguientes líneas:

    # Module RS232_UART constraints
    Net RS232_UART_RX_pin LOC=AJ8 | IOSTANDARD = LVCMOS25;
    Net RS232_UART_TX_pin LOC=AE7 | IOSTANDARD = LVCMOS25 | SLEW = SLOW | DRIVE = 1

  - En la pestaña *address*, se deberá asignar el tamaño del rango direcciones que tendrá este *IP Core*, por lo general es de 64K. Pulse *generate address*, para autogenerar las direcciones *BASE ADDRESS* y *HIGH ADDRESS*.

- En SDK:

  - En *Board Support Package Settings* seleccione la opción del RTOS que haya escogido y seleccione el IP Core UART (ejemplo: *RS232 Uart 1*) en la columna *Value*, para "STDIN" y "STDOUT", como se indica en la Figura. 4.19.

[37] **Nota:** La señal "*sin*" es para recepción de datos RX y la señal "sout" es para transmisión de datos TX

XPS GENERAL PURPOSE INPUT/OUTPUT (GPIO)

El *IP Core* GPIO permite la utilización de entradas y salidas digitales. Puede ser configurado para lectura y escritura de datos. Entre los usos comunes tenemos: recibir señales de *dip switches* o pulsadores y escritura de datos en *leds*.

Características

En un IP Core GPIO se puede:

- Configurar el número de bits desde 1 hasta 32.
- Programar dinámicamente cada bit como entrada o salida.
- Escoger entre utilizar uno o dos canales de cada GPIO.
- Configurar individualmente el ancho de cada canal.

Configuración

Al configurar este *IP Core*, se debe seleccionar:

- Un único canal.
- El ancho del canal escogido, según el uso que se le dará, teniendo como ancho máximo 32 bits.
- *TRUE*, si el canal 1 es solamente de entrada (*Channel 1 is Input only*), o *FALSE*, si el canal 1 será de escritura o lectura/escritura, como el ejemplo que indica la Figura. 3.33, este se lo utilizó para salida de información destinado al uso de *Leds*.

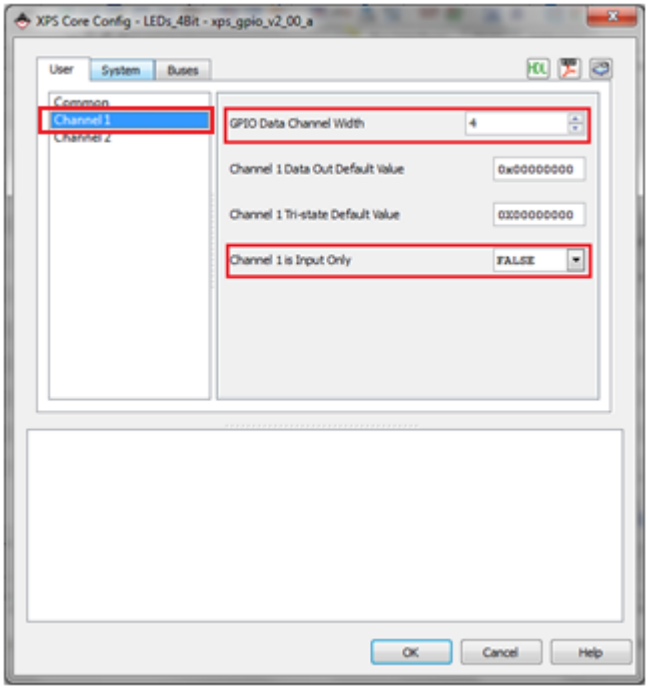

Figura. 3.33. Configuración del IP Core GPIO

Se debe tomar en cuenta la dirección base para escribir y leer los datos de los registros y poder interactuar con el GPIO.

XPS TIMER / COUNTER

Este *IP Core* es un módulo timer/counter de 32 bits que se conecta directamente al bus PLB.

Características

Las características más importantes de este *core* incluyen:

- Conexión como un esclavo de 32 bits al bus PLB v4.6 ya sea este de 32, 64, ó 128 bits.
- Contiene dos *timers* programables con interrupciones, generación de eventos, y la capacidad de capturar eventos.
- Ancho del *counter* configurable.
- Salida de PWM[38]

[38] PWM: Modulación por Ancho de Pulso (*Pulse Width Modulation*)

- Entrada *Freeze,* empleada para detener contadores durante la depuración de *software*.

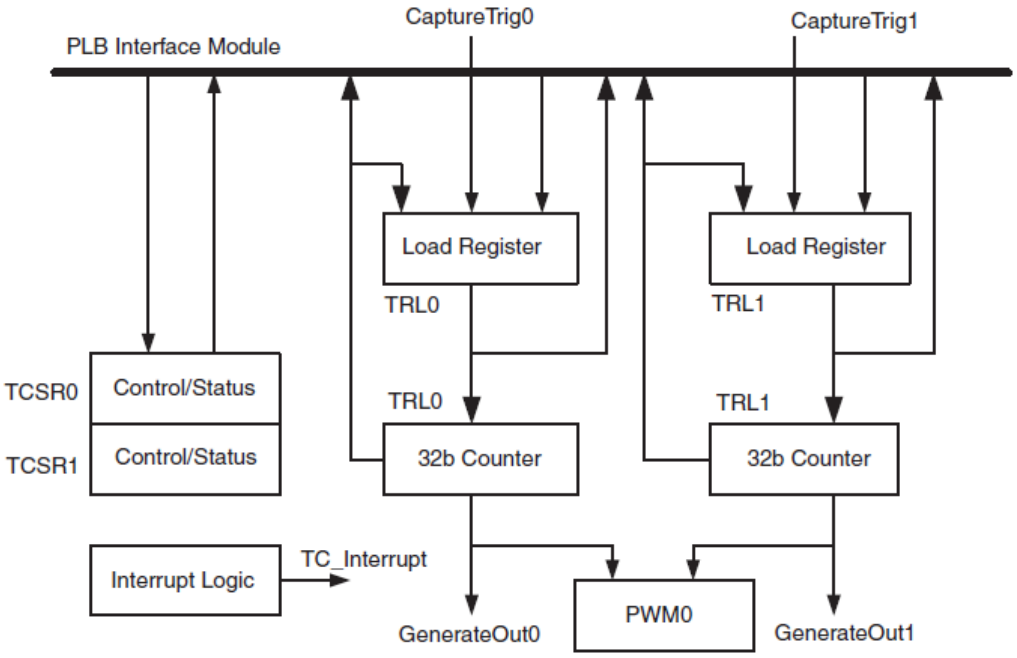

Figura. 3.34. Diagrama de Bloques detallado del XPS Timer/Counter
Fuente: XILINX, Inc., 2010

El *timer/counter* está constituido por dos módulos *counter* de 32 bits idénticos, como se muestra en la figura anterior. Cada uno de ellos asociado a un *load register* (registro de carga). Este registro es utilizado para mantener el valor inicial del contador cuando ocurre la generación de eventos o captura de valores dependiendo del modo del *timer*.

Existen tres modos que pueden ser utilizados con los dos módulos *timer/counter*.

- Modo Generación
- Modo Captura
- Modo PWM

Modo Generación

En este modo se puede generar un pulso durante un ciclo de reloj o un tren de pulsos con un periodo específico.

Para que se produzca la generación de señales, el valor presente en el *load register* se carga en el contador. Una vez hecho esto, el *counter* es habilitado y comienza a contar hacia arriba o hacia abajo, dependiendo de la selección del bit UDT (0 para cuenta ascendente 1 para cuenta descendente). Si se activa la señal externa *GenerateOut* (bit GENT del registro TCSR[39], 0 deshabilitar y 1 habilitar) se generará un pulso al exterior de un ciclo de reloj de duración cada que el *counter* alcance el valor deseado. Esto generará un único pulso cuando se seleccione 0 en el bit ARHT, o un tren de pulsos cuando se seleccione 1 en este mismo bit.

Modo Captura

Este modo es útil para contar el tiempo que transcurre hasta que suceda un evento externo, generando una interrupción simultáneamente.

Se utiliza el reloj del sistema (SPLB_Clk) para muestrear la señal de captura. Si esta señal ha sido configurada como *high-true* realiza la captura cuando ocurre la transición de 0 a 1, si fuese configurada como *low-true* realizaría la captura en la transición de 1 a 0.

Cuando el evento de captura ocurre, el valor del *counter* es escrito en el *load register*. Este valor es llamado valor de captura.

Cuando el bit ARHT (*Auto Reload/Hold*) es establecido a 0 y el evento de captura ocurre, el valor de captura es escrito al *load register* el cual mantiene el valor de captura hasta que sea leído, si este registro no es leído, el evento de captura subsecuente no actualizará el *load register*, y se perderá el valor de captura.

Si el bit ARHT es establecido a 1 y el evento de captura ocurre, el valor de captura se escribirá constantemente en el *load register*. Los eventos de capturas subsecuentes serán actualizadas al *load register* y sobrescribirán los valores previos, así se haya leído o no.

Si se desea que haya una interrupción mientras un evento de captura se produce se debe establecer el bit TINT (0 para que no haya interrupción y 1 para que haya interrupción). Es necesario borrar la bandera TINT para que nuevos valores puedan ser

[39] La descripción de cada uno de los registros será analizada mas adelante

capturados.

Modo PWM

En el modo PWM, se utiliza dos timer/counter para producir una señal de salida PWM0 con una frecuencia y un factor de trabajo (*duty factor*) específicos. El *Timer0* establece el periodo y el Timer1 establece el tiempo en alto para la salida PWM0. Para generar una salida PWM se debe:

- Establecer el Modo Generación (bit MDT a 0 en el registro TCSR) para ambos *timers* (Timer0 y Timer1).
- Establecer a 1, tanto el bit PWMA0 en el TSCR0 y PWMB0 en el TCSR1, para habilitar el modo PWM.
- Habilitar las señales *GenerateOut* en el TSCR (bit GENT = 1) para ambos *timers* para poder generar la señal PWM0.
- Establecer a 1, C_GEN0_ASSERT y C_GEN1_ASSERT.

El *counter* puede ser establecido para cuenta ascendente o descendente de pendiendo del bit UDT (0 - ascendente, 1 - descendente).

Configuración del Periodo y Factor de Trabajo (*duty factor*) en el modo PWM

El período de PWM es determinado por el valor de generación en el *load register* del Timer0 (TLR0). El tiempo en alto o *duty factor* del PWM es determinado por el valor de generación en el *load register* del Timer1 (TLR1). El período y el *duty factor* son cálculos según lo siguiente:

Cuando los *counters* son configurados para cuenta ascendente (UDT = 0)

$$PWM\_PERIOD = (MAX\_COUNT - TLR0 + 2) * PLB\_CLOCK\_PERIOD$$

Ecuación 3.3

$$PWM\_HIGH\_TIME = (MAX\_COUNT - TLR1 + 2) * PLB\_CLOCK\_PERIOD$$

Ecuación 3.4

Cuando los *counters* son configurados para cuenta descendente (UDT = 1)

$$PWM\_PERIOD = (TLR0 + 2) * PLB\_CLOCK\_PERIOD$$

Ecuación 3.5

$$PWM\_HIGH\_TIME = (TLR1 + 2) * PLB\_CLOCK\_PERIOD$$

Ecuación 3.6

Donde, MAX_COUNT es el máximo valor de conteo del *counter*, es decir 0xFFFFFFFF para el *counter* de 32 bits.

Descripción de Registros

Las direcciones utilizadas por los registros del XPS Timer/Counter son mostradas en la Tabla. 3.13.

Registro	Dirección (HEX)	Tamaño	Tipo	Descripción
TCSR0	C_BASSEADDR + 0x00	32 bits	lectura/escritura	Registro 0 de Control/Estado
TLR0	C_BASSEADDR + 0x04	32 bits	lectura/escritura	Load Register 0
TCR0	C_BASSEADDR + 0x08	32 bits	Lectura	Registro 0 Timer/Counter
TCSR1	C_BASSEADDR + 0x10	32 bits	lectura/escritura	Registro 1 de Control/Estado
TLR1	C_BASSEADDR + 0x14	32 bits	lectura/escritura	Load Register 1
TCR1	C_BASSEADDR + 0x18	32 bits	Lectura	Registro 1 Timer/Counter

Tabla. 3.13. Asignación de Direcciones para los Registros del XPS Timer/Counter

A continuación se detalla cada uno de los registros mostrados en la tabla anterior.

Load Register (TLR0 y TLR1)

Cuando el ancho del *counter* ha sido configurado con un valor menor que 32 bits, el valor del *load register* es justificado a la derecha en TLR0 y TLR1. El bit menos significativo es siempre asignado al bit 31 del *load register* (Figura. 3.40)**.** Cabe recalcar

que su valor de reinicio es de 0.

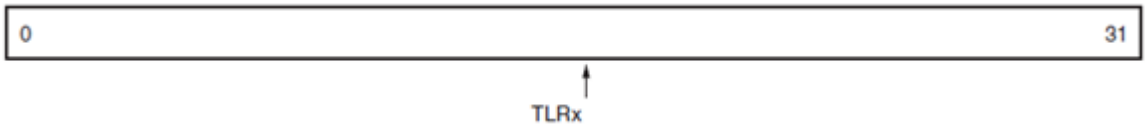

Figura. 3.35. Load Register (TLRx)

Registro Timer/Counter

Cuando el ancho del *counter* ha sido configurado con un valor menor que 32 bits, el valor de cuenta está justificado a la derecha en TCR0 y TCR1. El bit menos significativo es siempre asignado al bit 31 del registro Timer/Counter (Figura. 3.36)**.** Cabe recalcar que su valor de reinicio es de 0.

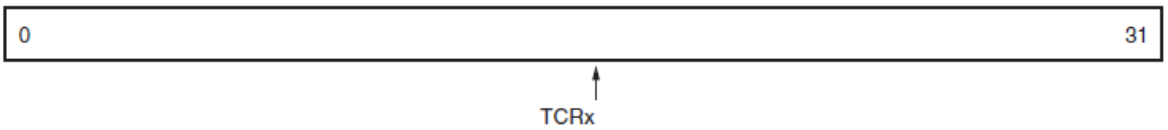

Figura. 3.36. Register Timer/Counter (TCRx)

Registro de Control/Estado (TSRCx)

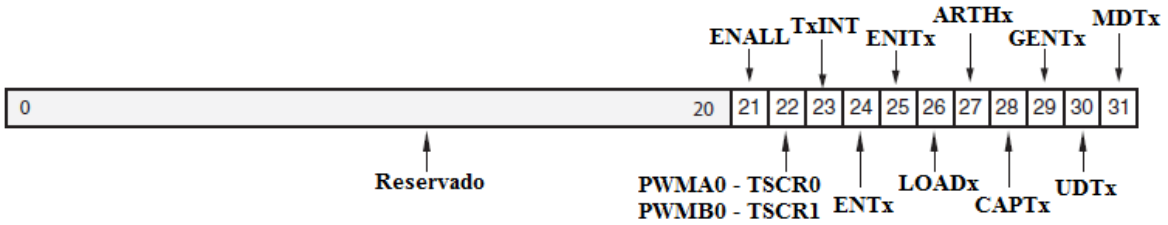

Figura. 3.37. Register Timer/Counter (TCRx)

Fuente: XILINX, Inc., 2010

Cabe recalcar que este registro es para dos timer/counters que son exactamente iguales, la x indica que puede ser 0 para TSCR0 o 1 para TSCR1. La descripción de cada uno de estos bits o banderas puede ser analizada en la hoja de datos del fabricante

2. Área de Desarrollo del Proyecto

System Assembly View

Esta vista contiene tres pestañas diseño para el desarrollo del proyecto, estas son:

- *Bus Interface.*- Permite modificar las conexiones de buses del diseño. En esta pestaña se cuenta con una representación gráfica de los buses del sistema llamada *Connectivity Panel.*
- *Ports.*- Permite modificar las conexiones de puertos internos y externos en el diseño.
- *Addresses.*- Muestra el rango de direcciones para cada *IP Core* del diseño. Estas direcciones pueden ser generadas automáticamente mediante el botón *Generate Addresses*.

3. Área de Depuración del Proyecto

Ventana de Consola

La ventana de consola, proporciona la información necesaria del estado de ejecución del sistema, por ejemplo muestra el proceso de la síntesis mientras se va ejecutando. Así como, en el caso de utilizar comunicación serial esta ventana sirve de interface entre el XPS y el usuario.

Además, en esta ventana se puede observar el tipo de errores o advertencias[40] que se produjeron durante la síntesis de *hardware* o compilación de un programa de aplicación.

HERRAMIENTAS DEL XPS

[40] **Nota:** Los errores y advertencias pueden ser visualizados de forma agrupada en las ventanas correspondientes según su propósito, es decir en la ventana de errores y en la ventana de advertencias.

Entre los instrumentos fundamentales que se necesitan para desarrollar los componentes de *hardware* y *software* de un sistema embebido procesado en XPS destacan:

- Base System Builder
- Create or Import Peripheral
- Libgen
- Standalone Board Support Package
- Xilinx Microprocessor Debugger
- Herramientas para la Generación de la Plataforma de Hardware: Platgen y Bitstream
- Herramienta para la Exportación de la Plataforma de Hardware al SDK

Base System Builder* (BSB),** es un asistente utilizado para crear nuevos proyectos con alto grado de personalización. Para esto, se escoge el procesador, y los diferentes *IP Cores* en base a la tarjeta de desarrollo utilizada. Al terminar, este asistente presenta en el XPS el diseño del sistema, incluyendo las conexiones de buses, puertos y el mapa de direcciones, para cada uno de los *IP Cores* esogidos. Se puede iniciar el BSB dirigiéndose a ***File > New Project en la ventana principal del XPS (Figura. 3.38).

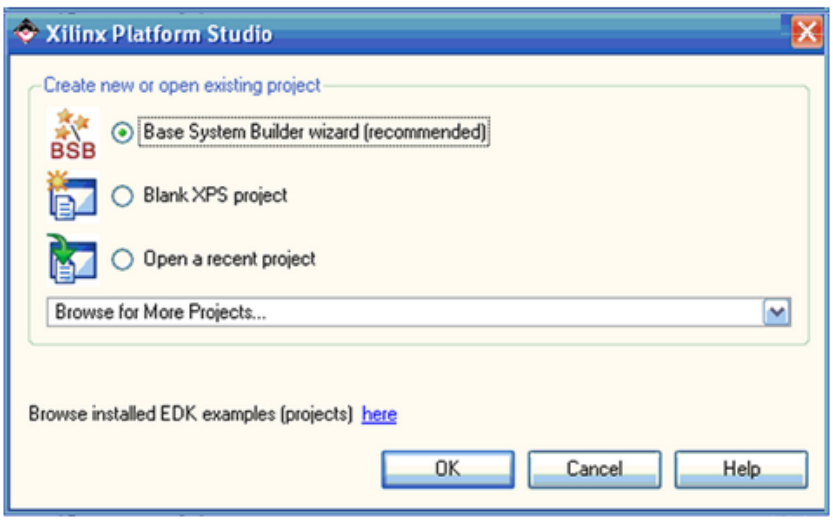

Figura. 3.38. Ventana de inicio de *Base System Builder*

Create or Import Peripheral, Este asistente ayuda a crear su propio periférico e importarlo dentro de los repositorios compatibles con EDK o proyectos de XPS. Para iniciar el asistente haga *click* en ***Hardware > Create or Import Peripheral.***

Libgen, es una herramienta para la generación de librerías. Esta herramienta configura librerías, controladores de dispositivos, archivos del sistema, y controladores de interrupciones, para un sistema embebido procesado. Click ***Software > Generate Libraries and BSPs***, para iniciar el *Libgen*.

Standalone Board Support Package (BSP), es un conjunto de módulos de *software* que acceden a funciones específicas del procesador. El BSP está diseñado para ser utilizado cuando una aplicación accede a la tarjeta o a las características del procesador directamente (sin que intervenga ninguna capa del Sistema Operativo), es típicamente construido con un gestor de arranque que contiene el mínimo soporte de dispositivo para cargar el Sistema Operativo y los *drivers* para todos los dispositivos de la tarjeta.

Xilinx Microprocessor Debugger (XMD), permite realizar una visualización a bajo nivel del funcionamiento de los elementos de cualquier tarjeta de desarrollo (*leds*, memorias, *Pushbuttons*, etc.) a través de la escritura y lectura de sus registros de trabajo.

Herramientas para la Generación de la Plataforma de Hardware: Platgen y Bitstream.- La plataforma de *Hardware* embebido, puede incluir uno o más procesadores, con los periféricos y bloques de memorias escogidos desde el *IP Catalog*. Todos estos *IP Cores* utilizan una interconexión basada en la arquitectura *CoreConnect* que proporciona XPS. El comportamiento de cada *IP Core* (procesador o periférico) puede ser personalizado con la ayuda del BSB o directamente por el usuario. Una plataforma de *hardware* se completa con la asignación de memoria correspondiente a cada *IP Core*, en un mapa de direcciones que se almacena en el procesador.

Desarrollo de la Plataforma de *Hardware* en el XPS

Como ya se mencionó anteriormente, el XPS proporciona un ambiente de desarrollo interactivo que permite especificar todos los aspectos y parámetros de configuración de su plataforma de *hardware*. XPS mantiene la descripción de la

plataforma de *hardware* en un formato de alto nivel, en el archivo *Microprocessor Hardware Specification* (MHS). El MHS define la configuración de su procesador embebido e incluye información sobre la arquitectura de buses, periféricos, conectividad de procesadores, así como el mapeo de direcciones.

La herramienta de Generación de la Plataforma de *Hardware* se llama ***Platgen***, y es utilizada para generar las conexiones del sistema embebido procesado, es decir el ***netlist***. Para iniciar la herramienta *Platgen*, dé click en ***Hardware > Generate Netlist.***

Generación del *Bitstream*

Al completar con éxito el proceso de diseño de un SoC en XPS, se puede utilizar el software *ISE Project Navigator* para generar el *bitstream*. El *ISE Project Navigator* lee el *netlist* que fue creado y junto al archivo UCF, producen un archivo BIT, el mismo que contiene el diseño de *hardware*. Cabe recalcar que el código de C compilado no forma parte del *bitstream*, este será agregado más adelante en el SDK. Para generar el *bitstream* diríjase a ***Hardware → Generate Bitstream***

En resumen para generar una plataforma de *Hardware*, se requiere de un proceso de tres pasos:

1. XPS genera un *netlist*, con la herramienta *Platgen*.
2. El diseño se implementa (asignación de pines del FPGA, en el archivo UCF). En caso de utilizar BSB, el archivo UCF se genera automáticamente y luego se edita por el diseñador.
3. El diseño implementado se convierte en *bitstream*, el cual puede ser descargado al FPGA.

Herramienta para la Exportación de la Plataforma de Hardware al SDK.- En el EDK, la plataforma de *hardware* es diseñada en XPS y su *software* es desarrollado y depurado en el SDK, por lo tanto la información acerca de la plataforma de *Hardware* es requerida por el SDK, razón que hace necesaria la exportación de un archivo llamado *system.xml*. Este archivo contiene la información que el SDK requiere para hacer el

desarrollo de *software* y el trabajo de depuración sobre la plataforma de *hardware* que se diseñó previamente. En el XPS. Para exportar la plataforma de hardware:

- Seleccione ***Project > Export Hardware Design to SDK.***
- Acepte la ubicación predeterminada.
- Seleccione ***Export Only***

3.3.1.2 SOFTWARE DEVELOPMENT KIT (SDK)

El SDK está basado en el estándar Eclipse de código abierto, y viene incluido en el *ISE Design Suite: Embedded Edition*. Además, al ser modular está disponible como un producto independiente para integrarlo al ISE.

El SDK es un programa que complementa directamente al XPS [27]. SDK es un ambiente de desarrollo recomendado para los proyectos de aplicaciones de *software* orientado a los procesadores embebidos de Xilinx: PowerPC y MicroBlaze.

En la Figura. 3.39, se muestra la ventana principal del *Software Development Kit*.

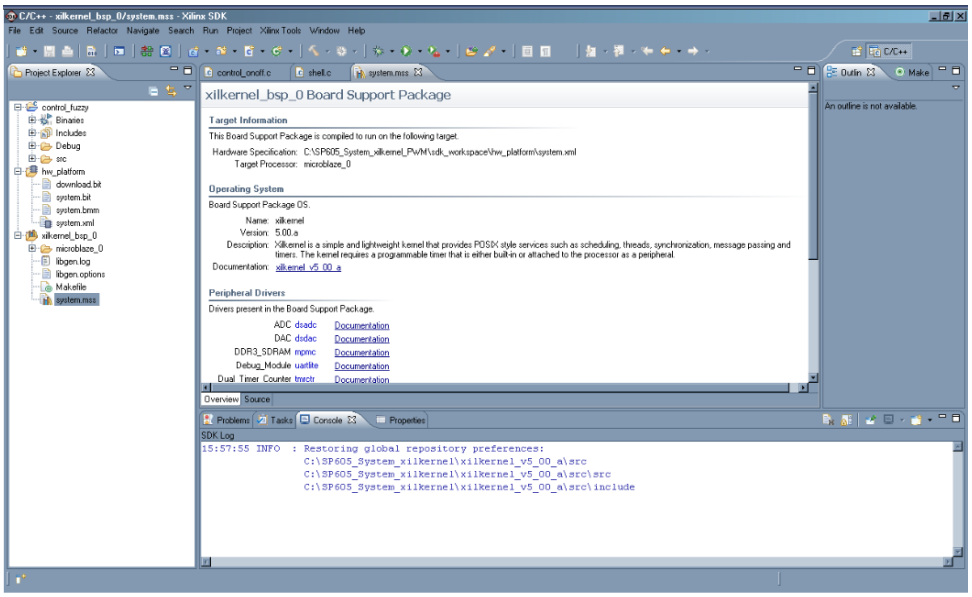

Figura. 3.39. Ventana Principal del SDK

SDK utiliza los diseños creados en XPS para poder generar la plataforma de *software*, la cual contiene un conjunto de *drivers* y librerías que conforma la capa más baja del *software*. Para esto se debe importar los diseños de *hardware*, luego generar el *Board*

Support Package (BSP), y para concluir crear la aplicación en lenguaje C. Los archivos resultantes de cada uno de estos procedimientos se encuentran en carpetas separadas que conforman el *SDK_Workspace*.

DRIVERS

Un controlador de dispositivo, llamado normalmente *driver*, es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del *hardware* y proporcionando una interfaz posiblemente estandarizada para usarlo. Se puede esquematizar como un manual de instrucciones que le indica al sistema operativo, cómo debe controlar y comunicarse con un dispositivo en particular. Por lo tanto, es una pieza esencial, sin la cual no se podría usar el *hardware* en una aplicación de *software*. [66]

En el SDK se puede visualizar la lista de *drivers* correspondientes a cada periférico de la plataforma de *hardware* a través del archivo system.mss del BSP (Figura. 3.40)

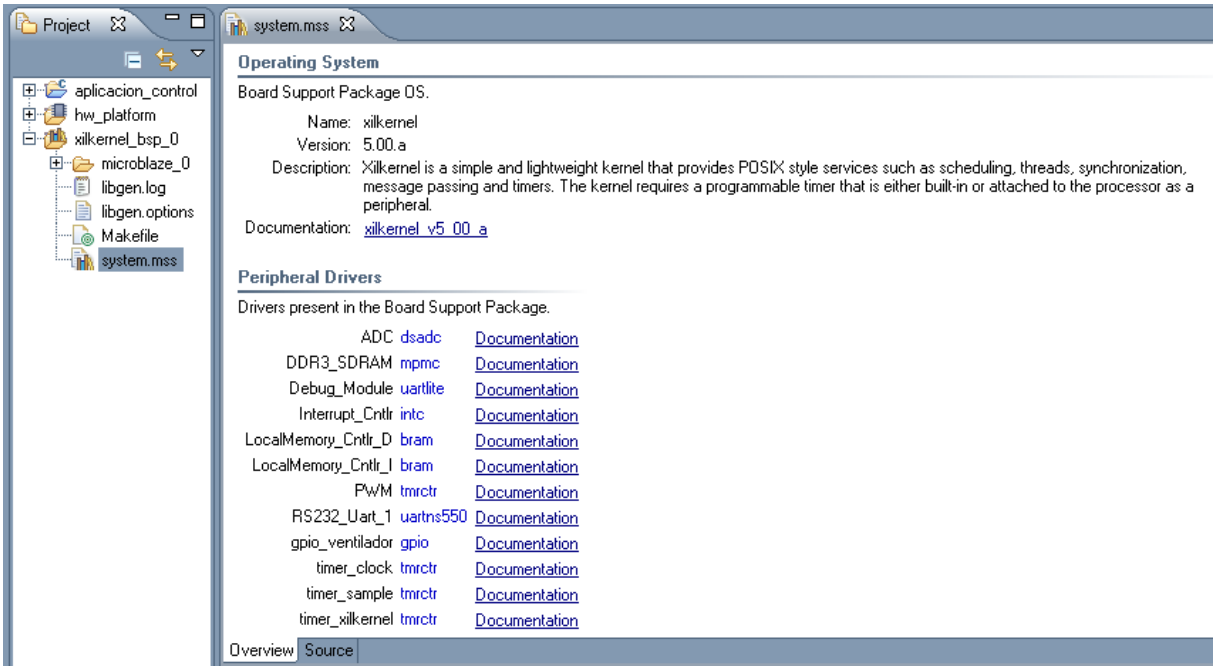

Figura. 3.40. Lista de Drivers en SDK

El archivo *system.mss* proporciona un enlace a la documentación de cada driver donde se describe su proceso de configuración, la descripción de sus funciones, así como sus definiciones. En la siguiente figura se muestra un ejemplo de la documentación del

driver para el IP Core XPS delta-sigma ADC.

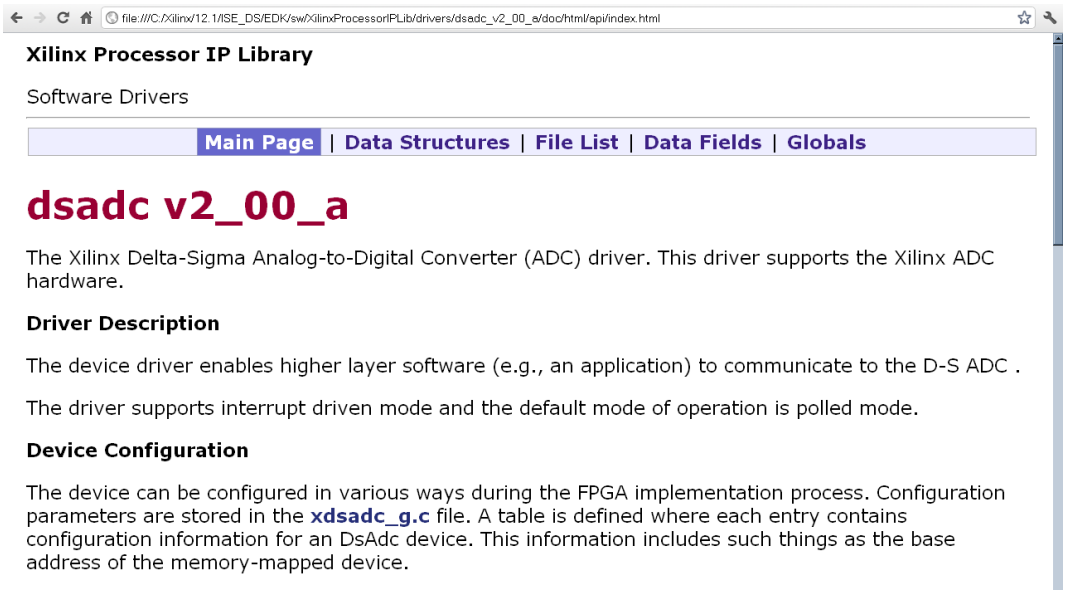

Figura. 3.41. Documentación de drivers para IP Core xps delta-sigma ADC

LIBRERÍAS

En ciencias de la computación, librería es un conjunto de subprogramas utilizados para desarrollar *software*. Las librerías contienen código y datos, que proporcionan servicios a programas independientes, como los *drivers* por ejemplo, es decir, pasan a formar parte de éstos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. [67]

BOARD SUPPORT PACKAGE BSP

Las aplicaciones de *software* deben ejecutarse en la parte superior de una plataforma de *software*, usando una API (*Application Program Interfaces*). Para poder crear y usar aplicaciones de *software* en SDK, se debe crear un proyecto con una plataforma de *software* base, que puede ser un RTOS. El SDK incluye los siguientes dos tipos plataformas de *software*: *Standalone* y *Xilkernel*[41].

[41] **Nota:** Un simple proyecto SDK puede contener más de una plataforma de software. Por ejemplo, usted puede tener una sola plataforma de *software* que está establecida para *Standalone*, y otra para *Xilkemel*.

- ***Standalone:*** Un entorno simple, semi-organizado y de un solo subproceso, que proporciona las características básicas como entradas/salidas estándar y que además permite el acceso a las funciones de hardware del procesador.
- ***Xilkernel***: Un simple y ligero *kernel* que proporciona servicios de estilo POSIX[42], como *scheduling*, hilos, la sincronización, el *message passing*, y temporizadores [58].

Xilkernel proporciona todas las características necesarias para diseñar una aplicación de *software* sobre un Sistema Operativo de Tiempo Real (RTOS). Cabe mencionar que las siguientes definiciones han sido tomadas de los documentos UG758 y UG646 de Xilinx.

XILKERNEL

Xilkernel es un simple pero robusto *kernel* embebido que se puede personalizar en gran medida para un determinado sistema [59], está altamente integrado con el marco de estudio de plataforma. Además, es un *software* libre integrado en el EDK. Xilkernel tiene las principales características de un *kernel* embebido tales como *multi-tasking* (múltiples tareas al mismo tiempo), *priority-driven preemptive scheduling*, comunicación entre procesos, facilidades para la sincronización, y manejo de interrupciones. El *kernel* tiene un diseño modular, donde cada módulo puede ser personalizado (Figura. 3.42).

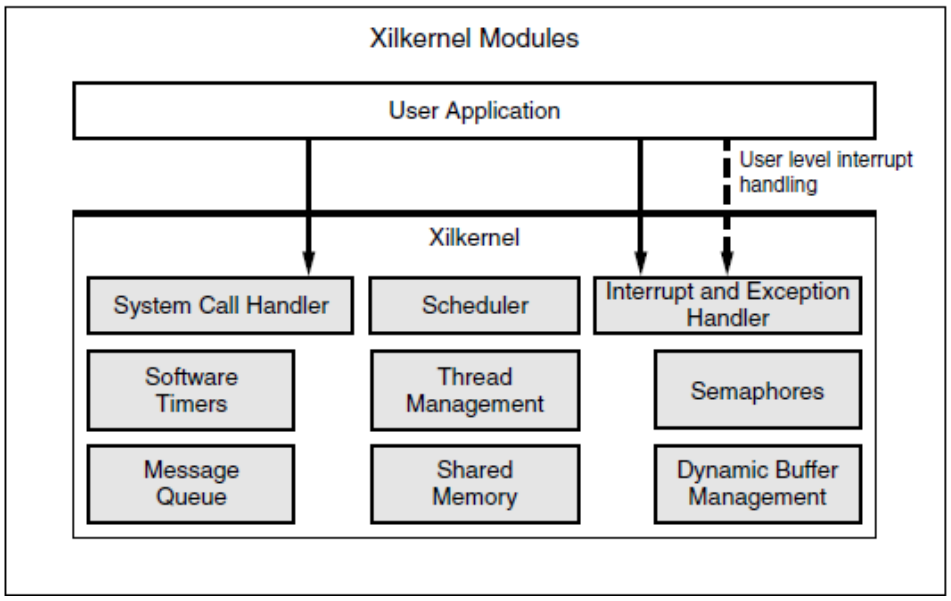

Figura. 3.42. Módulos de Xilkernel

Fuente: XILINX, Inc., 2010

[42] POSIX (*Portable Operating System Interfac*, la X viene de Unix): Es un conjunto de estándares para sistemas operativos basados en UNIX que sugieran con la idea de desarrollar programas que se puedan trasladar a otros sistemas sin recodificar la información [61].

Las aplicaciones de *software* pueden ser desarrolladas por separado en SDK, o como un proyecto de aplicación de *software* en XPS. Después se debe enlazar esta aplicación con la librería de Xilkernel, de esta manera se construirá la imagen final del *kernel*. Esta librería es generada con el nombre de ***libxilkernel.a***

Para vincular este *kernel* en el desarrollo de aplicaciones se debe tomar en cuenta lo siguiente:

- Los archivos de aplicación de código C, deben incluir la librería ***xmk.h***. Para esto se define *#include "xmk.h"*, y con esto se puede acceder a definiciones y declaraciones de GNU que son requeridos por Xilkernel y sus aplicaciones.
- La aplicación de *software* debe estar enlazada con la librería ***libxil.a***. Esta librería contiene las funciones actuales del *kernel* generado.
- Xilkernel es el responsable del manejo en alto nivel de interrupciones y excepciones del procesador *MicroBlaze*.
- El control de la asignación de memoria se la hace a través del *Linker Script* generado automáticamente.
- La aplicación debe estar provista de un *main ( )*, el cual es el punto de partida de la imagen del *kernel*. En el punto donde la configuración de su aplicación se complete, y se desee que el *kernel* se inicie, se debe invocar *xilkernel_main ( )*, esta función está definida en el archivo *main.c*. Conceptualmente la rutina *xilkernel_main ( )*, hace dos cosas: inicializa el kernel vía *xilkernel_init ( )* y *sys_init ( )*, y luego pone en marcha el kernel con *xilkernel_start ( )*.

La primera acción realizada dentro de *xilkernel_init ( )* es la inicialización del *hardware* de soporte del *kernel*. Esto incluye el registro del controlador de interrupciones y la configuración del *timer* del sistema, así como la inicialización de la protección de memoria, en caso de que ser requerida.

La rutina *sys_init ( )* realiza la inicialización de cada módulo en el siguiente orden:

1. Estructuras internas para los *process contexts*.
2. Disposición de colas
3. Módulos *pthread*

4. Módulos de semáforos
5. Módulos de colas de mensajes
6. Módulos de memoria compartida
7. Asignación del modulo de memoria
8. Modulo de temporizadores de software
9. Creación del hilo "*idle task()"*
10. Creación de *pthread* estáticos

A continuación de estos pasos, se invoca a *xilkernel_start( )*, rutina donde se habilitan las interrupciones y excepciones. En seguida el *kernel* entra en un lazo infinito ejecutado por el hilo *idle_tasks()*, seguido por la habilitación del *scheduler*, el mismo que inicializa el proceso de *scheduling*.

Modelo de Procesos de Xilkernel

Los procesos que se ejecutan dentro de Xilkernel son llamados *process contexts*, por ejemplo el *Scheduling* se realiza a nivel de *process contexts*. No existe el concepto de grupo de hilos que forman una combinación, que convencionalmente se llama proceso [59]. En cambio, todos los hilos son iguales y compiten por igualdad de recursos. La interfaces que se manipulan en Xilkernel, *POSIX threads*, por ejemplo, pueden crear, finalizar, y manipular los hilos en las aplicaciones creadas. Para esto se usa identificadores de hilos [60].

Cada *process contexts* se encuentra en cualquiera de los siguientes seis estados:

- *PROC_NEW* – Un proceso de reciente creación
- *PROC_READY* – Un proceso listo para ejecutarse
- *PROC_RUN* – Un proceso que esta ejecutándose
- *PROC_WAIT* – Un proceso que está bloqueado en un recurso
- *PROC_DELAY* – Un proceso que está en espera de un tiempo de espera
- *PROC_TIMED_WAIT* – Un proceso que está bloqueado en un recurso y tiene asociado un tiempo de espera.

Cuando un proceso termina, entra a un estado de finalización llamado *PROC_DEAD*. El diagrama de estados del *process contexts* se muestra en la Figura. 3.43.

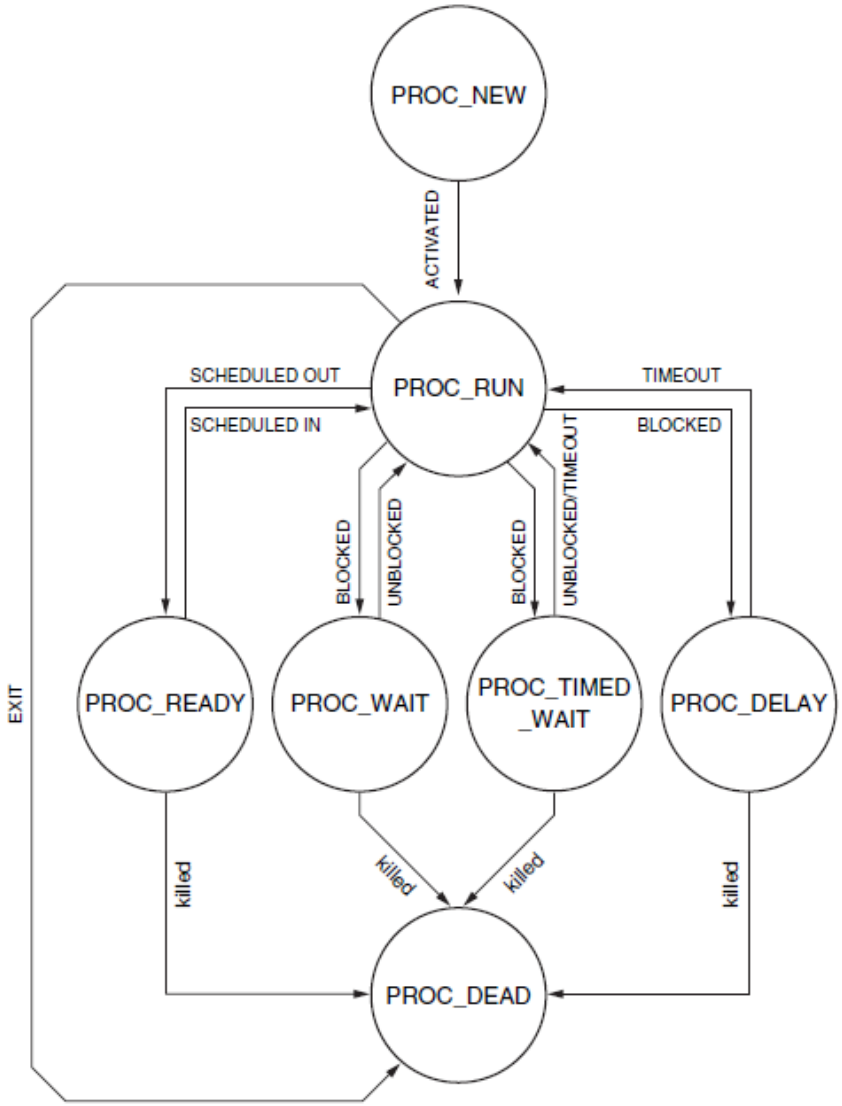

Figura. 3.43. Estados de un *process contexts*

Fuente: XILINX, Inc., 2010

Requerimientos de Hardware de Xilkernel

Algunos servicios en el *kernel* requieren soporte desde el *hardware*, y Xilkernel ha sido diseñado para funcionar tanto con el *IP Core "fit_timer"* o con el "*xps_timer/counter"*. Xilkernel es capaz de inicializar y utilizar estos *timers* automáticamente, pero es necesario especificar valores como el *system timer frecuency* y el *system timer interval*. Estos parámetros deben ser editados en el archivo *system.mss* del BSP. A continuación se detalla un ejemplo de los valores de estos parámetros:

 # MicroBlaze system timer device specification
PARAMETER systmr_spec = true
PARAMETER systmr_interval = 100
PARAMETER systmr_freq = 100000000
PARAMETER systmr_dev = xps_timer_1

Xilkernel ha sido también diseñado para trabajar en escenarios, que involucran múltiples interrupciones. Usando el IP Core *xps_intc* se puede manejar las interrupciones de *hardware* y alimentar una simple línea IRQ que va desde el controlador hasta el procesador. En el archivo *system.mss* se especifica el nombre del periférico controlador de interrupciones como se muestra a continuación:

 # Specification of the intc device
PARAMETER sysintc_spec = Interrupt_Cntlr

API DE XILKERNEL

Xilkernel proporciona una interface POSIX para el *kernel*, cabe recalcar que no todos los conceptos de interfaces definidas por POSIX están disponibles, pero las que se encuentran disponibles son suficientes para ejecutar aplicaciones sobre diferentes plataformas.

A continuación se enlista las API que puede manejar Xilkernel, en un proyecto:

- Administración de Hilos (*Thread Management*)
- Semáforos (*Semaphores*)
- Colas de Mensajes (*Message Queues*)
- Memoria Compartida (*Shared Memory*)
- Bloqueos Mutex (*Mutex Locks*)
- Administración del Buffer de Memoria Dinámica (*Dynamic Buffer Memory Management*)
- Temporizadores de Software (*Software Timers*)
- Manejo de interrupciones con APIs a nivel de usuario (*User-Level Interrupt Handling APIs*)

Thread Management

Xilkernel soporta la API de hilos básica de POSIX. La creación de hilos y la manipulación se la realiza con notación estándar POSIX. Como ya se mencionó los hilos son diferenciados por un identificador único. El identificador de hilos es de tipo ***pthread_t***, el mismo que corresponde únicamente a un hilo en una operación especifica [60].

Semáforos

Un semáforo es una variable especial de tipo abstracto, que constituye el método clásico para restringir o permitir el acceso a recursos compartidos. Por ejemplo un recurso de almacenamiento, variables de código fuente, o para el trabajo en un entorno de multiprocesamiento en el que se ejecutan varios procesos concurrentes [62].

Xilkernel admite semáforos asignados POSIX que pueden ser utilizados para la sincronización. Los semáforos POSIX cuentan por debajo de cero, es decir, un valor negativo indica el número de procesos bloqueados en el semáforo. El número de semáforos localizados en el *kernel* y la longitud de las colas de espera del semáforo pueden ser configurados durante la inicialización del sistema [60], ver la Tabla. 3.14 para ver la configuración de los parámetros del módulo semáforo.

Atributo	Descripción	Tipo	Valor Por Defecto
config_sema	necesidad del módulo semáforo	Booleano	false
max_sem	Número máximo de semáforos	Numérico	10
max_sem_waitq	Longitud del semáforo en la cola de espera	Numérico	10
config_named_sema	Configuración del Semáforo nombrado, que soporta el kernel	Booleano	false

Tabla. 3.14. Parámetros del Modulo Semáforo

Fuente: XILINX, Inc., 2010

Colas de Mensajes (*Message Queue*)

Las colas de mensajes ofrecen un protocolo asíncrono de comunicaciones, lo que significa que el emisor y el receptor del mensaje no tienen que interactuar con la cola de mensajes al mismo tiempo. Los mensajes colocados en la cola se almacenan hasta que el destinatario los recupera [63].

Xilkernel soporta en su *kernel* las colas de mensajes XSI (*X/Open System Interface*), esta XSI es un conjunto de interfaces opcionales bajo el estándar POSIX. Las colas de mensajes pueden tomar un tamaño arbitrario. El número de estructuras de colas de mensajes asignadas en el *kernel* y la longitud de las colas de mensajes pueden ser configuradas durante la inicialización del sistema [60]. El módulo para implementar una cola de mensajes es opcional y se lo puede configurar de acuerdo a la Tabla. 3.15.

Atributo	Descripción	Tipo	Valor Por Defecto
config_msgq	Necesidad del módulo Message Queue	Boolean	false
num_msgqs	Número de colas de mensajes en el sistema	numérico	10
msgq_capacity	Número máximo de colas de mensajes	numérico	10
use_malloc	Proporcionado para colas de mensajes más poderosas las cuales utilizan malloc y free para asignar memoria para los mensajes	boolean	false

Tabla. 3.15. Parámetros del Módulo Message Queue

Fuente: XILINX, Inc., 2010

Cabe mencionar que las funciones *malloc( )* y *free( )*, se utilizan para asignar y liberar espacio para los mensajes.

Memoria Compartida (*Shared Memory*)

La memoria compartida es un método por el cual el programa puede intercambiar datos con mayor rapidez, con el fin de no utilizar los servicios comunes de lectura y escritura del sistema. Para poner los datos en la memoria compartida, el cliente obtiene acceso después de comprobar el valor de un semáforo, escribe los datos, y luego se reinicia

el semáforo para indicarle al servidor que los datos están a la espera. A su vez, el servidor, escribe los datos en el área de la memoria compartida, y utiliza el semáforo para indicar que los datos están listos para ser leídos [64].

Un aspecto importante, es que el servidor comprueba periódicamente si en la memoria compartida hay posibilidad de entrada de nuevos datos. El modelo cliente – servidor es un buen ejemplo de uso de memoria compartida para el intercambio de datos. La configuración de este módulo es opcional y puede se detalla en la Tabla. 3.16.

Atributo	Descripción	Tipo	Valor Por Defecto
config_shm	Necesidad del módulo de memoria compartida	booleano	False
shm_table	Tabla de memoria compartida. Definida como un arreglo con cada elemento teniendo el parámetro shm_size	arreglo de 1-tuplas[43]	Ninguno
shm_size	Tamaño de la memoria compartida	numérico	Ninguno
num_shm	Número de memorias compartidas expresadas como el tamaño del arreglo shm_table	numérico	Ninguno

Tabla. 3.16. Parámetros del Módulo Memoria Compartida

Fuente: XILINX, Inc., 2010

Bloqueos Mutex (*Mutex Locks*)

Los bloqueos o exclusiones mutuas (*mutex*) son normalmente utilizados para serializar el acceso a una sección de código que no puede ser ejecutada simultáneamente por más de un hilo. Un objeto *mutex* solamente permite un hilo en una sección controlada, obligando a otros hilos que tratan de acceder a esta sección, a esperar hasta que el primer hilo haya salido de esa sección [65].

En Xilkernel, este es un mecanismo de sincronización que puede utilizarse junto con la API ***pthread***. El número de bloqueos *mutex* y la longitud de la cola de espera del

[43] **Tuplas**: Es la representación de una fila en una de las tablas que se está almacenando datos, en donde la fila tendrá un número limitado de objetos.

bloqueo *mutex* puede ser configurado durante las especificaciones del sistema. [60]. Este módulo también es opcional y puede ser configurado de acuerdo a la Tabla. 3.17.

Atributo	Descripción	Tipo	Valor Por Defecto
config_pthread_mutex	Necesidad del módulo pthread mutex	Booleano	False
max_pthread_mutex	Número máximo de bloqueos pthread mutex	Numérico	10
max_pthread_mutex_waitq	Longitud de cada bloqueo mutex en la cola de espera	Numérico	10

Tabla. 3.17. Parámetros del Módulo *Pthread Mutex*

Fuente: XILINX, Inc., 2010

Administración del Buffer de Memoria Dinámica

El *kernel* proporciona un esquema de asignación del buffer de memoria que puede ser utilizado por aplicaciones que necesiten de una asignación dinámica. Las rutinas de asignación rechazan segmentos de un grupo de memoria, y el usuario transfiere al administrador del buffer de memoria estos segmentos. En esta administración del buffer de memoria dinámica se puede crear grupos de memoria, también se puede especificar estáticamente los diferentes tamaños de bloques o segmentos de memoria, y el número de bloques requeridos por una determinada aplicación [60].

Al igual que los anteriores, este módulo también es opcional y puede ser configurado de acuerdo a la Tabla. 3.18, durante la inicialización del sistema.

Atributo	Descripción	Tipo	Valor Por Defecto
config_bufmalloc	Necesidad del administrador del buffer de memoria	booleano	False
max_buf	Número máximo de grupos de buffer que pueden ser administrados, de semáforos en cualquier momento por el kernel	numérico	10

Atributo	Descripción	Tipo	Valor Por Defecto
mem_table	Tabla del bloque de memoria, este está definido como un arreglo con cada elemento teniendo parámetros como mem_bsize, mem_blks	arreglo de 2- tuplas	Ninguno
mem_bsize	Tamaño del bloque de memoria en bytes	numérico	Ninguno
mem_nblks	Número de bloques de memoria de un mismo tamaño	numérico	Ninguno

Tabla. 3.18. Parámetros del Módulo de Administración de Memoria

Fuente: XILINX, Inc., 2010

Temporizadores de Software

Xilkernel proporciona las funcionalidades de los temporizadores de *software*, empleados para medir el tiempo de procesamiento de la información, observar en la Tabla. 3.19 los detalles de configuración de este módulo.

Atributo	Descripción	Tipo	Valor Por Defecto
config_time	Necesidad de temporizadores de software y módulos de administración de tiempo	Booleano	false
max_tmrs	Número máximo del temporizador de software en el kernel	Numérico	10

Tabla. 3.19. Parámetros del Módulo de Temporizador de Software

Fuente: XILINX, Inc., 2010

Para cada uno de estos módulos existen APIs que permiten su manejo, para un mayor detalle de cada una de estas funciones, consultar la referencia bibliográfica de Xilinx, documento UG646 Xilkernel (v 5.00.a) [60]. En esta referencia se encontrará la descripción de los parámetros, el valor que devuelve cada función, su descripción, y donde está incluida.

3.4 DISEÑO DE REFERENCIA

El *MicroBlaze Processor Subsystem*, es una plataforma completa, que sirve como diseño de referencia de un *System on Chip*. Está basado en el procesador *MicroBlaze* y varios *IP Cores* que se comunican directamente con los periféricos externos de la tarjeta SP605, así como algunos *IP Cores* utilizados para conexiones internas. La finalidad de este proyecto es mostrar varias características funcionales de la tarjeta de evaluación SP605, y de esta manera poder realizar aplicaciones embebidas disminuyendo el tiempo de diseño.

En este diseño de referencia se encuentra una aplicación de *software stand-alone* para testear la tarjeta SP605 y la imagen para ejecutar una aplicación de *software webserver* (servidor web), sobre el RTOS *Petalinux*.

Los archivos que contienen las plataformas de *hardware* y *software* de *MicroBlaze Processor Subsystem*, vienen como parte de la información proporcionada en la memoria de almacenamiento USB del SPARTAN 6 FPGA EMBEDDED KIT, donde además constan los tutoriales de *hardware* y *software*, necesarios para la compresión del funcionamiento de este SoC. La estructura de directorio de estos archivos se muestra en la Figura 3.44.

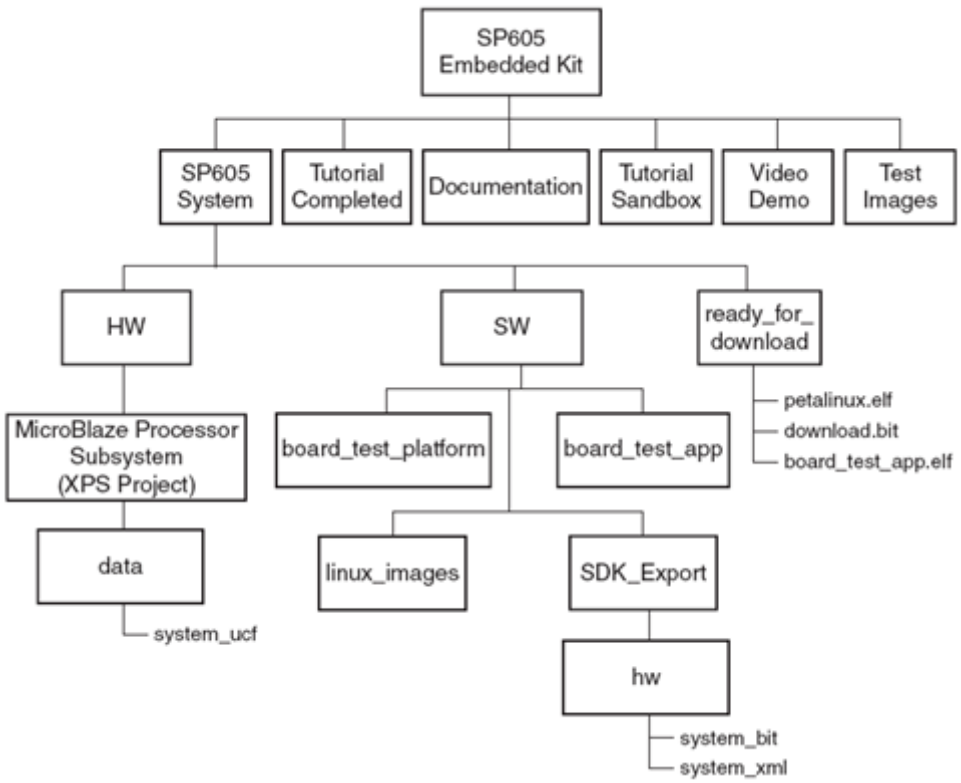

Figura. 3.44. Estructura del Sistema de Directorio

Plataforma de *Hardware* del Sistema

El SP605 *MicroBlaze Processor Subsystem* puede ser tratado así mismo como un componente SoC (Figura. 3.45), ya que posee un procesador, buses, memorias on-chip, e interfaces con los dispositivos periféricos, elementos que muestra la siguiente figura.

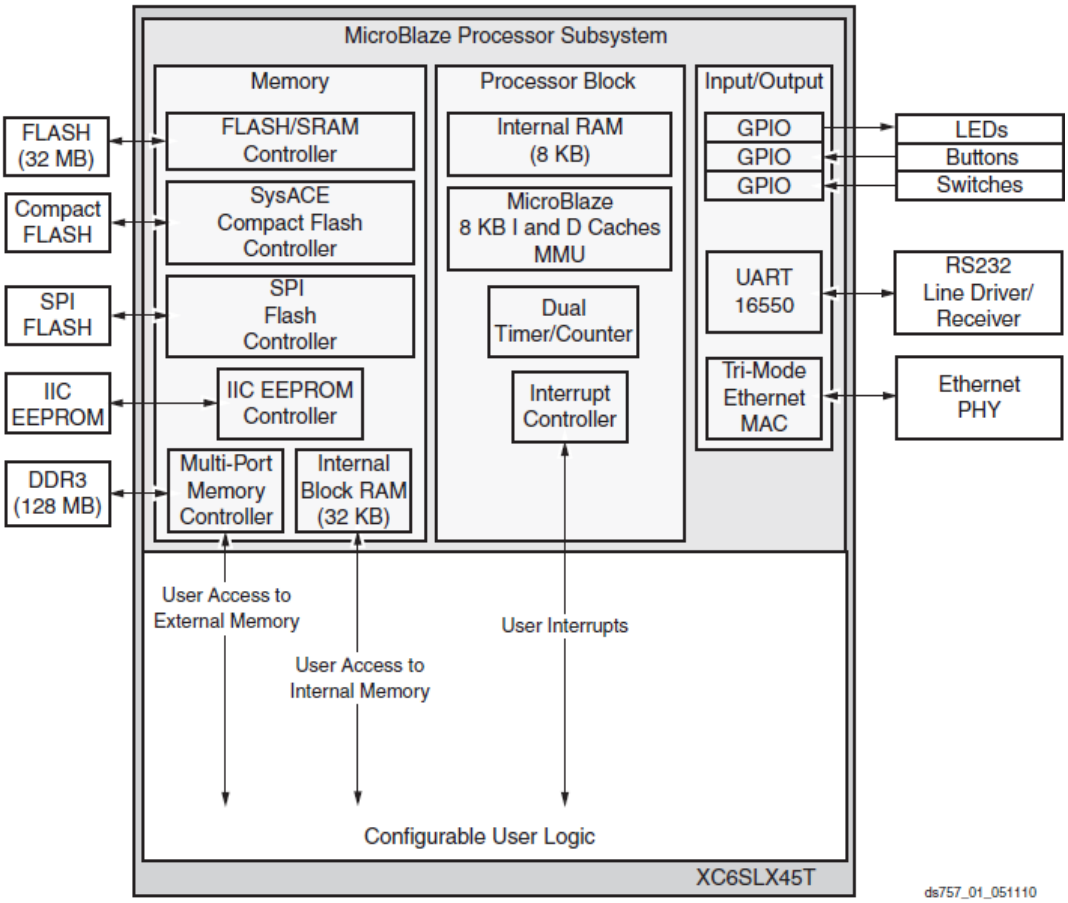

Figura. 3.45. Diagrama de Bloques del *SP605 MicroBlaze Processor Subsystem*

Este sistema es implementado en un FPGA *Spartan*-6 XC6SLX45TFGG484-3 utilizando la versión 12.1 de ISE *Design Suite Embedded Edition*.

La ventaja de implementar este sistema dentro de un FPGA es que puede ser fácilmente expandido y personalizado. Las instrucciones para modificar esta plataforma pueden ser encontradas en el Capítulo 4.

Componentes del *MicroBlaze Processor Subsystem*

En la Figura. 3.45, se mostraron los *IP Cores* que componen el sistema, en tres grupos principales Bloque de Procesador, Memoria, y Entradas / Salidas. A continuación se recalcan las principales características de cada uno de ellos.

Bloque Procesador

- Procesador *MicroBlaze* de 32 bits con 8 KB de cache de Instrucciones y 8 KB de cache de Datos.
    - *Hardware Barrel Shifter*
    - *Memory Management Unit* (MMU)
- 8 KB de memoria RAM interna.
- Controlador de interrupciones
- *Dual Timer/Counter* de 32 bits.

Memorias

- SDRAM - DDR3 de 128 MB, 16-bit, 333.5 MHz (667 Mb/s)
- *Internal Block* RAM - 32 KB
- *Linear (Parallel) Flash* - 32 MB
- *Quad SPI Flash* - 8 MB
- *Compact Flash Card* - 2 GB
- IIC EEPROM - 1 KB
- *Multi-Port Memory Controller* (MPMC) con un puerto disponible para la interface lógica de usuario para memoria SDRAM DDR3 externa.

Entradas / Salidas

- Tres controladores de E/S de propósito general

- 4-bit LED
- 4-bit *push button*
- 4-bit DIP *Switch*

- 16550 UART

  - Configuración por Software de velocidad de transmisión, ancho de datos, paridad y bits de parada.

- 10/100 /1000 Mb/s Tri-mode Ethernet MAC (TEMAC)

  - GMII *Interface* a PHY
  - *Scatter-Gather* DMA

Configuración del Sistema

La configuración detallada de los parámetros del sistema y de las configuraciones de puertos, para cada componente del *MicroBlaze Processor Subsystem* se encuentra en el archivo *SP605_Embedded_Kit \ SP605_System \ SW \ SDK_Export \ hw \ system.html.* A continuación se muestra un resumen de estas configuraciones.

Mapa de Direcciones

La siguiente tabla muestra la asignación de direcciones de memoria para cada *IP Core*. Estas direcciones fueron establecidas por defecto por Xilinx para este sistema, se recomienda que si se realiza alguna modificación, se generen nuevamente las direcciones mediante XPS y se tome en cuenta los cambios.

Instancia	Periférico	Dirección Base	Dirección Superior
LocalMemory_Cntlr_D	lmb_bram_if_cntlr	0x00000000	0x00001FFF
LocalMemory_Cntlr_I	lmb_bram_if_cntlr	0x00000000	0x00001FFF
Internal_BRAM	xps_bram_if_cntlr	0x41A00000	0x41A07FFF

Instancia	Periférico	Dirección Base	Dirección Superior
DDR3_SDRAM	mpmc - esta región de memoria es cacheable	0x50000000	0x57FFFFFF
	mpmc – SDMA	0x84600000	0x8460FFFF
FLASH	xps_mch_emc	0x7C000000	0x7DFFFFFF
Soft_TEMAC	xps_II_temac	0x81000000	0x8107FFFF
Push_Buttons_4bit	xps_gpio	0x81400000	0x8140FFFF
LEDs_4bit	xps_gpio	0x81420000	0x8142FFFF
DIP_Switches_4bit	xps_gpio	0x81440000	0x8144FFFF
IIC_EEPROM	xps_iic	0x81600000	0x8160FFFF
Interrupt_Cntlr	xps_intc	0x81800000	0x8180FFFF
Dual_Timer_Counter	xps_timer	0x83C00000	0x83C0FFFF
SPI_FLASH	xps_spi	0x83400000	0x8340FFFF
SysACE_CompactFlash	xps_sysace	0x83600000	0x8360FFFF
RS232_Uart_1	xps_uart16550	0x84000000	0x8400FFFF
Debug_Module	Mdm	0x84400000	0x8440FFFF

Tabla. 3.20. Mapa de Direcciones del *MicroBlaze Processor Subsystem*

Fuente: XILINX, Inc., 2010

Consideraciones de los relojes del Sistema

Este sistema funciona a una frecuencia de referencia de 200 MHz cuya fuente es el cristal en la tarjeta SP605. El procesador *MicroBlaze* y el bus de sistema funcionan a 100 MHz, mientras que la memoria DDR3 funciona a 333.5 Mhz. Más detalles acerca de la configuración de los relojes del sistema puede ser encontrada en el archivo *system.html*, en la sección *clock_generator_0*.

Reinicio (*Reset*) del Sistema

Las señales de reinicio del *MicroBlaze Processor Subsystem* son generadas a partir de la entrada de reinicio activada en alto de la tarjeta SP605. Esta señal es filtrada y

sincronizada con el reloj del sistema. El sistema genera secuencialmente señales de reinicio para las estructuras que lo componen en un orden determinado, dejando entre cada señal 16 ciclos de reloj.

1. Señal de reinicio de estructuras de bus.
2. Aquella de periféricos.
3. Reinicio de CPU.

Las señales de reinicio para el *MicroBlaze Processor Subsystem* son generadas por el *Proc_Sys_Reset_IP_core*. Posee una entrada de *reset* externa, la cual como ya se mencionó, está activada en alto y al presionarse resetea los componentes internos del *MicroBlaze* así como otros elementos asociados, tal es el casodel bus principal del sistema. Para más detalles sobre la configuración de este *IP Core* revisar la sección *proc_sys_reset_0* del documento *system.html*, que se lo encuentra en la carpeta ***report*** (en este directorio se guarda el proyecto).

Configuración del Procesador MicroBlaze

El procesador *MicroBlaze*, al ser el componente principal del *MicroBlaze Processor Subsystem*, es configurado de tal manera que se logre la mayor optimización posible del rendimiento total del sistema con un *barrel shifter* y una MMU[44].

El propósito principal del *barrel shifter* es el de desplazar o rotar una palabra de datos de cualquier número de bits en un simple ciclo de reloj. Esta acción es requerida por varias operaciones claves, como la generación de direcciones y las funciones aritméticas.

La MMU está configurada en modo virtual con dos zonas de protección de memoria. En modo virtual, la MMU controla el direccionamiento de direcciones efectivas a direcciones físicas y apoya a la protección de memoria. El modo virtual provee un gran control sobre la protección de memoria. La MMU proporciona protección a memoria y relocalización lo cual es útil para ambientes multi-tarea. Estos ambientes multi-tarea dan la sensación de ejecutar múltiples programas simultáneamente [34].

[44] MMU: Unidad de Administración de Memoria

Prioridad de Interrupciones

Las interrupciones son manejadas por el *core* controlador de interrupciones. La Tabla. 3.21 muestra las interrupciones internas generadas en el sistema embebido y su prioridad.

Para detalles acerca de la configuración del controlador de interrupciones revisar la sección *Interrupt_Cntlr* del archivo *system.html.*

Prioridad	Señal	Fuente	Descripción
Alta	MPMC_SDRAM_0_SDMA2 _Tx_IntOut	SDMA Modulo de Puerto en MPMC	Transmisión Completa de Interrupción de la Scatter-Gather DMA
	MPMC_SDRAM_0_SDMA2 _Rx_IntOut	SDMA Modulo de Puerto en MPMC	Recepción Completa de Interrupción de la Scatter-Gather DMA
	Soft_TEMAC_TemacIntc0 _Irpt	TEMAC	Condición de Interrupción en el TEMAC ha ocurrido como lo indicado en el registro TEMAC Interrupt Status
	IIC_EEPROM_IIC2INTC _Irpt	XPS_IIC	Condición de Interrupción en el Controlador IIC ha ocurrido como lo indicado en el registro IIC Interrupt Status
	SPI_FLASH_IP2INTC_Irpt	XPS_SPI	Condición de Interrupción en el Controlador SPI ha ocurrido como lo indicado en el registro SPI Interrupt Status
	SysACE_CompactFlash_ SysACE_IRQ	XPS_SYSACE	Paso a través de la interrupción del controlador externo System ACE
	Dual_Timer_Counter_Interrupt	XPS_TIMER	En el Modo Generar, indica que el contador se volcó, En el Modo de Captura, el evento de interrupción es el evento de captura.
Baja	RS232_Uart_1_IP2INCT_Irpt	XPS_UART16550	La Condición de Interrupción en el UART 16550 ha ocurrido como lo indicado en el registro UART Interrupt Identification

Tabla. 3.21. Prioridad de Interrupciones en el *MicroBlaze Processor Subsystem*

Fuente: Xilinx Inc., 2010

Dual Timer / Counter

El *XPS Timer Core* está configurado para proveer dos *timers* de 32 bits.

Controlador de Memoria Multi-Puerto (MPMC, *Multi-Port Memory Controller*)

El MPMC está configurado para proporcionar cuatro puertos bidireccionales de 32 bits para acceder a la memoria externa DDR3. La configuración de puertos del MPMC se muestra en la Tabla. 3.22.

Puerto	Ancho de dato	Conexiones
0	32	Acceso a la cache de Instrucciones y Datos de MicroBlaze
1	32	Bus PLBv46 para el periféricos maestros en el sistema embebido
2	32	Motor DMA utilizado para el TEMAC
3	32	Desconectado: Disponible para conexiones desde la fábrica o IP embebidos maestros

Tabla. 3.22. Configuración del Puerto MPMC

Fuente: XILINX, Inc., 2010

Controlador SysACE

El controlador *System ACE Compact Flash*, está configurado con una interface de datos MPU[45] de 8 bits hacia el controlador externo *Xilinx ACE CF* (XCCACE). La interface *CompactFlash* soporta una partición máxima de 2GB[46].

Controlador IIC EEPROM

El controlador IIC (*Inter-Integrated Circuit*), también conocido como I^2C, es un bus de comunicaciones en serie, que posee tres líneas; una para datos, otra para reloj, y otra para tierra, soporta direccionamiento de 7 o 10 bits y contiene FIFOs de transmisión y recepción de 16 bytes. Este controlador puede ser configurado en dos modos de operación, modo de operación estándar (100KHz), o modo de operación veloz (>100KHz a 400KHz).

[45] MPU: Unidad de Múltiples Procesos

[46] Si se desea utilizar la memoria *Compact Flash*, revisar el documento DS080 *System ACE CompactFlash Solution*.

En el *MicroBlaze Processor Subsystem*, el controlador IIC es usado para interactuar con la memoria IIC EEPROM y trabajar en modo estándar con direccionamiento de 7 bits.

Controlador FLASH

Flash, es una tecnología de almacenamiento, derivada de la memoria EEPROM que permite la lectura y escritura de múltiples posiciones de memoria, en la misma operación. El *XPS MCH EMC IP core* es usado para trabajar con el dispositivo lineal externo Flash (16bits, 256Mb Numonyx StrataFlash dispositivo JS28F256P30T). Este dispositivo Flash provee 32MB de almacenamiento no volátil, el cual puede ser usado para configuración o bien como almacenamiento de *software*. Después de la configuración, este puede ser accedido mediante el controlador *XPS MCH EMC FLASH*.

Controlador SPI FLASH

SPI (*Serial Peripheral Interface*), es un estándar de comunicaciones, utilizado principalmente para transferencia de información entre circuitos integrados. El controlador SPI FLASH está conectado a la Flash Serial externa de 64Mb *Winbond* con 4 interfaces SPI (W25Q64BV). Este controlador es utilizado para una comunicación directa con este dispositivo externo.

GPIO (*General Purpose Input - Output*)

El *XPS GPIO IP Core* es utilizado para entradas y salidas de propósito general, cuenta con 32 bits, y en este sistema permite que el sistema embebido controle y acceda a los *push buttons* (*Push_Buttons_4Bit*), DIP *swiches* (*DIP_Switches_4Bit*), y a los LEDs (*LED_4Bit*).

UART (*Universal Asynchronus Receiver-Transmiter*)

El *UART core*, es empleado para establecer la comunicación serial, cabe destazar que este *core* está configurado para trabajar con interrupciones. Los parámetros de velocidad, bits de datos, y paridad son controlados vía *software* por equipo con el que se comunique.

Configuración ETHERNET

El *TEMAC*[47] está configurado para trabajar en una interface *GMII/MII PHY* y contiene FIFOs de transmisión y recepción de 4KB. En encendido o en reinicio, el PHY sobre la tarjeta está configurado para operar en modo GMII con dirección física establecida en 00111. El TEMAC puede correr a 10Mb por segundo, 100Mb por segundo, o 1000Mb por segundo dependiendo de la red a la que esté conectado.

Aplicación de *Software* y *Board Support Package* (BSP)

El SP605 *MicroBlaze Processor Subsystem* contiene una aplicación de *software* (*Board_Test_App*) y su BSP configurado como *Stand-Alone*, y usado para ejecutar un test de memorias y periféricos de la tarjeta SP605. En el Capítulo 4 se mostrará como ejecutar y modificar la aplicación de software y el BSP.

Board_Test_App

La aplicación de software *Board_Test_App* es un simple programa que trabaja con la mayoría de las funciones de la tarjeta SP605. En este programa, se puede escoger entre una lista de opciones para realizar el test de periféricos y memorias. La realización del tutorial de *hadware* (anexo A2) contiene la ejecución de este programa.

Casos de Usos del *MicroBlaze Processor Subsystem*

El *MicroBlaze Processor Subsystem* es la plataforma base para sistemas embebidos en muchos modelos de uso. La Figura. 3.46 muestra un ejemplo del *MicroBlaze Processor Subsystem* usado en un dominio industrial y de sistemas de control. Entre los principales *IP Cores* destacan Industrial Ethernet, EtherCAT, PROFINET, SERCOS II, UART (RS 232, RS 485), CAN, SPI, entrada analógica, TIMER/COUNTER para la realización de algoritmos de control PID utilizando PWM (Modulación por Ancho de Pulso), así como el uso de ciertos elementos comunes; BRAM, GPIO para entradas y salidas a nivel industrial así como indicadores LCD/LED y controlador de Memorias (DDR2, DDR3).

[47] **TEMAC:** Acrónimo para Tri-Mode Ethernet Media, hace referencia a sus tres velocidades (10, 100, 1000 Mb/s)

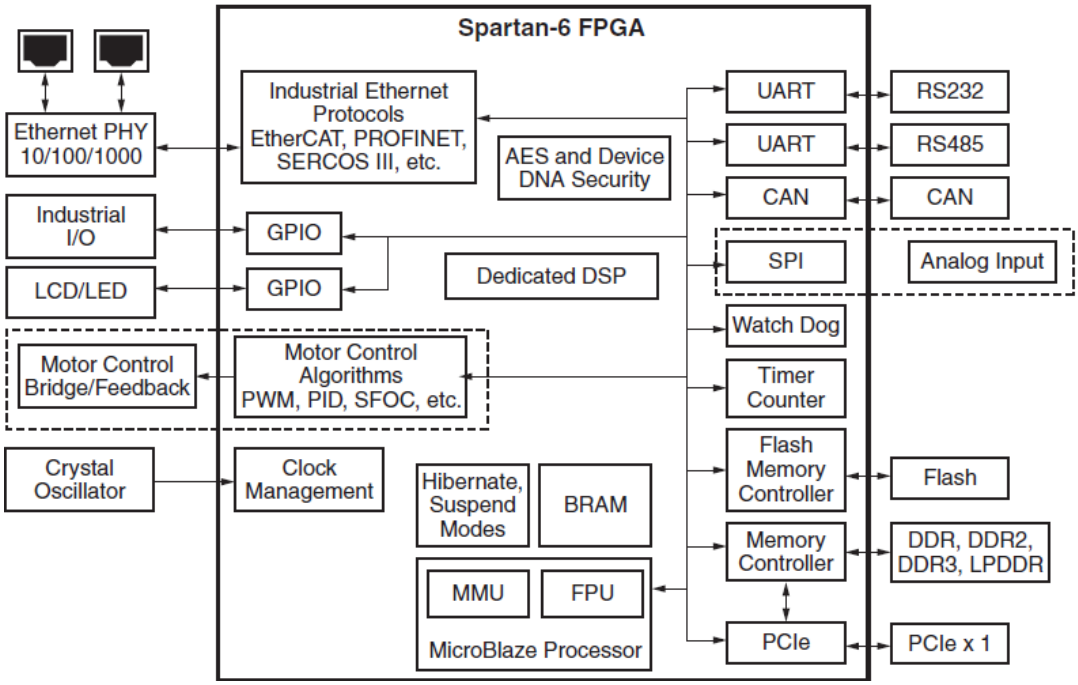

Figura. 3.46. Subsistema con Procesador MicroBlaze en un dominio Industrial

Fuente: XILINX, Inc., 2010

CAPÍTULO 4

TUTORIALES Y GUIAS DE ESTUDIO

Antes de comenzar con el diseño de un SoC utilizando el XILINX SPARTAN-6 FPGA EMBEDDED KIT y continuar con este capítulo, se recomienda completar los tutoriales que se encuentran en los Anexos A2: Tutorial de Hardware Embebido, y A3: Tutorial de Software Embebido. Estos tutoriales explican de manera detallada el uso de las herramientas XPS y SDK, analizadas en el Capítulo 3.

4.1 PROCEDIMIENTO PARA GENERAR UN PROYECTO BASADO EN LOS TUTORIALES DE *HARDWARE* Y *SOFTWARE*

Una vez desarrollados los tutoriales de *hardware* y *software*, se tendrá una visión general del funcionamiento del XILINX SPARTAN-6 FPGA EMBEDDED KIT, y la experticia suficiente en el manejo de las herramientas XPS y SDK, a fin de comprender los pasos para el diseño de un SoC utilizando la herramienta EDK

A continuación se presenta una explicación resumida y abreviada de los tutoriales mencionados. Para mayor detalle dirigirse a los anexos A2 y A3. El procedimiento de diseño de un SoC basado en los tutoriales de *hardware* y *software* es el siguiente:

- Inducción al *MicroBlaze Processor Subsystem*.
- Personalización del *Microblaze Processor Subsystem* (XPS).
- Asignación de pines del FPGA Spartan 6 en el archivo UCF (XPS).
- Generación del *bitstream* de la plataforma de *hardware* (XPS).
- Depuración de la plataforma de *hardware* (XMD – *ChipScope Pro*).
- Exportación de la plataforma de *hardware* al SDK (XPS).

- Creación de un *Workspace* (SDK).
- Importación de la plataforma de *hardware* (SDK).
- Creación y Configuración del *Board Support Package* (SDK).
- Creación o Importación de una aplicación de *software* (SDK).
- Depuración de la aplicación de *software* (SDK).

Inducción al MicroBlaze Processor Subsystem

El MicroBlaze Processor Subsystem constituye un SoC que puede ser utilizado como base para el diseño de sistemas sobre la tarjeta SP605. Para la inducción a este SoC se siguen los siguientes pasos:

- Inicie el XPS:
  - En *Linux*, ingrese *xps* en el *command prompt*.
  - En *Windows*, seleccione ***Inicio → Todos los programas → Xilinx ISE Design Suite 12.x → EDK → Xilinx Platform Studio***
- Seleccione ***Open existing Project*** y dé click en ***OK***.
- Examine la ruta ***SP605_Embedded_Kit/Tutorial_Sandbox/HW/ MicroBlaze_Processor_SubSystem*** y seleccione ***system.xmp***. Click ***Open.***

Una vez ejecutados estos pasos se desplegará la ventana principal del XPS, mostrando el proyecto *MicroBlaze Processor Subsystem*, con todos los *IP Cores* que incluye así como sus conexiones, como se observa en la Figura. 4.1.

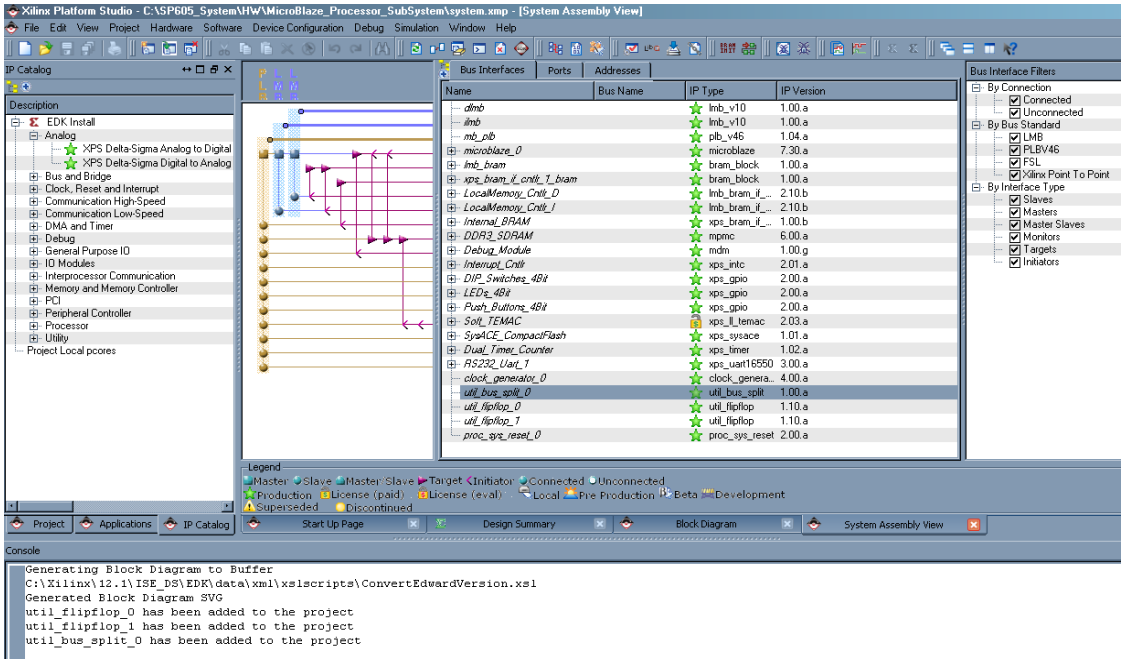

Figura. 4.1. Proyecto *MicroBlaze Processor Subsystem* abierto con XPS.

El diagrama de bloques de la Figura. 4.2 muestra la distribución de *IP Cores* y sus conexiones con los buses del sistema dentro del *MicroBlaze Processor Subsystem*.

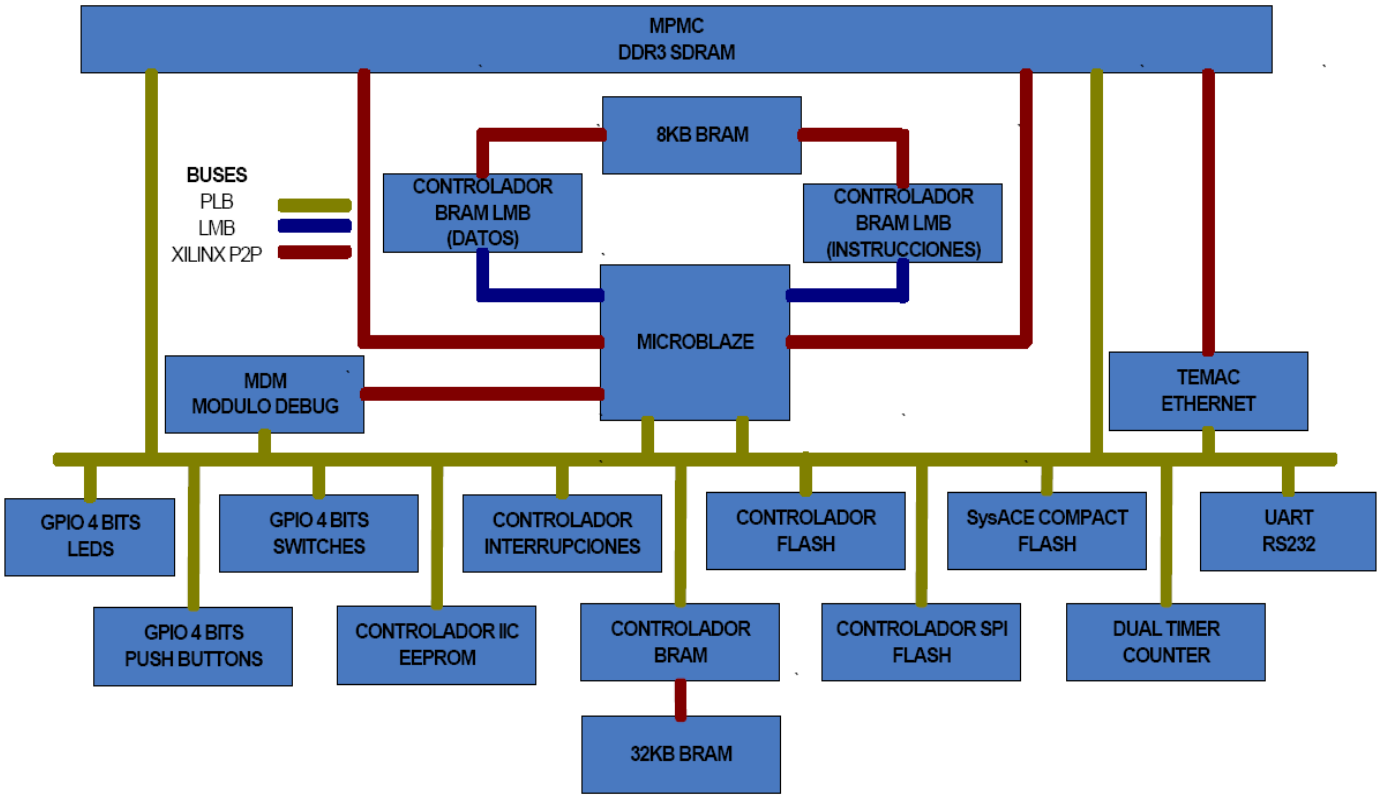

Figura. 4.2. Diagrama de Bloques del MicroBlaze Processor Subsystem.

El elemento principal de este SoC es el procesador de 32 bits *MicroBlaze* (parte central de la Figura. 4.2), el cual gobierna el sistema. Conectados a este procesador se encuentran dos controladores de memoria para el bloque RAM (BRAM), separando la memoria de datos de la memoria de instrucciones (arquitectura Harvard). Los buses que se utilizan en el *MicroBlaze Processor Subsystem*, son: PLB, LMB, y Xilinx P2P (punto a punto). Además, este SoC cuenta con los siguientes IP Cores:

- GPIO
- Controlador IIC EEPROM.
- Controlador de Interrupciones
- Controlador BRAM.
- Controlador Flash
- Controlador SPI Flash
- *Multi-Port Memory Controller* MPMC
- SysACE Compact Flash
- Timer Counter
- UART RS232
- Módulo de Depuración MDM.
- Módulo Ethernet TEMAC

Cabe mencionar que las configuraciónes de estos *IP Cores*, se encuentran detalladas en el Capítulo 3, y que la hoja de datos completa de este sistema se encuentra en el documento de Xilinx DS757 *SP605 Embedded Kit MicroBlaze Processor Subsystem*.

Personalización del *Microblaze Processor Subsystem*

Este paso se realiza en XPS y consiste en añadir o eliminar *IP Cores* (Figura. 4.3), de acuerdo a las especificaciones y requerimientos del diseño.

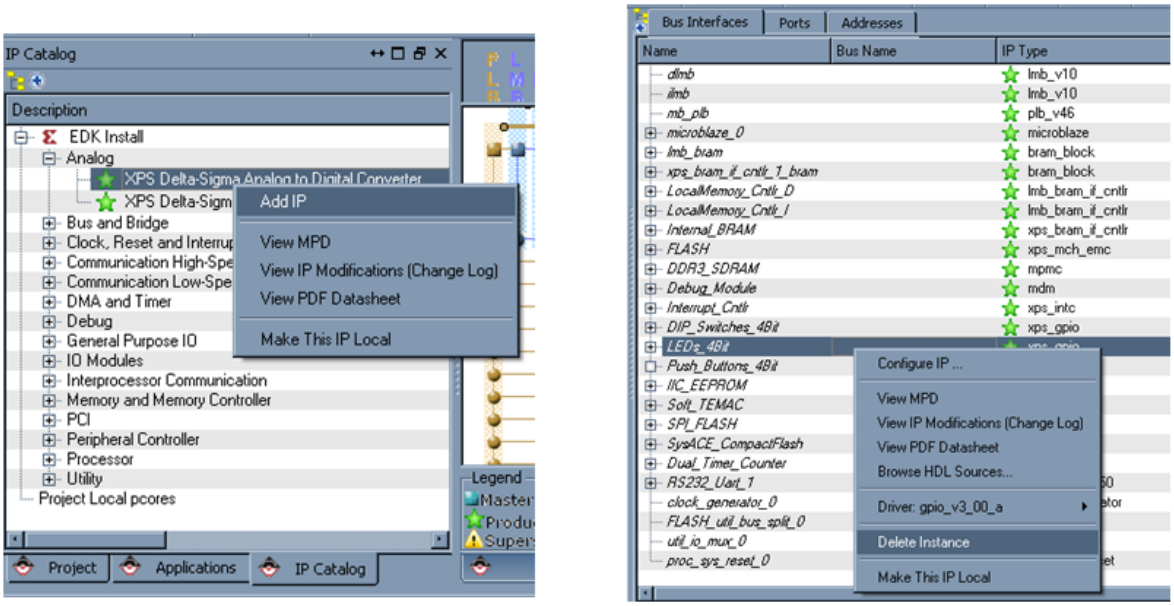

Figura. 4.3. Adición y/o Eliminación de IP Cores.

Al añadir un *IP Core*, siempre es necesario asignar un espacio de memoria, esto se realiza en la pestaña ***Address***[48], como se indica en la Figura. 4.4.

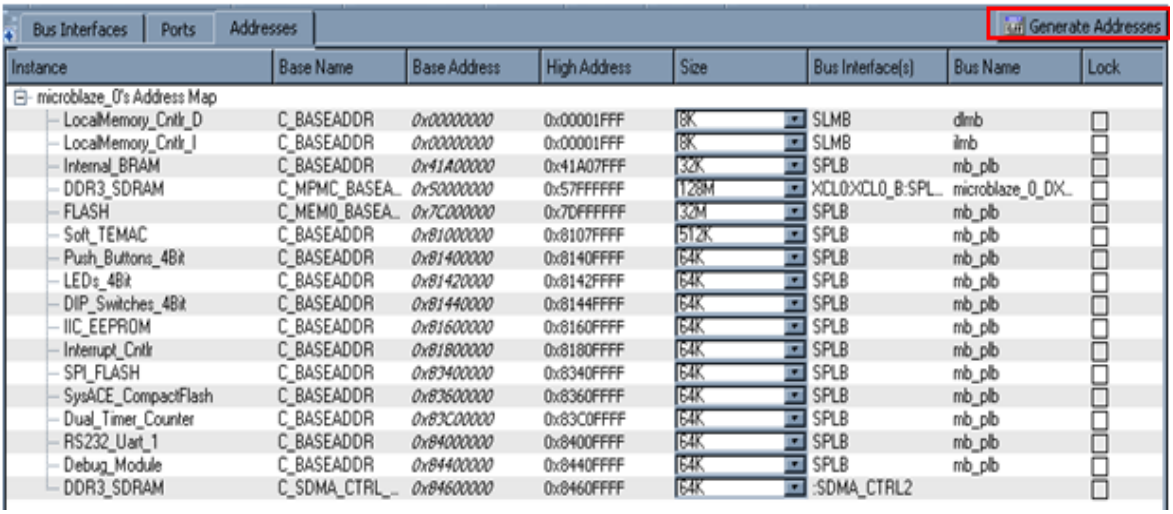

Figura. 4.4. Asignación de Direcciones de Memoria a los IP Cores

Una vez que se ha asignado las direcciones de memoria se conecta los *IP Cores* a los buses del sistema en caso de ser requerido. Estas conexiones se las realiza en la pestaña ***Bus Interfaces*** del SAV (Figura. 4.5).

[48] **Nota:** Se recomienda que en este paso se seleccione el botón que se encuentra en la parte superior derecha de la Figura. 4.4 ***Generate Address.*** **Esto** genera automáticamente las direcciones.

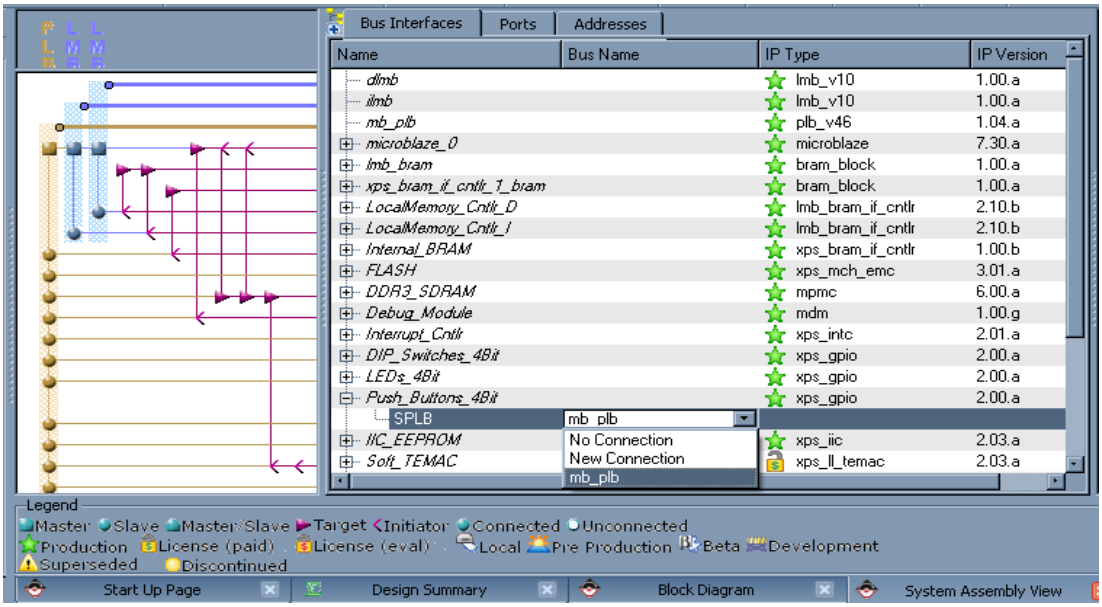

Figura. 4.5. Conexiones de los IP Cores a Buses del Sistema.

A continuación, se realiza la configuración de cada *IP Core* de acuerdo a los requerimientos del diseño. Para esto, señalar el *IP Core* deseado (por ejemplo el GPIO – LEDs_4bits), dar *click* derecho y seleccionar la opción ***Configure IP…***

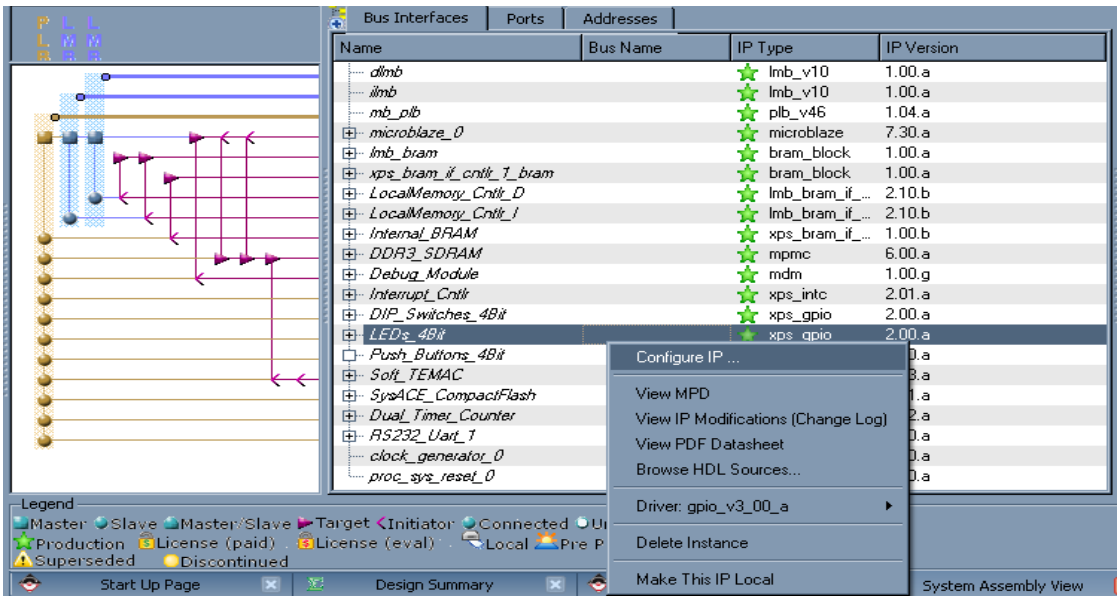

Figura. 4.6. Selección para Configuración de los IP Cores.

Inmediatamente después de haber seleccionado la opción ***Configure IP...***, aparecerá una ventana (Figura. 4.7), en la cual se puede realizar la configuración del *IP Core*.

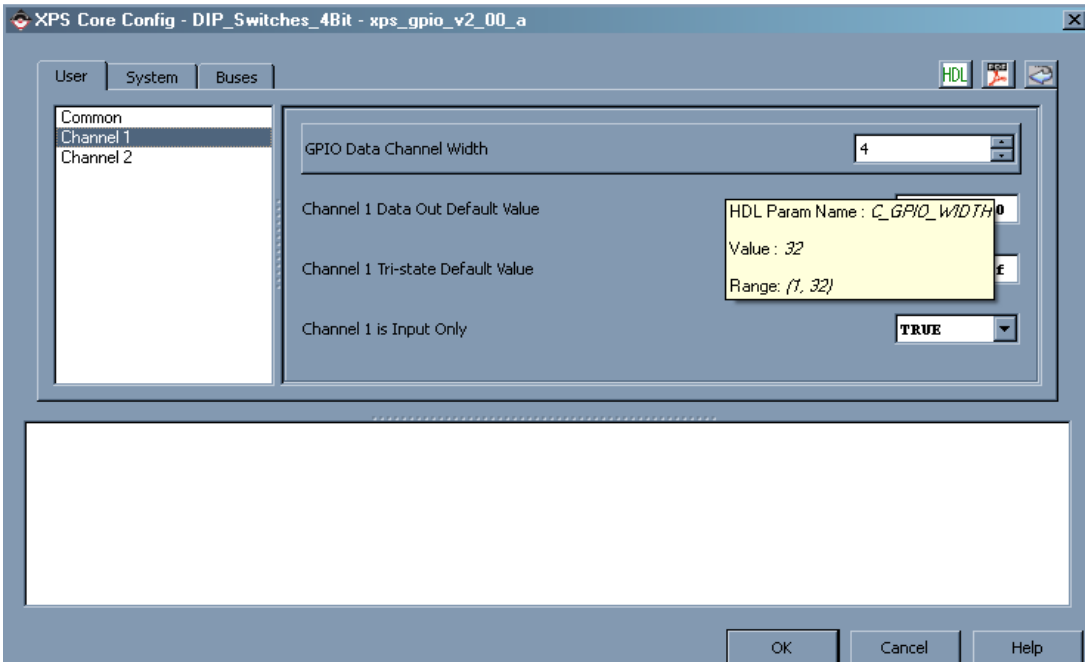

Figura. 4.7. Configuración de los IP Cores.

En el caso del Controlador de Interrupciones, se debe escoger las interrupciones necesarias, y establecer la prioridad de cada una de ellas (Figura. 4.8). La interrupción de menor prioridad se colocará en la parte superior y la de mayor prioridad en la parte inferior.

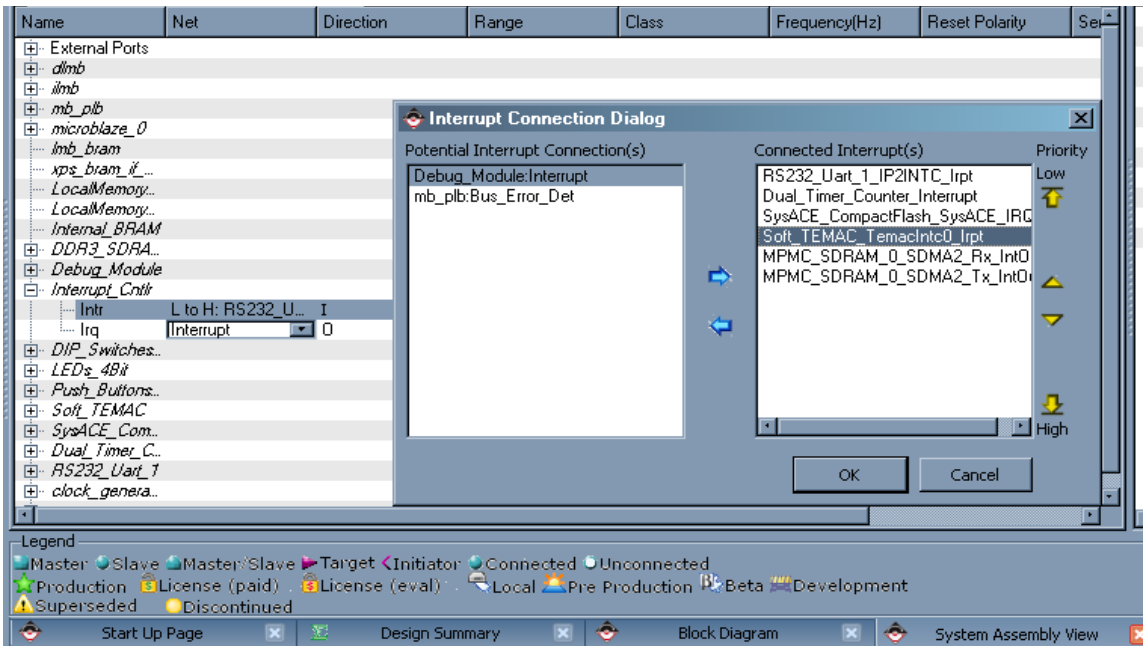

Figura. 4.8. Modificación de Prioridades de las Interrupciones.

Para finalizar con la personalización *Microblaze Processor Subsystem*, se establece las señales externas que los *IP Cores* necesiten. Para esto, dirigirse a la pestaña ***Ports***

(Figura. 4.9), y seleccionar ***Make External*** en la señal que se desee conectar a un pin del FPGA.

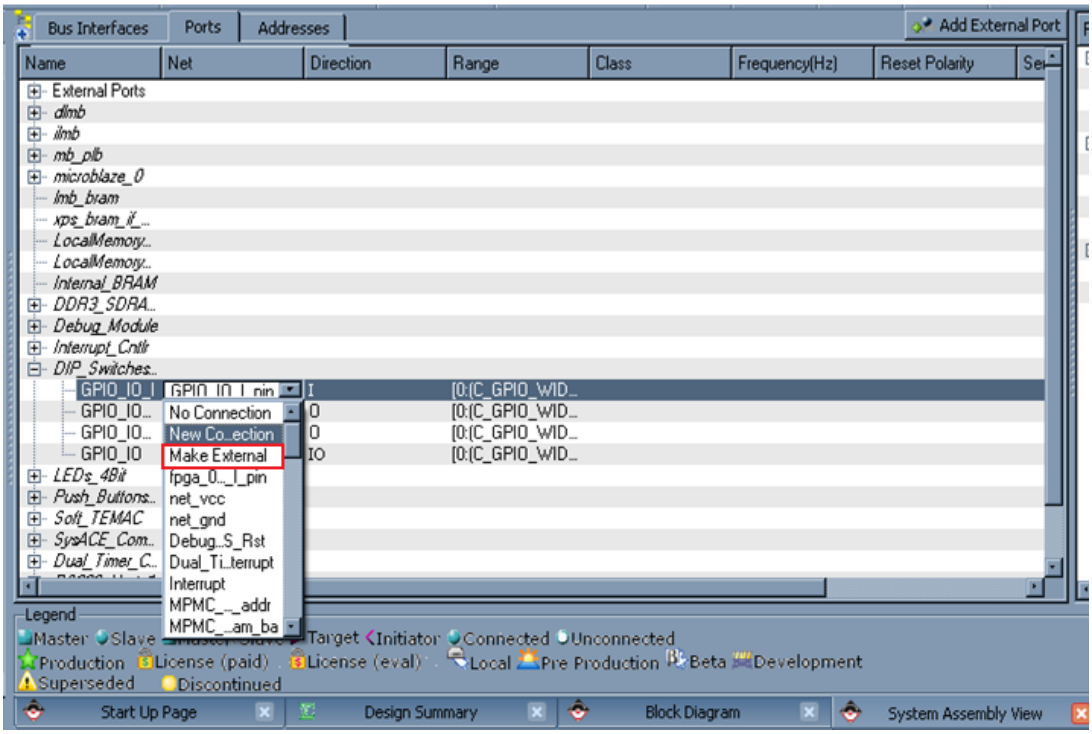

Figura. 4.9. Configuración de señales Internas a Externas

Seguido a esto, se visualizará en la parte superior de la pestaña ***ports*** la sección ***External Ports*** (Figura. 4.10). En esta sección se encuentran las señales que se establecieron como externas. En la parte derecha se visualizará el nombre de la conexión externa, este nombre será usado en el archivo UCF para la asignación de pines. Mientras que, en la parte izquierda se visualizará el nombre de la conexión interna que será usado para la conexión con otros *IP Cores*.

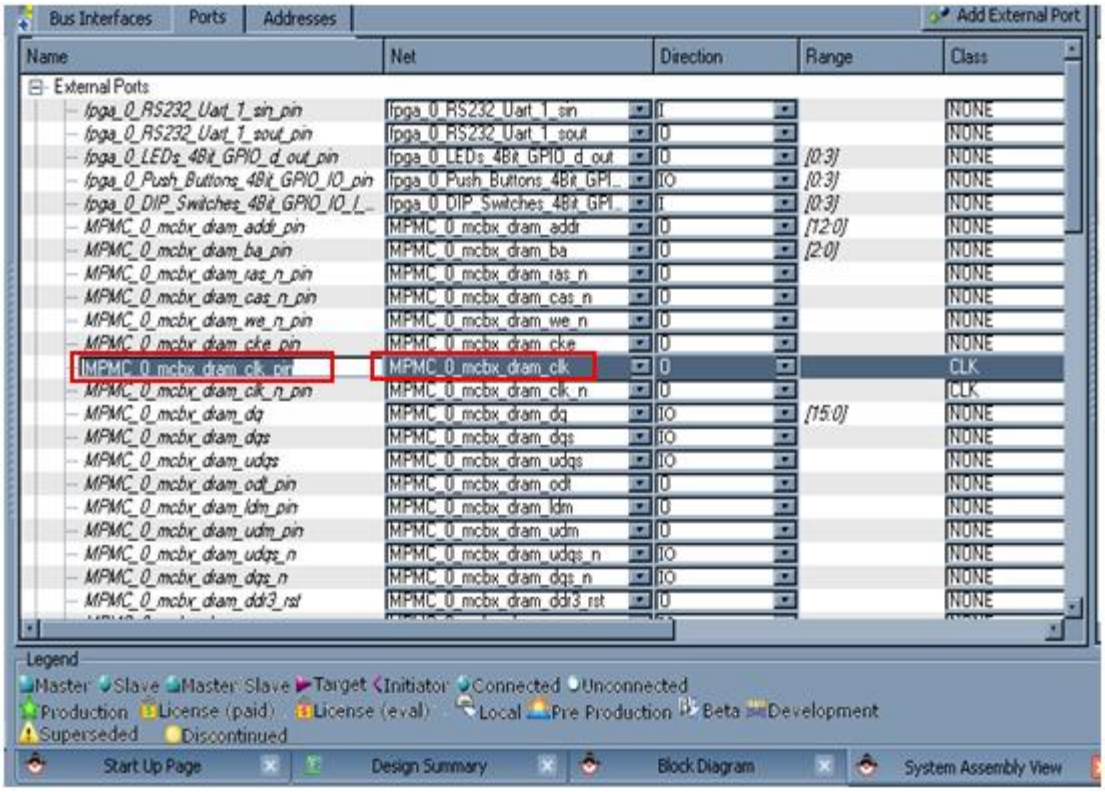

Figura. 4.10. Puertos de señales Externas

Asignación de Pines del FPGA Spartan 6 en el Archivo UCF

Tomando en cuenta las señales que han sido asignadas como externas, es necesario dirigirse a la ventana *Proyect* y seleccionar el archivo UCF, es decir ***Project → Project Files → UCF File: data/system.ucf***. Este archivo muestra las conexiones de las señales externas con los pines físicos del FPGA, teniendo en cuenta consideraciones como: restricción de ubicación, especificaciones de recursos del FPGA, y estándares de entrada/salida.

En la Figura. 4.11 se muestra un ejemplo de las líneas de código que se deben especificar para la asignación de pines del FPGA.

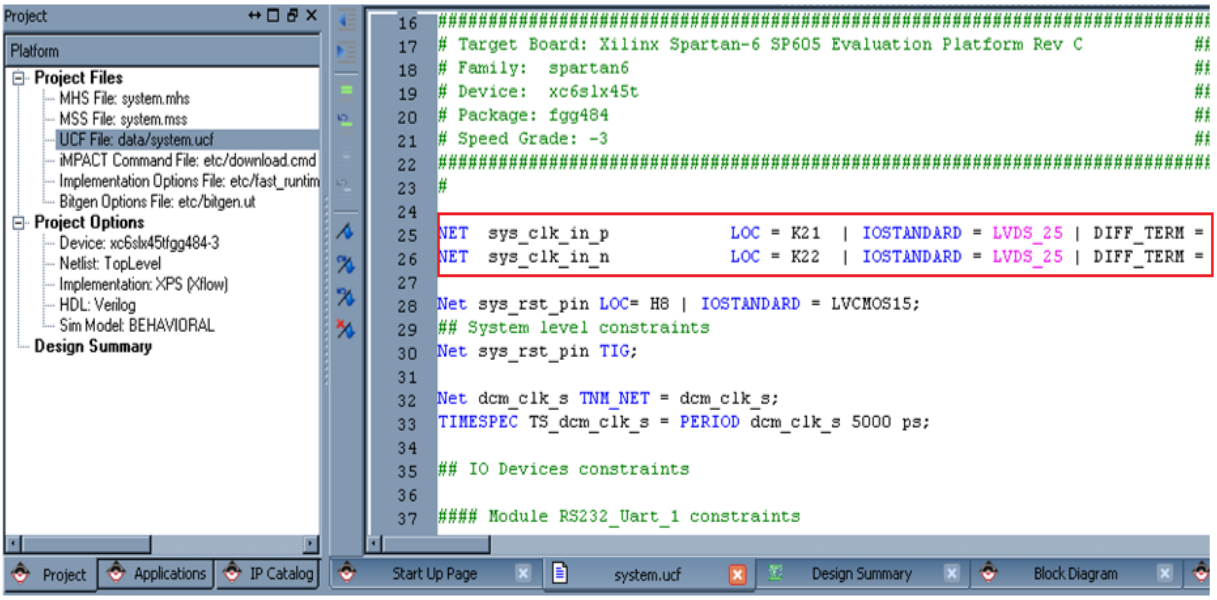

Figura. 4.11. Archivo system.ucf de asignación de pines del FPGA

La estructura de una línea del archivo UCF para asignación de pines se aprecia en el siguiente ejemplo:

NET **DAC_out_pin** LOC = **E21** | IOSTANDAR = **LVCMOS25**;

Donde:

- **DAC_out_pin**, es el nombre de la conexión externa.
- **E21**, es el pin físico del FPGA.
- **LVCMOS25**, es el estándar de E/S y depende del banco donde se encuentra el pin físico del FPGA.

Cabe recalcar que el ejemplo presentado anteriormente funciona para una asignación simple. Si se necesita implementar restricciones adicionales de configuración, síntesis, físicas, lógicas, de *grouping*, *placement* o *timing*, se recomienda revisar el documento de Xilinx UG625 *Constraints Guide*. Además, para obtener más detalles acerca de los bancos del FPGA, estándares de E/S, *buffers* de salida disponibles, y demás recursos de la familia de FPGAs Spartan 6, se recomienda revisar el documento de Xilinx UG381 *Spartan-6 FPGA SelectIO Resources.*

Generación del *bitstream* de la plataforma de hardware.

Como se describió en el Capítulo 3, para generar la Plataforma de *Hardware*, primero se debe generar el ***netlist***. Para esto, dirigirse en XPS a ***Hardware → Generate Netlist***. El siguiente paso es generar el ***bitstream***, para ello diríjase a ***Hardware → Generate Bitstream*** (ver la Figura. 4.12). Para generar el archivo *.bit* que contiene el diseño de *hardware* ISE utiliza herramientas de implementación con el propósito de leer el ***netlist*** y el archivo UCF del proyecto.

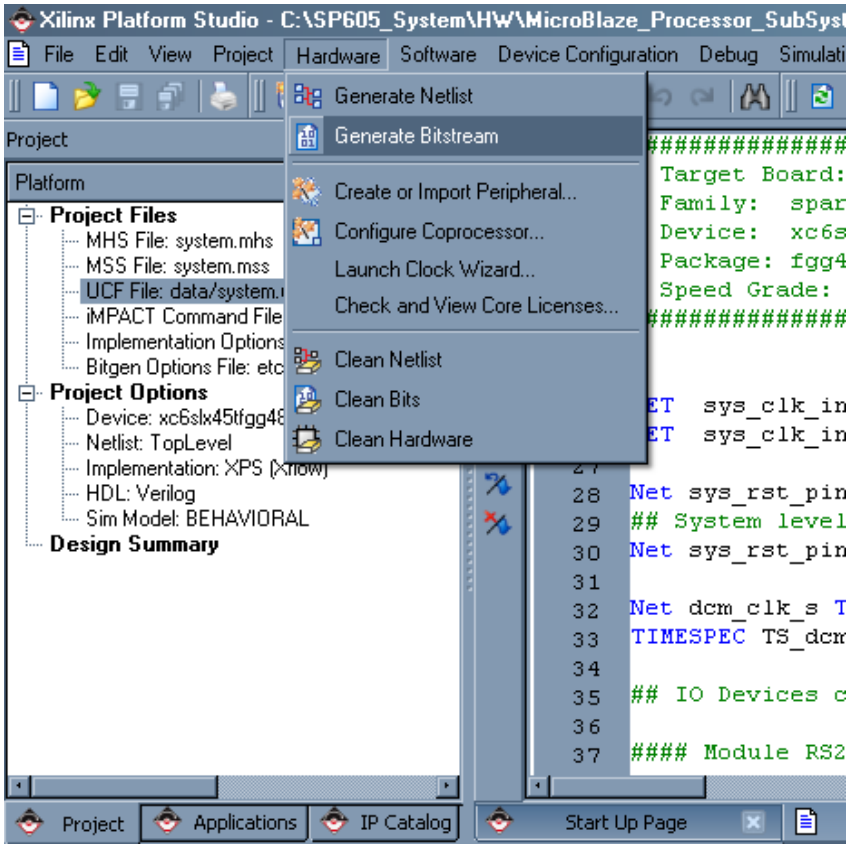

Figura. 4.12. Selección para la generación del *bitstream*.

Depuración de la Plataforma de *Hardware*

La depuración de la plataforma de *hardware* no siempre será necesaria, ya que los *IP Cores* utilizados son preprobados y verificados. Se recomienda la realización de este paso cuando se presenten los siguientes casos:

- Se han añadido bloques creados por el usuario.

- Se ha implementado lógica interna que necesita ser comprobada.
- Se necesita comprobar la comunicación con componentes externos (memorias, *displays*, etc.).
- El diseño presenta problemas de *timing*.
- Se requiere verificar el diseño usando entradas y salidas virtuales.

Para realizar la depuración de *hardware*, existen dos herramientas; *Xilinx Microprocessor Debugger* y *ChipScope Pro*.

Xilinx Microprocessor Debugger (XMD)

El XMD permite la visualización a bajo nivel del funcionamiento de los bloques de la plataforma de *hardware*, a través de la escritura y lectura de sus registros internos (Figura. 4.13). El XMD está ubicado en **Inicio → Todos los programas → Xilinx ISE Design Suite 12.x → EDK → Tools → Xilinx Bash Shell** [49].

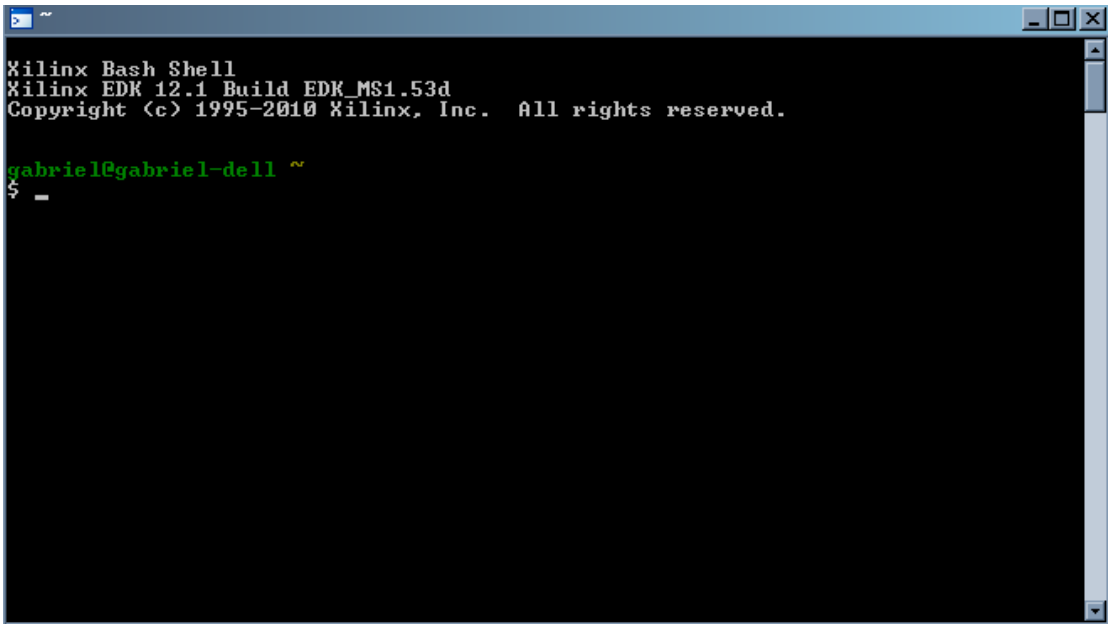

Figura. 4.13. Xilinx Bash Shell.

A continuación se muestra un ejemplo donde se comprueba la comunicación con una memoria externa.

[49] **Nota:** Se debe conocer la carpeta en la cual se está actualmente ejecutando el XMD, para lo cual se utilizará los comandos como *dir* o *cd*.

$xmd

XMD% fpga –f download.bit *(Descarga bitstreamal FPGA)*

XMD% connect mb mdm *(Conecta JTAG al IP Core MDM)*

XMD% mwr 0x41A00000 0x12345678 *(Escribe en la memoria externa)*

XMD% mrd 0x41A00000 *(Lee memoria externa)*

El valor 0x12345678 debería ser devuelto de la dirección de memoria 0x41A0000: 41A00000: 0x12345678

ChipScope Pro

La herramienta *ChipScope Pro*, es un *software* que proporciona una variedad de *IP Cores* para el análisis y depuración de un diseño [46]. Conectando adecuadamente estos *IP Cores*, se puede monitorear cualquier o todas las señales del diseño. En la Figura. 4.14 se observa los bloques del *ChipScope Pro* que se utilizan generalmente en la depuración de un sistema.

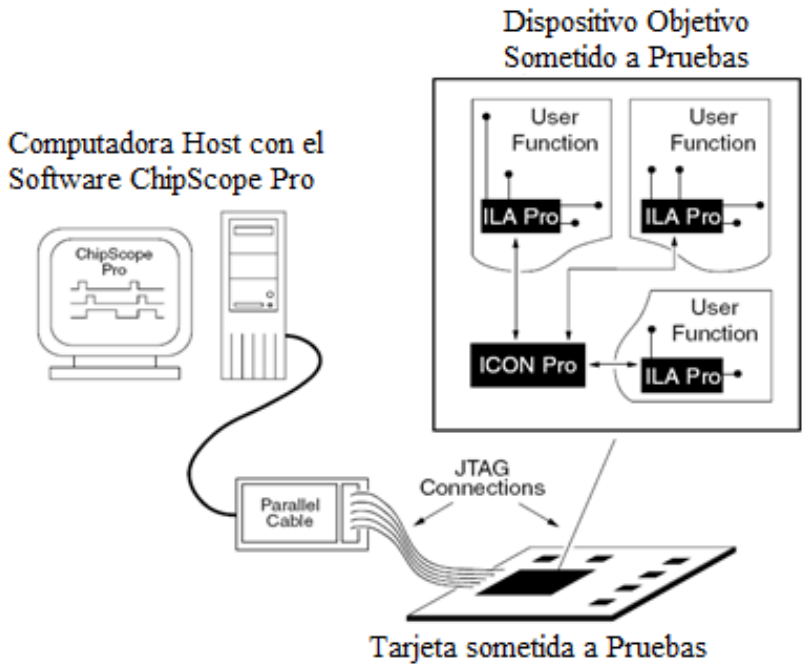

Figura. 4.14. Cores del ChipScope Pro

Basado en: XILINX, Inc., 2010

El XPS proporciona un asistente para añadir un Controlador Integrado (ICON, *Integrated Controller Core*), un Analizador Integrado Lógico (ILA, *Integrated Logic*

Analyzer), y un Analizador Integrado de Bus (IBA, *Integrated Bus Analyzer*).

El *IP Core* ICON ofrece una vía de comunicación mediante JTAG, entre la PC que contiene el software *ChipScope* y los módulos *ChipScope* en el diseño. El *IP Core* IBA, proporciona conexiones definidas que permiten supervisar el bus PLB dentro del diseño. El *IP Core* ILA es utilizado para supervisar señales internas del diseño, se puede pensar en ILA tal como si fuese un osciloscopio digital quese puede integrar en el diseño para ayudar en la depuración [46].

Para mayor detalle acerca de el uso de esta herrameinta de depuración dirigirse al documento de Xilnx UG029 *ChipScope Pro 12.1 Software and Cores*.

Exportación de la Plataforma de Hardware al SDK

Una vez generada la plataforma de h*ardware*, se procede a exportarla al *Software Development Kit*, para poder desarrollar el *software* del sistema. Para ello, dirigirse a ***Project → Export Hardware Design to SDK…*** (Figura. 4.15).

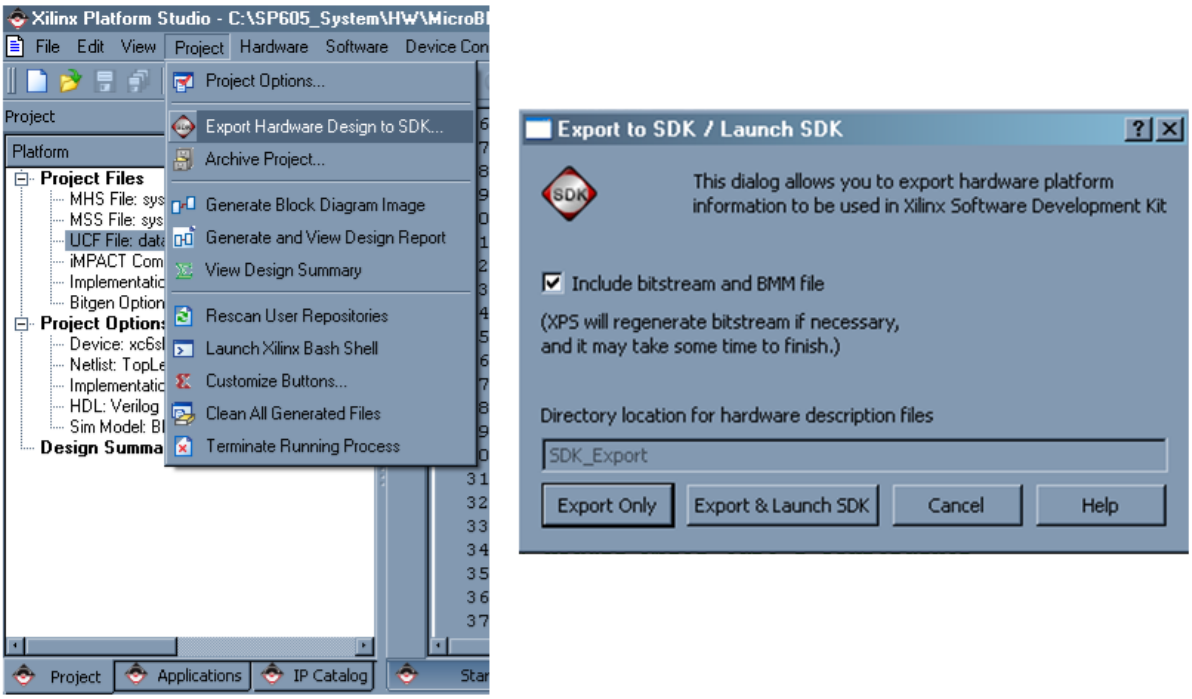

Figura. 4.15. Exportación de la Plataforma de Hardware.

Hay que tomar en cuenta incluir el *bitstream* de la plataforma de *hardware* y, el archivo BMM (*Block RAM Memory Map*). Para esto *check* en *Include bitstream and BMM file*. A continuación, pulsar ***Export Only***, para generar el archivo ***system.xml***, que contendrá la información de la plataforma de *hardware* requerida por el SDK, con el fin de poder realizar la aplicación de *software*.

Creación de un *Workspace* en SDK

Un *workspace* es un directorio en el sistema de archivos, que utiliza SDK con la finalidad de mantener la información sobre el diseño de *hardware*, los archivos de software, y los registros [47]. En general, un *workspace* maneja los directorios que contienen la plataforma de *hardware*, el BSP, y la plataforma de *software*.

Para crear un nuevo *workspace*:

1. Inicie el SDK. En *Windows*, seleccione ***Inicio → Todos los Programas → Xilinx ISE Design Suite 12.1 → EDK → Xilinx Software Development Kit***. En *Linux*, ingrese el comando ***xsdk*** en el símbolo del sistema.
2. Cuando aparezca *Workspace Launcher*, especifique el SDK Workspace como ***SP605_Embedded_Kit/Tutorial_Sandbox/SW/SDK_Workspace***. Click en ***OK*** en el *Workspace Launcher*, ver en la Figura. 4.16.
3. Cuando el SDK aparezca, una ventana de bienvenida, se desplegará, cierre está ventana después de navegar a través de la información mostrada.

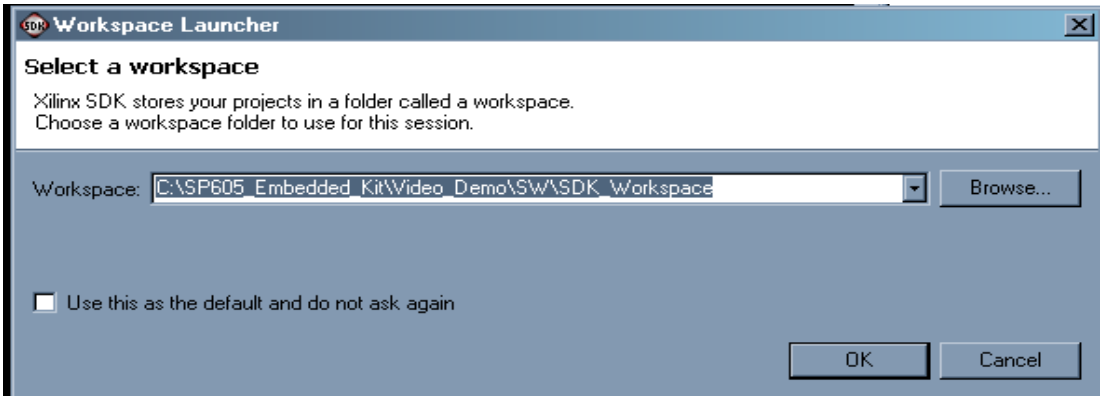

Figura. 4.16. Workspace Launcher.

Importación de la Plataforma de Hardware

A fin de importar al SDK la plataforma de hardware creada en el XPS, se debe:

- Seleccionar en el SDK, ***File → New → Xilinx Hardware Platform Specification***.

Se desplegará una ventana igual a la de la Figura. 4.17, en donde se podrá importar el archivo *system.xml* antes generado.

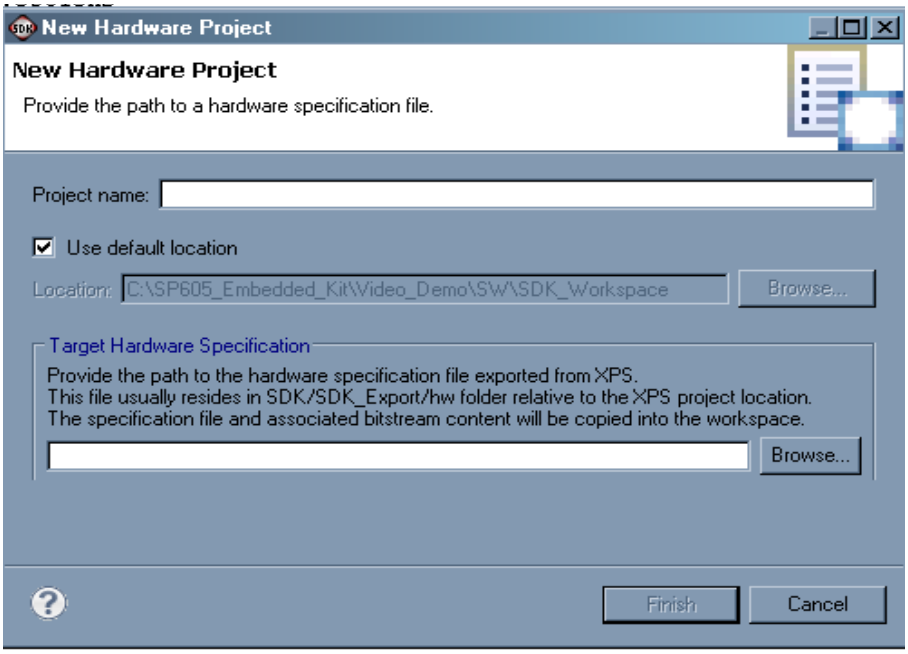

Figura. 4.17. Importación de la Plataforma de Hardware.

Seguido a esto, se debe llenar los siguientes campos:

- *Project Name:* ***nombre_plataforma_hw*** Este nombre especifica la plataforma de *hardware* que se va a utilizar durante todo el proyecto.
- *Project Location:* Seleccionar ***Use defaul location*** con el propósito de guardar la plataforma de *hardware* en el *workspace* creado.
- *Target Hardware Specification:* Dar click en ***Browse*** y buscar el archivo ***system.xml***, que contiene las especificaciones de la plataforma de *hardware*.

Creación y Configuración del *Board Support Package* (BSP)

Para crear un nuevo BSP, dirigirse a **File → New → *Xilinx Board Support Package***, y llenar los datos que requiere la ventana del BSP de la siguiente manera:

- Project name: ***nombre_bsp***
- Project location: **Asegúrese que el *checkbox Use default location* esté seleccionado**
- Hardware platform: ***nombre_plataforma_hw***
- CPU: **microblaze_0**
- Board Support Package OS: **Standalone o Xilkernel**

Se puede crear varios BSP, ya sea *standalone* o *xilkernel*, con el fin de poder crear varias aplicaciones de *software* sobre una misma plataforma de hardware. Cuando se genera el BSP, se crea el archivo *system.mss* (Figura. 4.18), el cual contiene la descripción de los *drivers* y librerías de los *IP Cores* añadidos en el XPS.

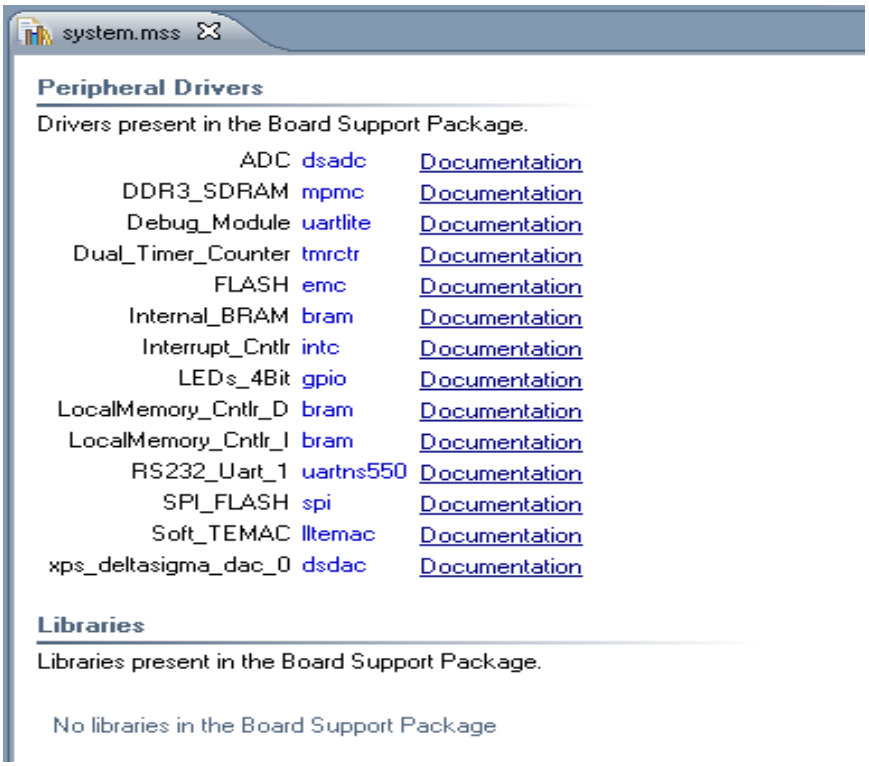

Figura. 4.18. Visualización del archivo system.mss

La configuración básica del BSP se muestra en la Figura. 4.19. Esta configuración consiste en escoger dispositivo ***stdin*** y ***stdout.*** Este dispositivo se encargará de la comunicación entre la tarjeta SP605 y la PC. Cabe recalcar que para este ejemplo se escogió como sistema operativo del Board Support Package el conjunto de librerías *standalone.*

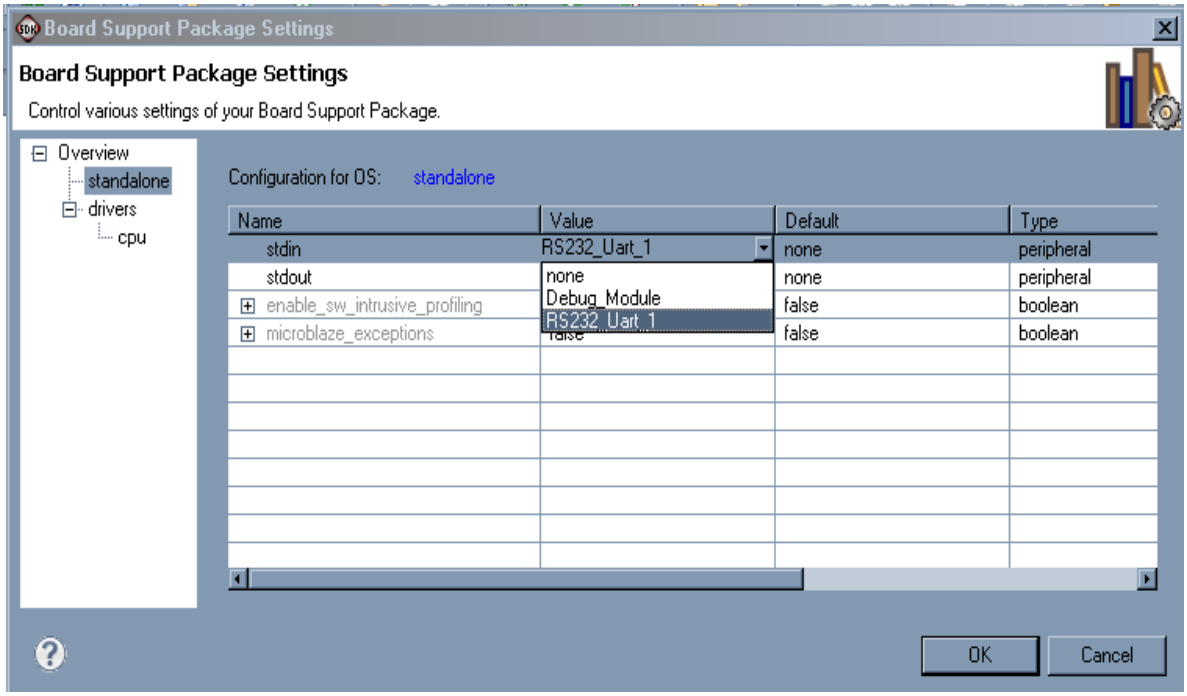

Figura. 4.19. Configuración Básica del BSP.

Creación o Importación de un Proyecto de Software

Para crear o importar un proyecto de Software se debe seleccionar:

- ***File → New → Xilinx C Project*** en el SDK.

Esta acción desplegará una ventana como la que se muestra en la Figura. 4.20. En la parte inferior izquierda de esta ventana, se escoge entre las diferentes opciones que nos brinda el SDK para la creación o importación de un proyecto[50].

De la lista que refleja esta ventana para la creación de un nuevo proyecto, se destaca ***Empty Application***, una aplicación en blanco para empezar desde cero un proyecto de

[50] **Nota:** Al igual que en la creación del BSP, tomar en cuenta el nombre de la plataforma de *hardware* con la que se está trabajando.

software. Además, es posible seleccionar plantillas para la realización de pruebas en la plataforma de hardware, las opciones son:

- ***Hello World:*** Imprime *Hello World* a través del dispositivo escogido como ***stdin***.
- ***Memory Test:*** Realiza pruebas de esritura y lectura en las memorias del sistema.
- ***Peripheral Test:*** Realiza pruebas inicializando y configurando los periféricos del sistema.
- ***Xilkernel POSIX Threds Demo:*** Ejecuta ejemplo básico del uso de hilos en Xilkernel.

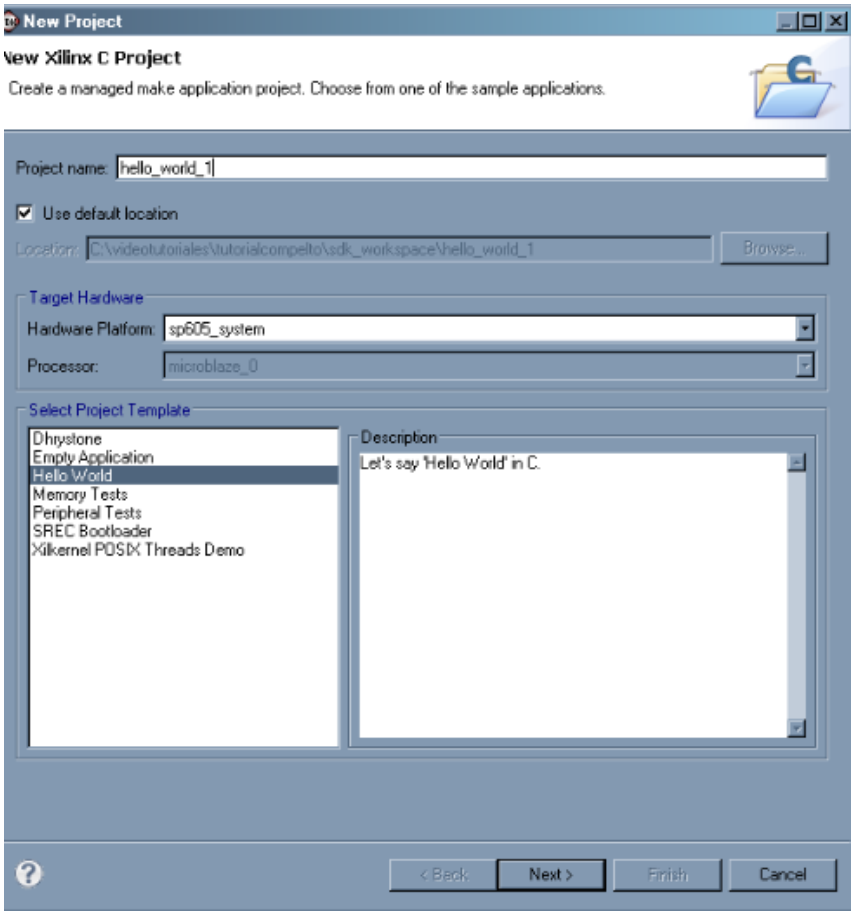

Figura. 4.20. Creación o Importación de un Proyecto de Software.

Para finalizar con la creación de un proyecto de software, seleccionar ***Target an existing Board Support Package***, para escoger un BSP creado anteriormente (Figura 4.21).

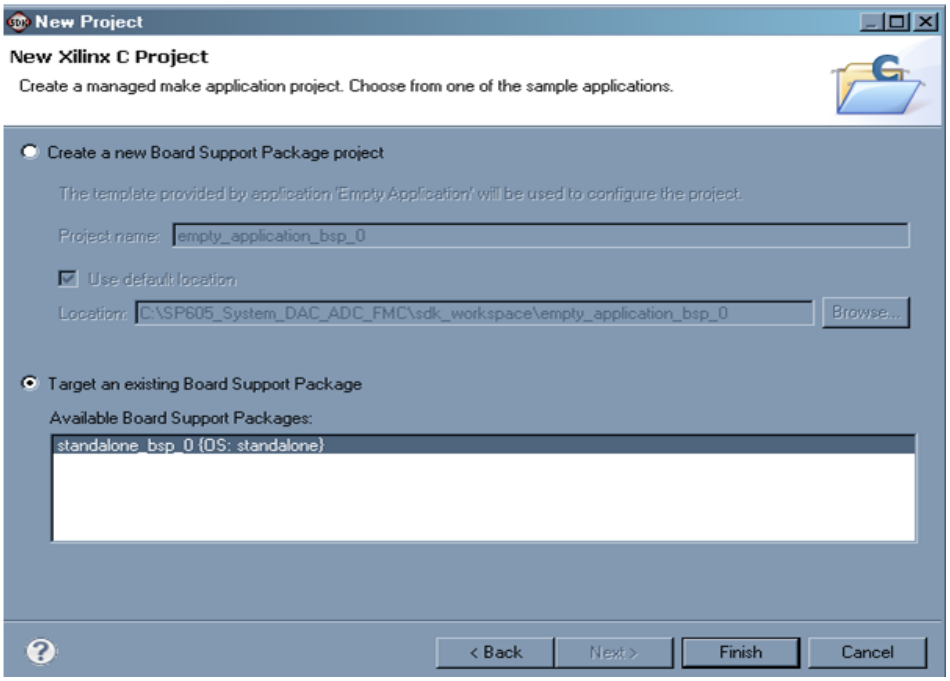

Figura. 4.21. Selección del BSP existente.

Para agregar un archivo de cabecera o de código al proyecto de *software*, *click* derecho sobre la carpeta ***src*** y seleccionar ***new*** (Figura. 4.22).

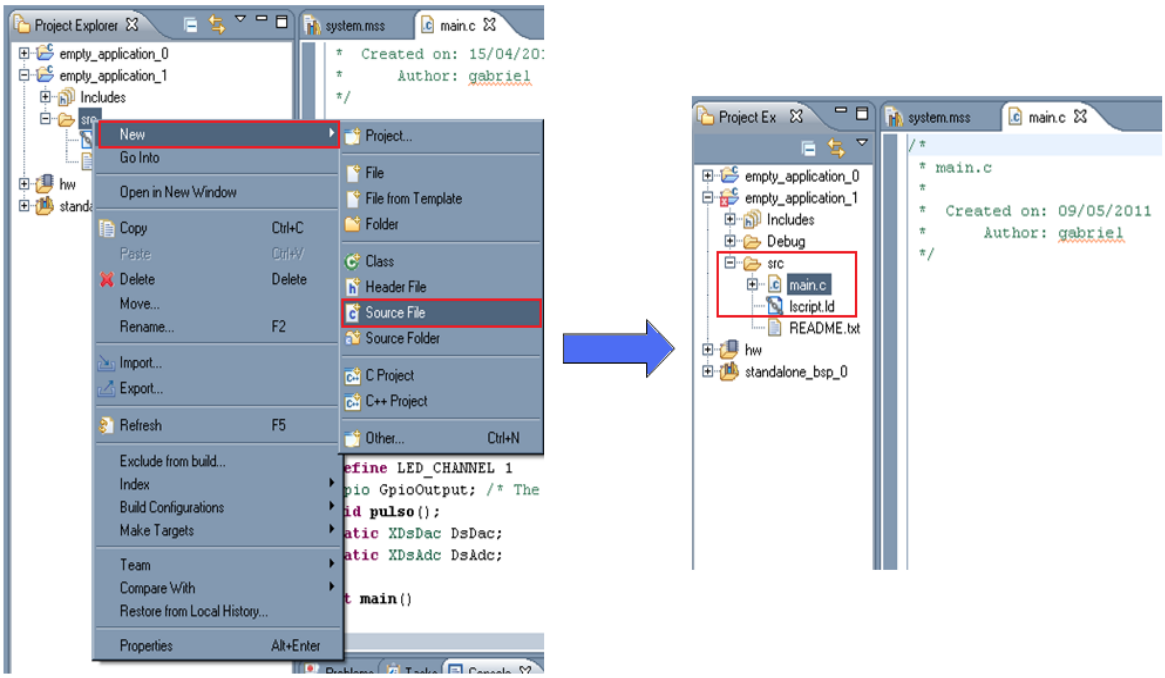

Figura. 4.22. Pasos para la Creación del Archivo *Main.c*.

Cabe mencionar que en la carpeta ***src*** se encuentra el *linker script* del proyecto de *software*. Este archivo es requerido para asignar las secciones de código, datos, pila y

almacenamiento dinámico a una memoria específica. Esto es muy importante para la correcta ejecución de una aplicación de *software* de tamaño mediano o grande.

Depuración de la Aplicación de Software

Una vez se haya desarrollado las sentencias del programa en C, para la aplicación de *software*, se procede a descargar el *bitstream* al FPGA para iniciar con la depuración. A fin de lograr esto, seleccione en el SDK, ***Xilinx Tools → Program FPGA***

Con esta acción se desplegará una ventana en la cual se escogerá la plataforma de *hardware* con la que se está trabajando. A continuación, como se indica en la Figura. 4.23, se debe verificar que los archivos ***system.bit*** y ***system.bmm***, correspondientes al *hardware*, estén seleccionados.

A continuación se da *click* sobre ***Program***, y se descargarán estos archivos al FPGA.

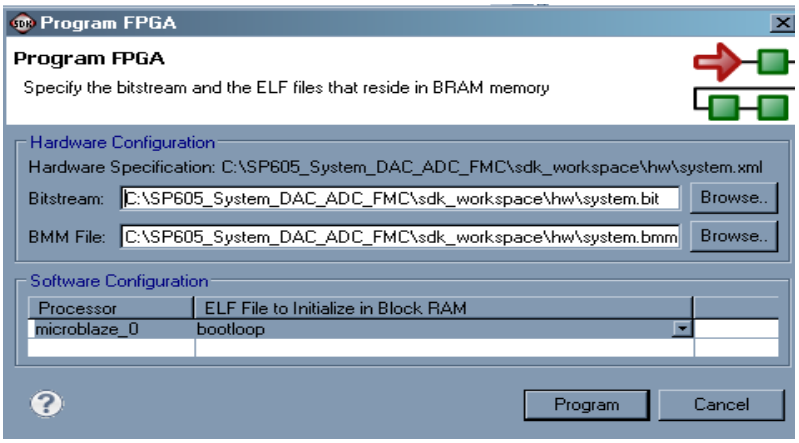

Figura. 4.23. Programación del FPGA con archivos .bit, .bmm, y .elf

Luego de que se haya programado el FPGA, se escoge uno de dos modos de configuración para la ejecución de una aplicación de *software*: ***Debug*** para la depuración, sin ninguna optimización o ***Release*** para la ejecución del programa con alta optimización.

Así como existe una herramienta para la depuración de *hardware*, existe una para la depuración de *software*. Para realizar este procedimiento se sigue los siguientes pasos:

- *Click* derecho sobre el proyecto de aplicación de *software*

- Seleccionar ***Debug as → Debug Configuration*** (Figura. 4.24)

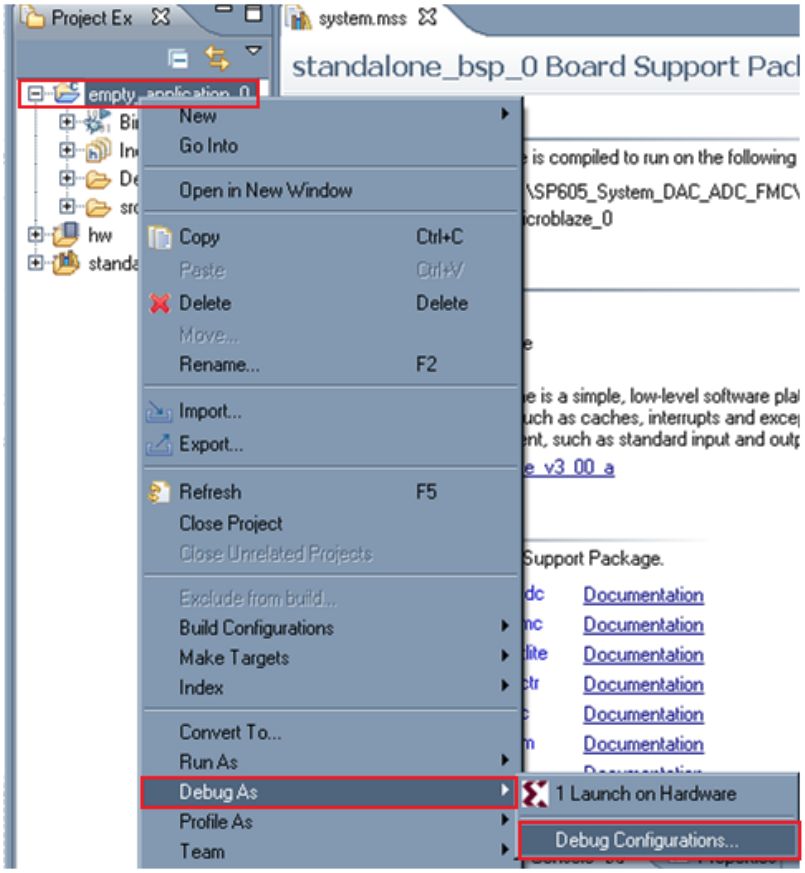

Figura. 4.24. Herramienta de Depuración de Software

Al seguir estos pasos se despliega una ventana como (Figura. 4.25), en donde se selecciona el botón ***New***, para crear una configuración del tipo seleccionado. En este caso se puede observar es de tipo ***Xilinx C/C++ ELF***.

Por último, dirigirse a la pestaña ***STDIO Connection***, habilitar con un *check* ***Connect STDIO to Console***, y configurar el puerto de comunicación serial, según se lo ha realizado con el cable usb-serial en la PC, luego pulsar *Debug*.

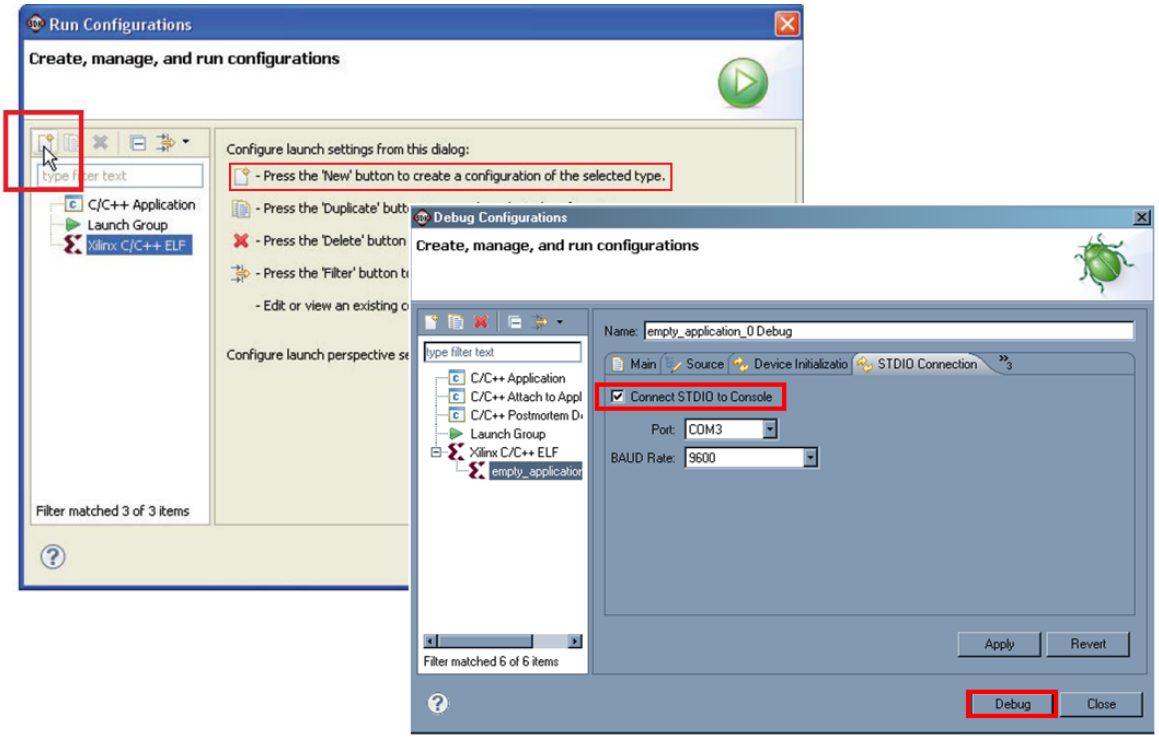

Figura. 4.25. Configuración de la Depuración de Software

En la ventana de consola del SDK, realizar pruebas en base a la plantilla escogida, ya sea pruebas de los periféricos, memorias, o de la aplicación de C que se haya realizado.

4.2 GUIAS DE ESTUDIO

4.2.1 GUIA 1: Co-Diseño de *Hardware/Software* Básico empleando SP605

INTRODUCCIÓN

Esta guía presentará el proceso de diseño de un sistema embebido simple gobernado por un procesador, utilizando *Xilinx Platform Studio* (XPS) e implemetandolo sobre la tarjeta de desarrollo SP605.

Después de finalizar esta guía usted será capaz de:

- Crear un proyecto en XPS, utilizando la herramienta *Base System Builder* (BSB)
- Crear un simple diseño de hardware, con *IP Cores* de propósito general.

- Exportar la plataforma de *hardware* creada en XPS a SDK, y ejecutar una plantilla de un programa de C.
- Crear un proyecto de *software* desde cero e implementarlo en el SDK *Workspace*
- Interactuar con la tarjeta SP605, co-diseñando el *hardware* y *software* desde cero.

PROCEDIMIENTO

En esta guía, se utilizará BSB del XPS, para crear un sistema procesado, el mismo que consistirá de los siguientes *IP Cores* (Figura. 4.26)

- Procesador *MicroBlaze* de 32 bits
- 2 Controladores BRAM LMB: 1 para datos y otro para instrucciones
- BRAM
- Módulo Debug MDM
- UART RS232 para la comunicación serial
- 3 GPIO de 4 bits para: *pushbutton*, *leds*, y *switches*
- Bloque de memoria RAM interna (BRAM) de 32 bits

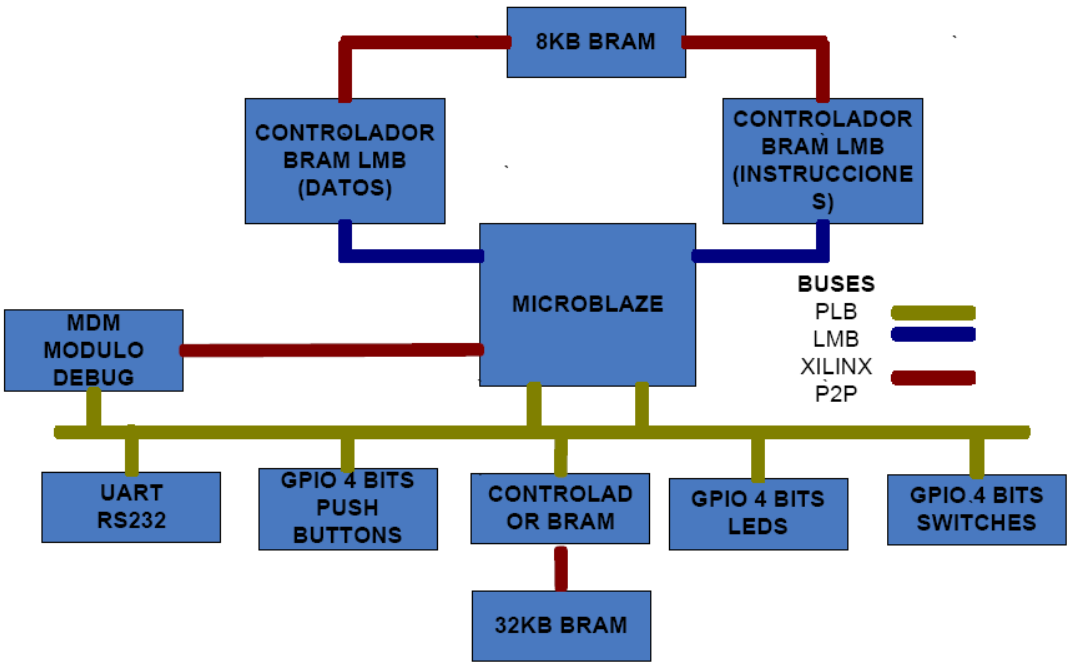

Figura. 4.26. Diseño Básico a Implementar en Guía 1

La aplicación de software consiste en la utilización de la plantilla ***Peripheral Test***. Cabe mencionar que esta aplicación es proporcionada por el SDK en base a los *IP Cores* que conforman la plataforma de hardware creada en XPS. Adicionalmente, se implementará una aplicación creada desde cero, que consistirá en la activación y desactivación de *leds* de la tarjeta SP605 a través de los *pushbuttons* y *switches* de la misma tarjeta.

Para crear el sistema antes expuesto se deben seguir los siguientes pasos:

a) Crear un proyecto utilizando *Base System Builder*
b) Analizar el proyecto creado en XPS
c) Exportar la Plataforma de *Hardware*
d) Crear un proyecto de software en SDK
e) Verificar el co-diseño en la tarjeta SP605

En el Anexo A10 se encuentra el procedimiento detallado para la realización de esta guía. Adicionalmente se puede visualizar el video GUIA1 que se encuentra en la memoria USB del SPARTAN 6 FPGA EMBEDDED KIT.

4.2.2 GUIA 2: Co-diseño de un SoC que contenga: DAC, ADC y MPMC.

INTRODUCCIÓN

Esta guía presentará el proceso de diseño de un sistema embebido, que realice las siguientes funciones:

- Adquisición de datos a través de un *XPS Delta-Sigma ADC*
- Reconstrucción de una señal analógica a través de un *XPS Delta-Sigma DAC*
- Manejo de la memoria DDR3 externa de la tarjeta SP605 a través de un MPMC.

Después de finalizar esta guía usted será capaz de:

- Crear un proyecto en XPS, utilizando la herramienta *Base System Builder* (BSB)
- Añadir *IP Cores* desde *el IP Catalog*.
- Crear un diseño de hardware con *IP Cores* del grupo analógico del *IP Catalog*.
- Exportar la plataforma de *hardware* creada en XPS a SDK, y ejecutar una plantilla de un programa de C.
- Crear un proyecto *software* desde cero e implementarlo en el SDK *Workspace*.
- Adquirir y reconstruir señales analógicas
- Añadir memoria cache de datos e instrucciones al diseño.

PROCEDIMIENTO

En esta guía, se utilizará el asistente *Base System Builder* (BSB) explicado en el capítulo 3, para crear un sistema procesado, el mismo que consistirá de los siguientes *IP Cores* (Figura. 4.27):

- Procesador MicroBlaze de 32 bits
- 2 Controladores BRAM LMB: 1 para datos y otro para instrucciones
- BRAM
- Módulo Debug MDM
- 1 Conversor Analogo – Digital XPS Delta – Sigma
- 1 Conversor Digital – Analogo XPS Delta – Sigma
- Multi-Port Memory Controller – MPMC.
- UART RS232 para comunicación serial
- 1 GPIO de 2 bits.
- 1 Utility Bus Split
- 2 Utility Flip Flop

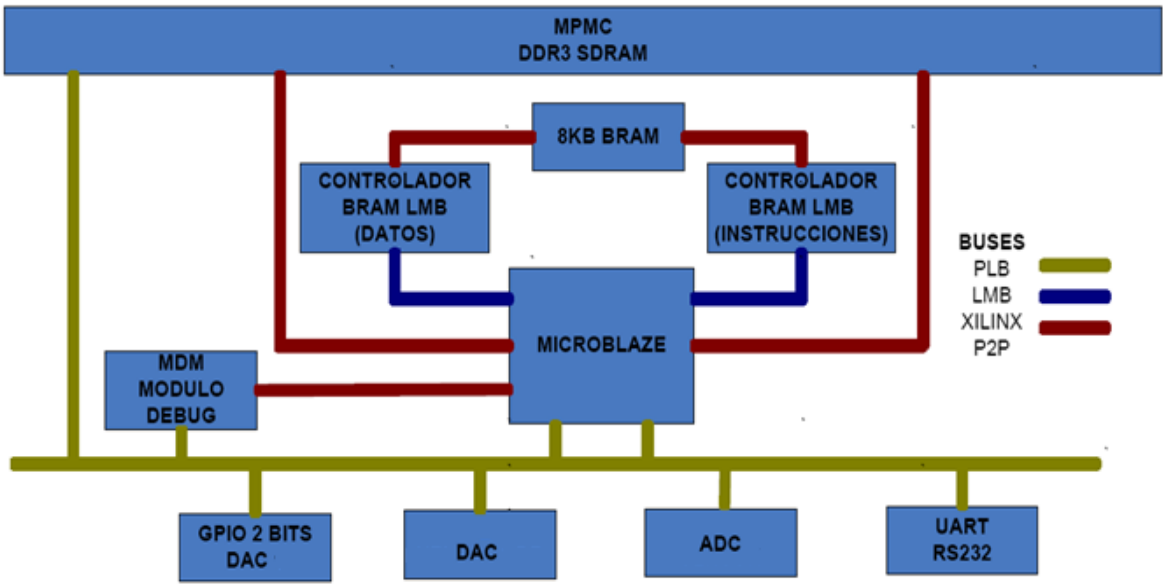

Figura. 4.27. Diseño a Implementar en Guía 2

La aplicación de *software* de esta guía consiste en la utilización de la plantilla ***memory test.*** Que consiste en un programa en C que permite la interacción con las memorias externas e internas disponibles en el diseño. Además, se creará una aplicación en C desde cero que proveerá la capacidad de adquirir una señal analógica en el ADC y reconstruirla por medio del DAC, empleando los *drivers* y librerías de cada *IP Core*.

Cabe recalcar que al emular el sistema se va a trabajar con la Tarjeta de Acondicionamiento de Entradas y Salidas (Anexo A9), la cual está provista con dos *jumpers*. Para implementar esta aplicación se debe asegurar que los *jumpers* en mención se encuentren en la posición 2. Esto habilita los siguientes circuitos:

- Filtro Pasa bajos para reconstruir la señal analógica. El punto P1 señalado en la tarjeta servirá para medir el voltaje de esta señal.
- Potenciómetro que simula una señal analógica a la entrada del ADC, cuyo voltaje será medido en el punto P2 señalado en la tarjeta.

Para crear el sistema antes expuesto se deben seguir los siguientes pasos:

a) Crear un proyecto de hardware utilizando *Base System Builder*
b) Añadir IP Cores desde el IP Catalog

c) Exportar la Plataforma de *Hardware*
d) Implementar el circuito externo para interactuar con la tarjeta SP605
e) Crear un proyecto de software en SDK
f) Verificar el co-diseño en la tarjeta SP605

En el Anexo A11 se encuentra el procedimiento detallado para la realización de esta guía. Adicionalmente se puede visualizar el video GUIA2 que se encuentra en la memoria USB del SPARTAN 6 FPGA EMBEDDED KIT.

4.2.3 GUIA 3: UG758 sobre la tarjeta SP605 - Xilkernel

INTRODUCCIÓN

Esta guía presentará el proceso de diseño de un SoC que trabaje con el Sistema Operativo en Tiempo Real Xilkernel. En este caso se tomará como plataforma de *hardware* base el proyecto *MicroBlaze Processor Subsystem*. Cabe mencionar que la aplicación de software consistirá en acoplar el diseño del documento de Xilinx UG758 *Using EDK to Run Xilkernel on a MicroBlaze Processor* a la tarjeta SP605.

Después de finalizar esta guía el usuario será capaz de:

- Crear un proyecto en XPS empleando la plataforma *MicroBlaze Processor Subsystem*.
- Eliminar IP Cores del Sistema.
- Añadir *IP Cores* desde el IP Catalog.
- Asignar pines en el archivo UCF.
- Exportar la plataforma de *hardware* creada en XPS a SDK, y ejecutar la plantilla ***Xilkernel POSIX Threds Demo***.
- Crear un proyecto *software* empleando Xilkernel

PROCEDIMIENTO

En esta guía, se empleará el *MicroBlaze Processor Subsystem* como plataforma base

para crear un sistema procesado en XPS, el mismo que consistirá de los siguientes *IP Cores* (Figura. 4.28):

- Procesador *MicroBlaze* de 32 bits
- 2 Controladores BRAM LMB: 1 para datos y otro para instrucciones
- BRAM
- Módulo Debug MDM
- *Multi-Port Memory Controller* - MPMC
- Controlador de Interrupciones
- UART RS232 para la comunicación serial
- Dual Timer Counter
- Timer Counter Clock

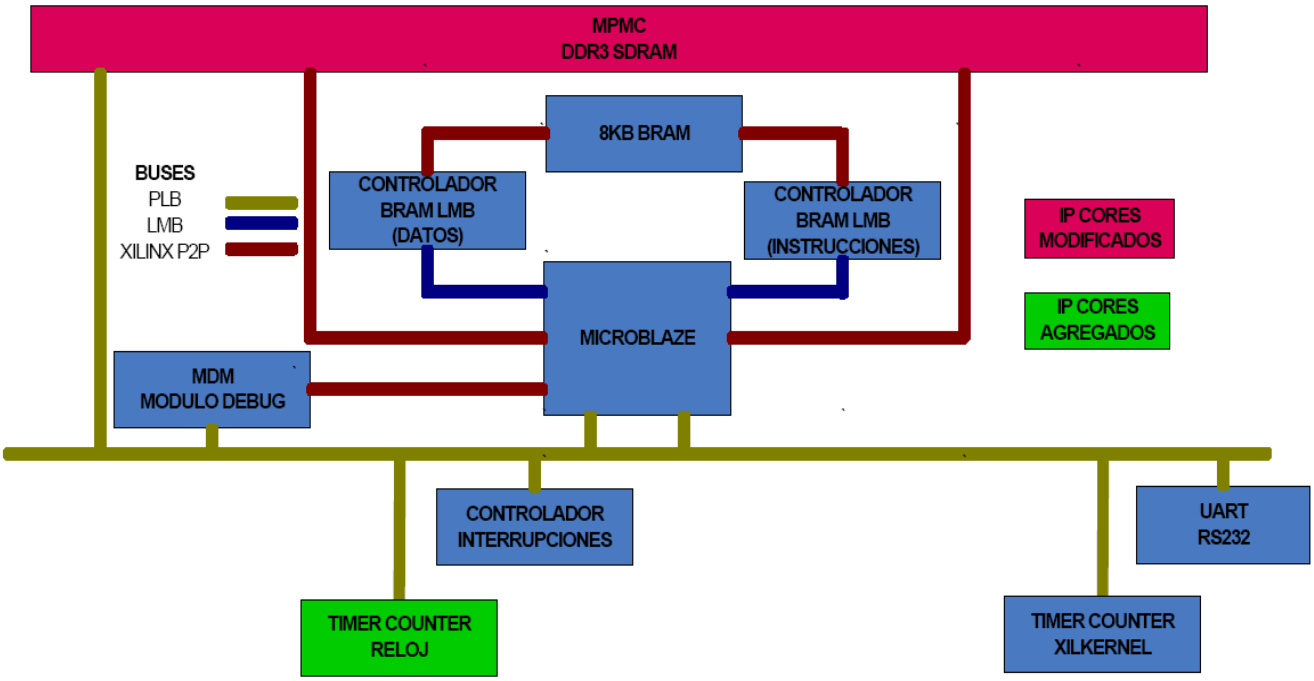

Figura. 4.28. Diseño a Implementar en Guía 3

La aplicación en C consiste en la utilización de la plantilla ***Xilkernel POSIX Threds Demo.*** Esta plantilla proporciona un ejemplo de la creación de múltiples hilos POSIX y su sincronización. El ejemplo crea un hilo inicial maestro, este hilo crea 4 hilos trabajadores que realizan el cálculo parcial de una suma y luego retorna la suma total como resultado. El hilo maestro acumula las sumas parciales e imprime el resultado.

Además, se acoplará el diseño del documento de Xilinx UG758 *Using EDK to Run Xilkernel on a MicroBlaze Processor* a la tarjeta SP605. Esta aplicación consiste de un hilo *shell_main*, el cual es capaz de enlistar siete hilos o subprogramas diferentes para interactuar con los módulos de Xilkernel mediante una Shell CLI[51]. Los archivos en lenguaje C de esta aplicación son:

- **clock.c:** Proporciona la funcionalidad de un reloj utilizando una interrupción de 1 Hz.
- **llist.c:** Ilustra
- **mutexdemo.c:** un ejemplo que enseña el uso de un API mutex para mostrar múltiples hilos
- **prodcon.c**: Ejemplo de productor consumidor, utilizando un bloque de una cola de mensajes. El productor se mantiene alimentando datos a la cola mientras el receptor intenta buscar el dato desde la cola. Ambos tanto productor como consumidor se bloquean cuando la cola se llena.
- **sem.c**: Ejemplo de uso de semáforos, con dos hilos utilizando semáforos para sincronizarse.
- **standby.c:** Es un hilo de alta prioridad que causa que el *kernel* vaya al estado de *standby* es decir que duerma por un tiempo determinado.
- **tictac.c:** es un programa que le permitirá jugar tres en raya

Para crear el sistema antes expuesto se deben seguir los siguientes pasos:

a) Crear un proyecto de hardware a partir de la plataforma *MicroBlaze Processor Subsystem*
b) Eliminar IP Cores del Sistema
c) Añadir IP Cores desde el IP Catalog
d) Asignación de pines en el archivo UCF
e) Exportar la Plataforma de *Hardware*
f) Crear un proyecto de software en SDK empleando la plantilla xilkernel
g) Crear un proyecto de software ejecutando xilkernel

[51] **Shell CLI**: código de programa que permite al usuario interactuar con el kernel. CLI (*Command Line Interface*): Shell que recibe instrucciones a través de comandos de texto.

En el Anexo A12 se encuentra el procedimiento detallado para la realización de esta guía. Adicionalmente se puede visualizar el video GUIA3 que se encuentra en la memoria USB del SPARTAN 6 FPGA EMBEDDED KIT.

CAPÍTULO 5

DISEÑO CONCEPTUAL DE LA APLICACIÓN

En este capítulo se detallará las especificaciones para el diseño de un sistema de control de temperatura. El sistema tendrá el esquema de proceso que se muestra en la Figura. 5.1, donde el controlador consiste en un SoC desarrollado en base a los conceptos de co-diseño de *hardware* y *software*, y la metodología PBD (diseño basado en plataforma).

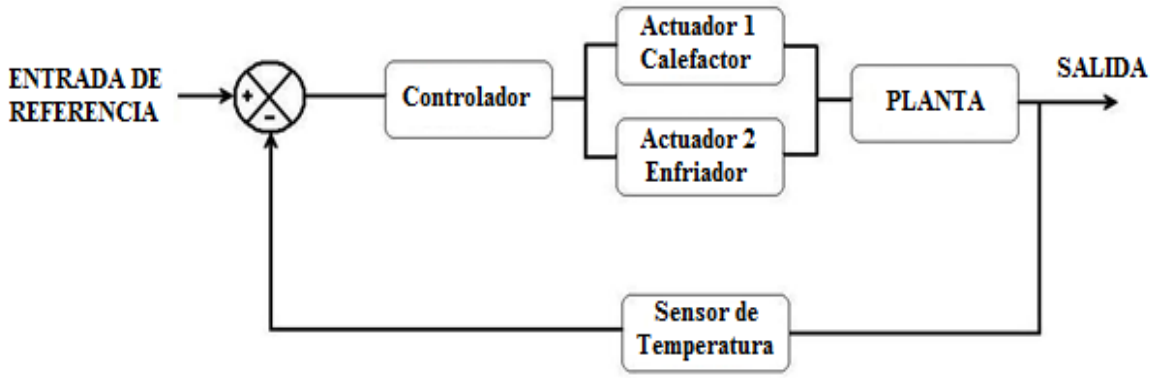

Figura. 5.1. Esquema de Proceso de Control para Planta de Temperatura

5.1 DISEÑO CONCEPTUAL DE HARDWARE Y SOFTWARE

El SoC realizará el control en lazo cerrado de la planta de temperatura que se detalla en el anexo A9. Se emplearan dos técnicas de control, *On-Off* y PID[52]. El proceso de diseño de este SoC llegará hasta la fase de emulación. A continuación se detalla las capas requeridas en el futuro SoC (Figura. 5.2).

[52] PID: Controlador Proporcional Integral Derivativo

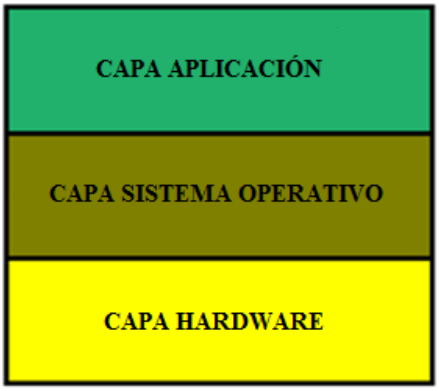

Figura. 5.2. Vista en Capas del Diseño

- **CAPA HARDWARE:** Diseño realizado en XPS de la plataforma de *hardware*.
- **CAPA SISTEMA OPERATIVO:** BSP creado en SDK.
- **CAPA APLICACIÓN:** Aplicación de *software* en lenguaje C desarrollada en SDK.

5.1.1 DISEÑO CONCEPTUAL DE HARDWARE

En la Figura. 5.3 se indica la disposición física de los elementos que conforman el proceso de control de temperatura.

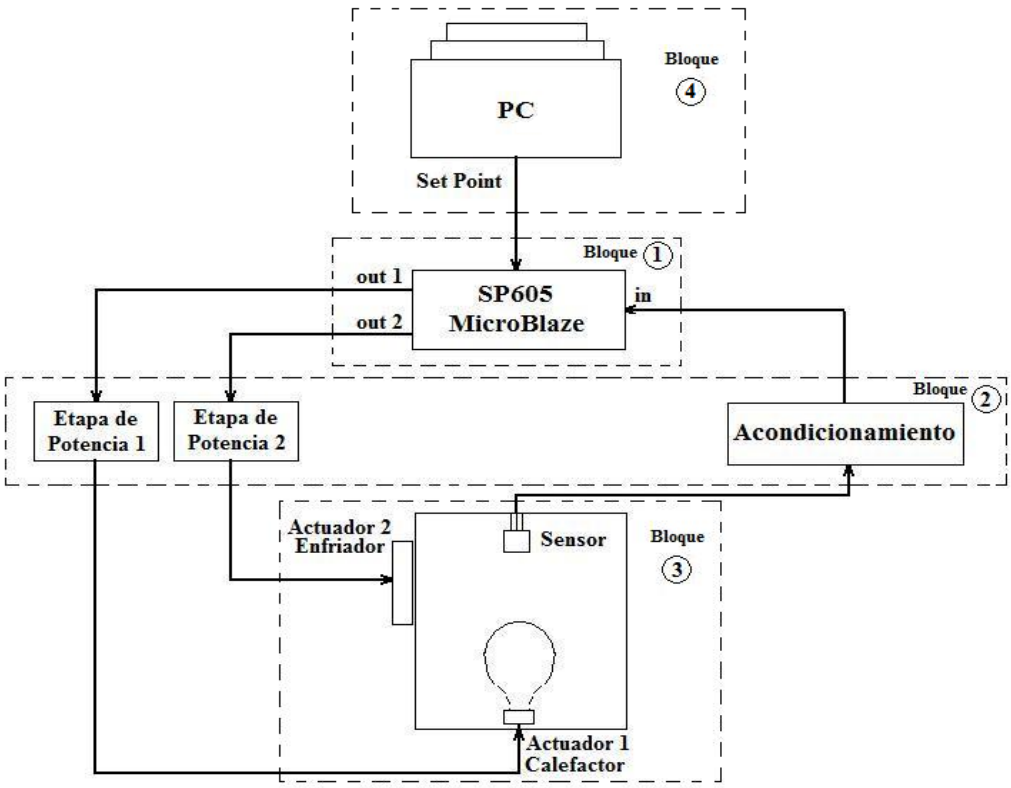

Figura. 5.3. Disposición Física de Elementos del Proceso de Control de Temperatura

Bloque 1

Este bloque corresponde a la plataforma de emulación SP605. EL FPGA Spartan 6 de esta plataforma será configurado con el *bitstream* de *hardware* generado en XPS y el código de *software* creado en SDK. El *hardware* implementará un microprocesador *Microblaze* con arquitectura *CoreConnect* para el procesamiento de la información y toma de decisiones, y una serie de *IP Cores* que cumplan con las siguientes funciones:

- Adquisición de datos del sensor de temperatura.
- Manejo de actuadores de calefacción (Foco) y de enfriamiento (Ventilador).
- Interfaz de usuario RS232.
- Controlador de memoria externa DDR3.
- Reloj del sistema.
- Soporte para RTOS.
- Generación de periodo de muestreo.
- Controlador de interrupciones.
- Depuración del sistema.

El diagrama de bloques de la plataforma de *hardware* que se espera obtener se detalla en la Figura. 5.4. Esta plataforma constituirá la capa de *hardware* del diseño y se desarrollará partiendo del proyecto *MicroBlaze Processor Subsystem*.

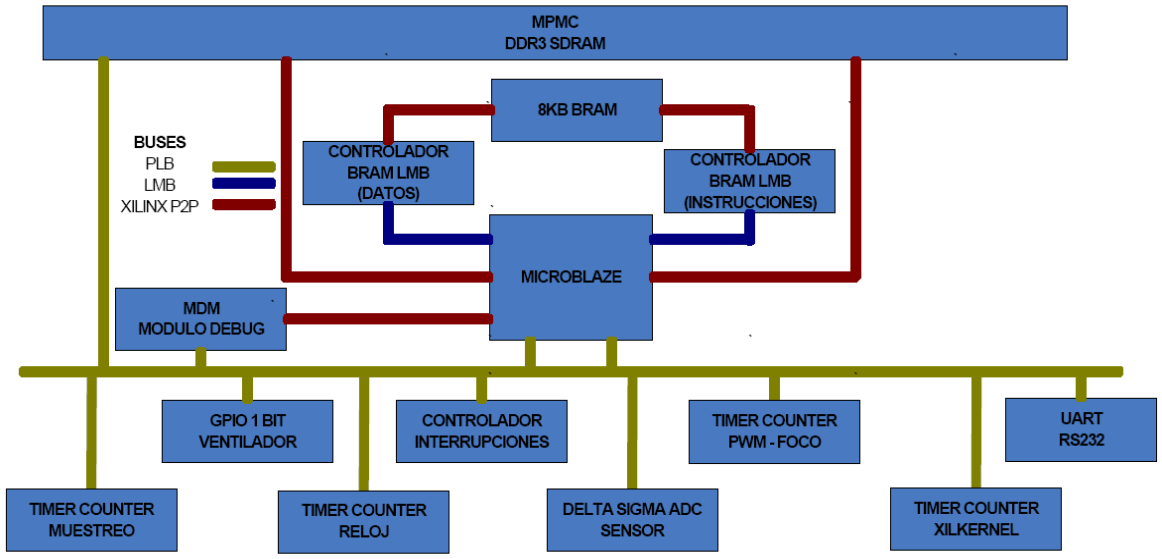

Figura. 5.4. Diagrama de Bloques del SoC

Para generar este *hardware* se seguirá el procedimiento de diseño propuesto en el Capítulo 4.

Bloque 2

Este bloque corresponde a la tarjeta de acondicionamiento de entradas y salidas, que contendrá los circuitos necesarios para trabajar con la planta de temperatura y la tarjeta SP605. Los circuitos externos que se necesitarán son:

- Dos etapas de potencia para los actuadores.
- Acondicionamiento para el sensor de la planta.
- *Hardware* externo necesario para trabajar con los *IP Cores* del SoC.

Bloque 3

Este bloque corresponde a la planta en sí, la misma que se encuentra detallada en el anexo A9. Esta planta tiene un rango de control de temperatura entre 40 y 65 grados centígrados, y su consumo es de 1.35 A (1.25A foco+ 0.1A ventilador).

Bloque 4

Este bloque contiene una terminal RS232 que sirva como interfaz de usuario, y permita el ingreso de comandos y la visualización de resultados. Se puede utilizar la consola del SDK o la *hiperterminal* de *Windows* para este propósito.

5.1.2 DISEÑO CONCEPTUAL DE SOFTWARE

Esta parte del diseño incluirá la implementación de las capas de sistema operativo y de aplicación, ambas realizadas en SDK. Para esto, se ejecutará el procedimiento de diseño propuesto en el Capítulo 4.

CAPA SISTEMA OPERATIVO

El BSP de esta capa contendrá los drivers y librerías para el manejo de las funciones

de hardware, y el RTOS *Xilkernel* para el trabajo con hilos, semáforos e interrupciones.

CAPA APLICACIÓN

La aplicación de *software* de esta capa realizará las siguientes funciones:

- Interfaz de línea de comandos (Shell CLI[53]).
- Control de temperatura ON-OFF.
- Control de temperatura PID.
- Gestión del reloj del sistema.
- Gestión de interrupciones.
- *Hardware Setup*.

Esta capa será programada en la herramienta SDK y utilizará las APIs del BSP creado en la capa de sistema operativo.

[53] **Shell CLI:** código de programa que permite al usuario interactuar con el *kernel*. *CLI (Command Line Interface):* Shell que recibe instrucciones a través de comandos de texto.

CAPÍTULO 6

IMPLEMENTACIÓN DEL CO-DISEÑO DE HARDWARE Y SOFTWARE

En el capítulo anterior se analizó la aplicación de Automatización y Control a diseñarse. Para desarrollar esta aplicación se partirá del proyecto *MicroBlaze Processor Subsystem*. Este es parte de los archivos del SPARTAN 6 FPGA EMBEDDED KIT, y sirve como plataforma base para generar el *hardware* del sistema con la herramienta XPS, en base a las especificaciones del diseño.

6.1 DISEÑO DE HARDWARE

Para generar la plataforma de *hardware* se eliminarán, modificarán, y añadirán *IP Cores*, al proyecto *MicroBlaze Processor Subsystem*, con el fin de cumplir las especificaciones del diseño conceptual.

6.1.1 PERSONALIZACIÓN DEL MICROBLAZE PROCESSOR SUBSYSTEM

A continuación se analizará la plataforma de *hardware* implementada en XPS y las configuraciones de cada uno de los *IP Cores* añadidos. Para esto, se presenta el diagrama de bloques del *MicroBlaze Processor Subsystem* (Figura. 6.1), plataforma base para el desarrollo de este SoC.

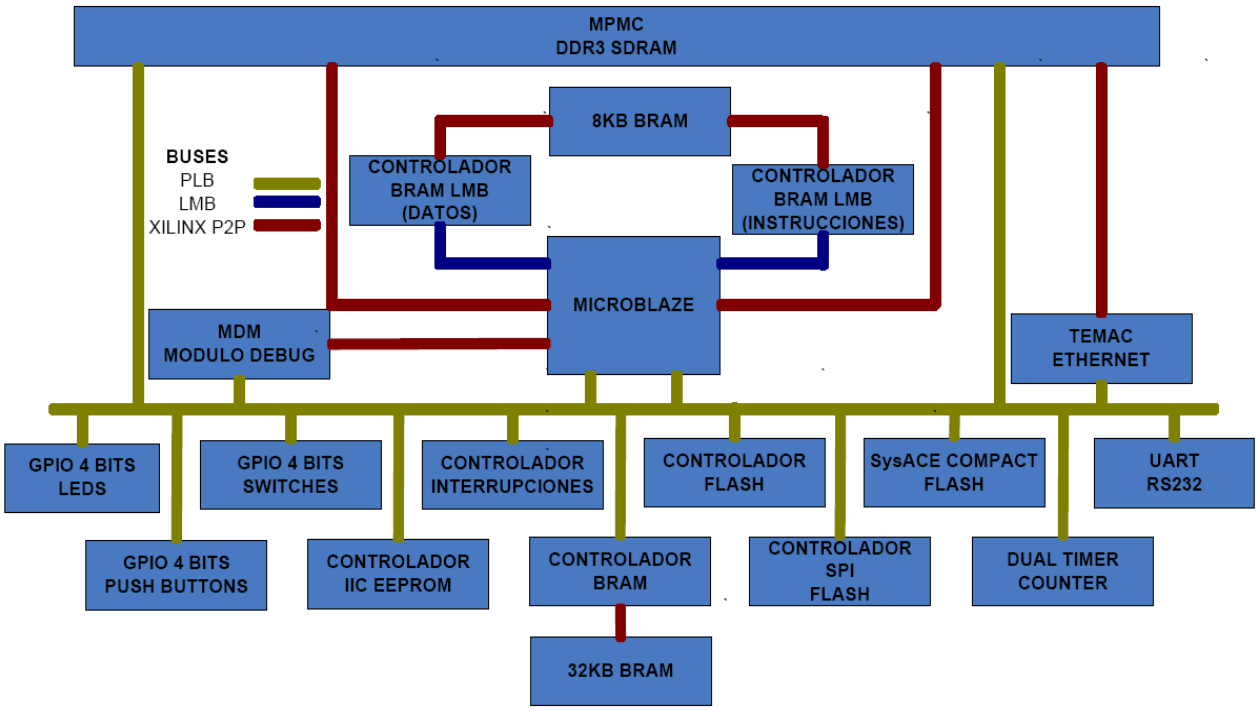

Figura. 6.1. Diagrama de Bloques de *MicroBlaze Processor Subsystem*

La Figura. 6.2, indica los *IP Cores* que van a ser eliminados de la plataforma *MicroBlaze Processor Subsystem*, ya que no son necesarios para cumplir las funciones del diseño.

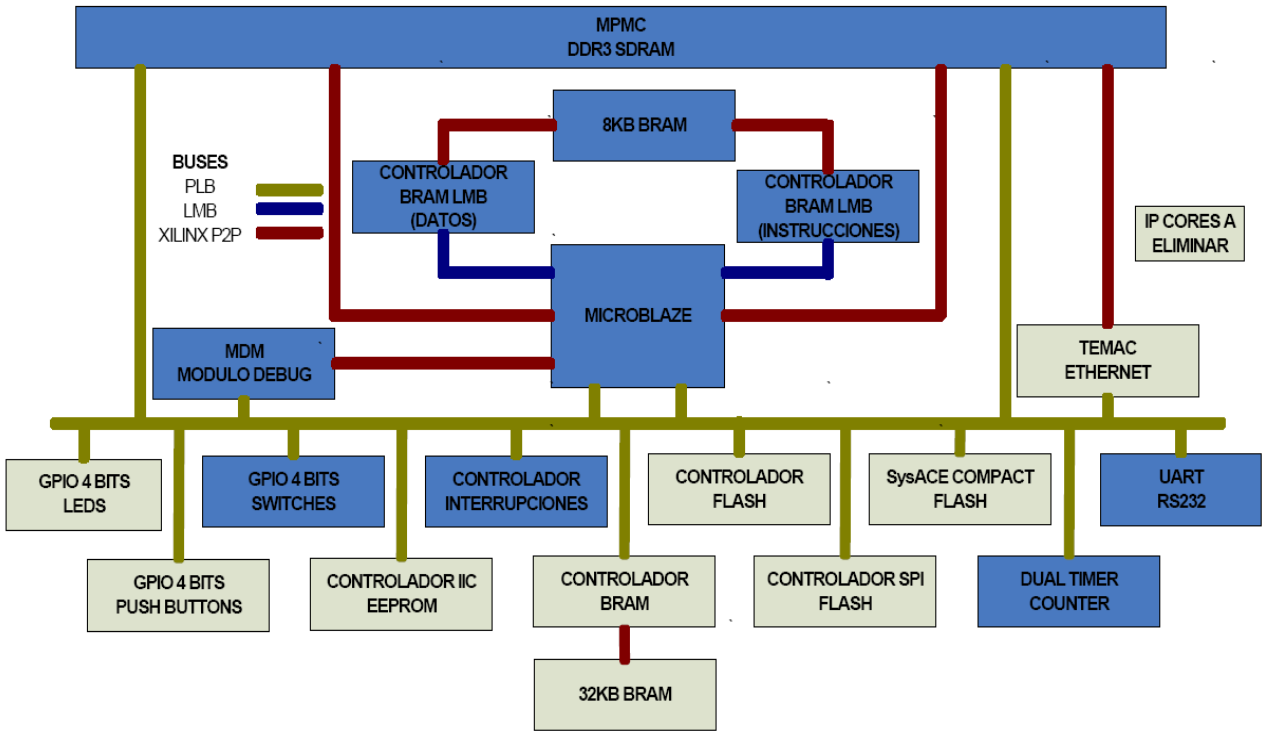

Figura. 6.2. Diagrama de Bloques de *MicroBlaze Processor Subsystem* con *IP Cores* a ser eliminados

El resultado de eliminar los *IP Cores* innecesarios se muestra en la siguiente figura.

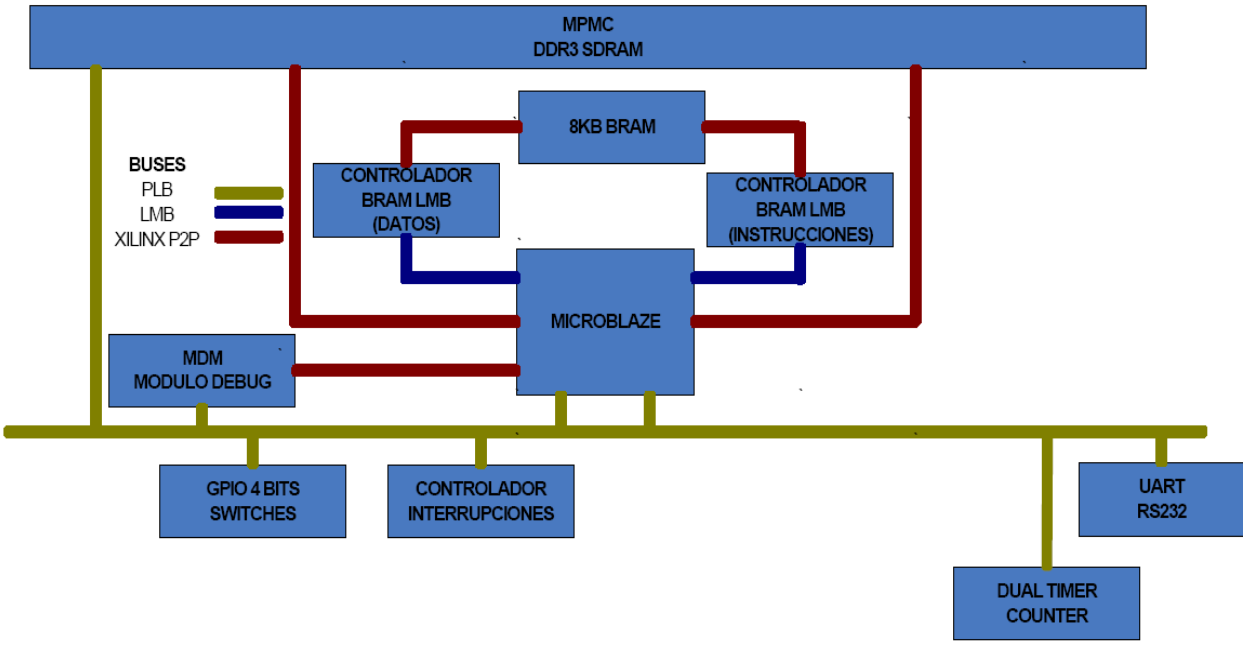

Figura. 6.3. Diagrama de Bloques de *MicroBlaze Processor Subsystem* – *IP Cores* Eliminados

En este caso, el *MicroBlaze Processor Subsystem* no contiene todos los *IP Cores* necesarios para cumplir con las especificaciones del diseño, así que se añadirán los siguientes módulos del *IP Catalog*:

- XPS Delta – Sigma ADC para adquisición de datos del sensor de temperatura.
- XPS Timer/Counter para manejo del actuador de calefacción (Foco) a través de PWM.
- XPS Timer/Counter para reloj del sistema.
- XPS Timer/Counter para generar periodo de muestreo.

Por otro lado, se modificará la configuración del GPIO destinado a los *Switches* de la tarjeta SP605. Este *IP Core* manejará el elemento de enfriamiento (ventilador). Además, se debe eliminar el puerto destinado a SDMA del controlador MPMC de la memoria DDR3 externa, ya que este puerto estaba conectado al módulo TEMAC, que fue eliminado del *MicroBlaze Processor Subsystem* anteriormente.

El diagrama de bloques de la plataforma de *hardware* resultante de la personalización del *MicroBlaze Processor Subsystem* se muestra en la Figura. 6.4.

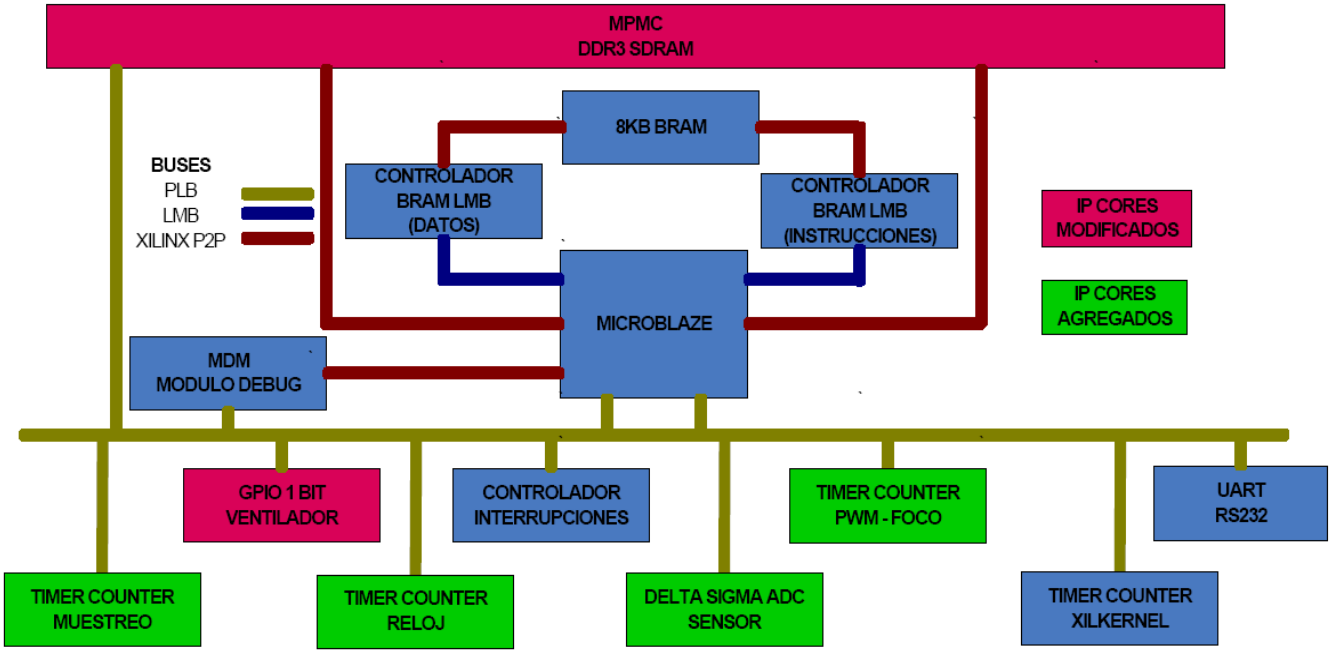

Figura. 6.4. Diagrama de Bloques de *MicroBlaze Processor Subsystem* – *IP Cores* Editados y Agregados

CONFIGURACIÓN DE LOS IP CORES

A continuación se muestra un resumen de las configuraciones finales de los *IP Cores* de la plataforma de *hardware* resultante.

a) XPS General Purpose Input/Output (GPIO)

Este *IP Core XPS General Purpose IO*, maneja el elemento de enfriamiento (ventilador). A continuación se muestra su configuración.

Nombre del IP Core: gpio_ventilador

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	GPIO_IO_O	O	1	gpio_ventilador_GPIO_IO_O

Tabla. 6.1. Puertos de gpio_ventilador

Nombre	Valor
C_BASEADDR	0x81400000
C_HIGHADDR	0x8140ffff
C_GPIO_WIDTH	1
C_INTERRUPT_PRESENT	0
C_TRI_DEFAULT	0xffffffff
C_ALL_INPUTS	0
C_IS_DUAL	0

Tabla. 6.2. Parámetros de Configuración del gpio_ventilador

b) XPS 16550 UART

Este *IP Core* es usado para la comunicación serial con la terminal de usuario RS232. A continuación se detalla su configuración.

Nombre del IP Core: RS232_Uart_1

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	Sin	I	1	fpga_0_RS232_Uart_1_sin
1	Sout	O	1	fpga_0_RS232_Uart_1_sout
2	IP2INTC_Irpt	O	1	RS232_Uart_1_IP2INTC_Irpt

Tabla. 6.3. Puertos de RS232_Uart_1

Nombre	Valor
C_BASEADDR	0x83e0000
C_HIGHADDR	0x83e0FFFF
C_IS_A_16550	1
C_HAS_EXTERNAL_XIN	0
C_HAS_EXTERNAL_RCLK	0
C_EXTERNAL_XIN_CLK_HZ	25000000
C_SPLB_AWIDTH	32

Tabla. 6.4. Parámetros de Configuración del RS232_Uart_1

c) XPS Interrupt Controller.

El *IP Core XPS Interrupt Controller* gestiona todas las interrupciones provenientes del resto de IP Cores en el sistema. A continuación se detallan su configuración.

Nombre del IP Core: Interrupt_Cntlr

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	Irq	O	1	Interrupt
1	Intr	I	1	timer_xilkernel_Interrupt&timer_sample_Interrupt&timer_clock_Interrupt&stop_interrupt&ADC_IP2INTC_Irpt&PWM_Interrupt&RS232_Uart_1_IP2INTC_Irpt

Tabla. 6.5. Puertos de Interrupt_Cntlr

Prioridad	Señal	Módulo
0	timer_xilkernel_Interrupt	timer_xilkernel
1	timer_sample_Interrupt	timer_sample
2	timer_clock_Interrupt	timer_clock
3	stop_Interrupt	
4	ADC_IP2INTC_Irpt	ADC
5	PWM_interrupt	PWM
6	RS232_Uart_1_IP2INTC_Irpt	RS232_Uart_1

Tabla. 6.6. Prioridad de Interrupciones

d) XPS Delta-Sigma Analog to Digital Converter

El *IP Core XPS Delta-Sigma Analog to Digital Converter*, se utiliza para la adquisición de datos del sensor de temperatura. Tiene la ventaja de usar únicamente dos pines del FPGA y un circuito externo que consta de un filtro pasa bajos y un comparador. Esto hace de este *IP Core* una buena opción para disminuir el tamaño de un sistema embebido. A continuación se indica su configuración.

Nombre del IP Core: ADC

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	AgtR	I	1	ADC_AgtR
1	DACout	O	1	ADC_DACout
2	IP2INTC_Irpt	O	1	ADC_IP2INTC_Irpt

Tabla. 6.7. Puertos de ADC

Nombre	Valor
C_DACIN_WIDTH	9
C_FSTM_WIDTH	1
C_BASEADDR	0x8040000
C_HIGHADDR	0x8040FFFF
C_SPLB_AWITH	32
C_SPLB_DWITH	32

Tabla. 6.8. Parámetros de Configuración del ADC

e) XPS Timer/Counter

En este sistema se han añadido 4 *XPS Timer/Counter*, para realizar las funciones de reloj del sistema, generación de periodo de muestreo, soporte para el RTOS Xilkernel y generación de la señal de control PWM para el control del elemento de calefacción. A continuación se detallan las configuraciones de cada módulo.

Nombre del IP Core: timer_clock

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	Interrupt	O	1	timer_clock_Interrupt

Tabla. 6.9. Puertos de timer_clock

Nombre	Valor
C_BASEADDR	0x83c40000

Nombre	Valor
C_HIGHADDR	0x83c4ffff
C_COUNT_WIDTH	32
C_ONE_TIMER_ONLY	0

Tabla. 6.10. Parámetros de timer_clock

Nombre del IP Core: timer_sample

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	Interrupt	O	1	timer_sample_Interrupt

Tabla. 6.11. Puertos de timer_sample

Nombre	Valor
C_BASEADDR	0x83c20000
C_HIGHADDR	0x83c2ffff
C_COUNT_WIDTH	32
C_ONE_TIMER_ONLY	0

Tabla. 6.12. Parámetros de timer_sample

Nombre del IP Core: timer_xilkernel

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	Interrupt	O	1	timer_xilkernel_Interrupt

Tabla. 6.13. Puertos de timer_xilkernel

Nombre	Valor
C_BASEADDR	0x83c00000
C_HIGHADDR	0x83c0ffff
C_COUNT_WIDTH	32
C_ONE_TIMER_ONLY	0

Tabla. 6.14. Parámetros de timer_xilkernel

Nombre del IP Core: PWM

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	PWM0	O	1	PWM_PWM0

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
1	Interrupt	O	1	PWM_Interrupt

Tabla. 6.15. Puertos de PWM

Nombre	Valor
C_BASEADDR	0x83c60000
C_HIGHADDR	0x83c6ffff
C_COUNT_WIDTH	32
C_ONE_TIMER_ONLY	0

Tabla. 6.16. Parámetros de PWM

f) MicroBlaze Soft Processor

El *IP Core MicroBlaze Soft Processor* se encarga del procesamiento de información y de la toma de decisiones en el sistema. Además, contiene el mapa de memoria para los periféricos esclavos conectados al bus PLB (Tabla. 6.19). Cabe recalcar que en la configuración de este *IP Core*, es necesario deshabilitar el parámetro *Use Memory Managment* (en caso de que esté habilitado), ya que el RTOS Xilkernel no soporta memoria virtual. A continuación se muestra la configuración del *MicroBlaze Soft Processor*.

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	MB_RESET	I	1	mb_reset
1	Interrupt	I	1	Interrupt

Tabla. 6.17. Puertos de MicroBlaze Soft Processor Core

Nombre	Tipo	BUSSTD	BUS	Conectado a
DXCL	INITIATOR	XIL_MEMORY_CHANNEL	microblaze_0_DXCL	DDR3_SDRAM
IXCL	INITIATOR	XIL_MEMORY_CHANNEL	microblaze_0_IXCL	DDR3_SDRAM
DPLB	MASTER	PLBV46	mb_plb	10 Periféricos
IPLB	MASTER	PLBV46	mb_plb	10 Periféricos
DLMB	MASTER	LMB	dlmb	LocalMemory_Cntrl_D
ILMB	MASTER	LMB	ilmb	LocalMemory_Cntrl_I
DEBUG	TARGET	XIL_MBDEBUG2	microblaze_0_dbg	Debug_Module

Tabla. 6.18. Buses de MicroBlaze Soft Processor Core

Módulo	Nombre Base	Base	Superior	Tamaño
Local_Memory_Cntlr_D	C_BASEADDR	0x00000000	0x00001FFF	8K
Local_Memory_Cntlr_I	C_BASEADDR	0x00000000	0x00001FFF	8K
ADC	C_BASEADDR	0x80400000	0x8040FFFF	64K
gpio_ventilador	C_BASEADDR	0x81400000	0x8140FFFF	64K
Interrupt_Cntlr	C_BASEADDR	0x81800000	0x8180FFFF	64K
timer_xilkernel	C_BASEADDR	0x83C00000	0x83C0FFFF	64K
timer_sample	C_BASEADDR	0x83C20000	0x83C2FFFF	64K
timer_clock	C_BASEADDR	0x83C40000	0x83C4FFFF	64K
PWM	C_BASEADDR	0x83C60000	0x83C6FFFF	64K
RS232_Uart_1	C_BASEADDR	0x83E00000	0x83E0FFFF	64K
Debug_Module	C_BASEADDR	0x84400000	0x8440FFFF	64K
DDR3_SDRAM	C_MPMC_BASEADDR	0x88000000	0x8FFFFFFF	128M

Tabla. 6.19. Asignación de Memoria

g) Processor Local Bus (PLB)

Este bus está destinado a conectar el procesador MicroBlaze a los periféricos del sistema A continuación se indica su configuración.

Instancia	Tipo de Interface	Nombre de Interface
microblaze_0	MASTER	DPLB
microblaze_0	MASTER	IPLB
Debug_Module	SLAVE	SPLB
RS232_Uart_1	SLAVE	SPLB
DDR3_SDRAM	SLAVE	SPLB 1
Interrupt_Cntlr	SLAVE	SPLB
timer_xilkernel	SLAVE	SPLB
timer_ clock	SLAVE	SPLB
timer_ simple	SLAVE	SPLB
PWM	SLAVE	SPLB
ADC	SLAVE	SPLB
gpio_ventilador	SLAVE	SPLB

Tabla. 6.20. Conexión de Maestros y Esclavos al bus PLB

#	Nombre	Dirección	[LSB:MSB]	SEÑAL
0	PLB_Clk	I	1	sys_clk_s
1	SYS_Rst	I	1	sys_bus_reset

Tabla. 6.21. Puertos del bus PLB

h) Local Memory Bus (LMB)

Existen dos buses LMB que conectan el procesador *Microblaze* con los controladores de memoria para datos e instrucciones. A continuación se indica su configuración.

#	Nombre	Dir	[LSB:MSB]	Señal
0	LMB_Clk	I	1	sys_clk_s
1	SYS_Rst	I	1	sys_bus_reset

Tabla. 6.22. Puertos de buses LMB

Instancia	Tipo de interface	Nombre de Interface
microblaze_0	MASTER	ILMB
LocalMemory_Cntrl_I	SLAVE	SLMB

Tabla. 6.23. Conexiones de bus LMB para instrucciones

Instancia	Tipo de interface	Nombre de Interface
microblaze_0	MASTER	DLMB
LocalMemory_Cntrl_D	SLAVE	SLMB

Tabla. 6.24. Conexiones de bus LMB para datos

6.1.2 ASIGNACIÓN DE PINES DEL FPGA SPARTAN 6 EN EL ARCHIVO UCF

Una vez conocidas las conexiones de todos los puertos de cada *IP Core*, se procede a realizar la asignación de pines del FPGA asociados a estos puertos. A continuación se muestra el archivo *system.ucf* resultante de este proceso.

```
#
# Target Board: Xilinx Spartan-6 SP605 Evaluation Platform Rev C

NET  sys_clk_in_p            LOC = K21  | IOSTANDARD = LVDS_25 |
DIFF_TERM = TRUE;
NET  sys_clk_in_n            LOC = K22  | IOSTANDARD = LVDS_25 |
DIFF_TERM = TRUE;

Net sys_rst_pin LOC= H8 | IOSTANDARD = LVCMOS15;
## System level constraints
Net sys_rst_pin TIG;

Net dcm_clk_s TNM_NET = dcm_clk_s;
TIMESPEC TS_dcm_clk_s = PERIOD dcm_clk_s 5000 ps;

## IO Devices constraints

#### Module RS232_Uart_1 constraints

Net fpga_0_RS232_Uart_1_sin_pin LOC=H17  | IOSTANDARD = LVCMOS25;
Net fpga_0_RS232_Uart_1_sout_pin LOC=B21 | IOSTANDARD = LVCMOS25;

#### MCB parameters
NET MPMC_0_mcbx_dram_zio LOC = M7 | IOSTANDARD = SSTL15_II;

#### GPIO ventilador

NET gpio_ventilador_GPIO_IO_O_pin<0> LOC= H13 | IOSTANDARD = LVTTL |
DRIVE = 24;##G15 FMC

#### Timer PWM

NET PWM_PWM0_pin LOC= G13 | IOSTANDARD = LVTTL | DRIVE = 24;##G16 FMC

#### ADC

NET ADC_AgtR_pin LOC = C5 | IOSTANDARD = LVCMOS25; ##G18 FMC
NET ADC_DACout_pin LOC= A5 | IOSTANDARD = LVTTL | DRIVE = 24; ##G19 FMC

#### BOTON STOP

NET stop LOC = F3 |  IOSTANDARD=LVCMOS15  |  PULLDOWN  |  SLEW=SLOW  |
DRIVE=2;
NET "stop" CLOCK_DEDICATED_ROUTE = FALSE;
```

6.1.3 GENERACIÓN Y EXPORTACIÓN DEL BITSTREAM DE LA PLATAFORMA DE HARDWARE

Para concluir con el diseño de *hardware* embebido se genera y exporta al SDK el archivo que contiene el *bitstream* de la plataforma de *hardware*, tal como se analizó en el capítulo 4.

6.2 COMPONENTES EXTERNOS DE HARDWARE

El SoC de control de temperatura, requiere de circuitos externos para acondicionar las señales de la planta al *hardware* interno. Todos estos circuitos están incluidos en la tarjeta de acondicionamiento de entradas y salidas que se detalla en el anexo 10. Los elementos que conforman esta tarjeta son:

- Dos etapas de potencia basadas en un Optoacoplador y un Triac.
- Etapa de acondicionamiento basada en amplificador operacional.
- Comparador y filtro pasa bajos para el *IP Core XPS Delta-Sigma ADC*.

Cabe mencionar que esta tarjeta también contiene circuitos para el desarrollo de la GUIA 2, estos son:

- Un filtro pasa bajos para la reconstrucción de la señal del *IP Core XPS Delta-Sigma DAC*
- Un potenciómetro que simula una señal analógica para *IP Core XPS Delta-Sigma ADC*.

Cuando se desea ejecutar la aplicación de control, se debe configurar ambos *jumpers* de la tarjeta de acondicionamiento de entradas y salidas en la posición 1, en tanto que para ejecutar la GUIA 2 deben estar en la posición 2 (ver Anexo A9)

6.2.1 ETAPAS DE POTENCIA

Las etapas de potencia 1 y 2, indicadas en la Figura. 5.3, están constituidas por los elementos de la Tabla. 6.25, que se detalla a continuación:

Cantidad	Elemento	Modelo / Valor	Descripción
2	Optoacoplador	MOC 3031	Dispositivo de aislamiento óptico para el FPGA
2	Resistencias	680 ohm	Elemento pasivo para interrumpir el paso de la corriente
2	Resistencias	1Kohm / 1W	Elemento pasivo para interrumpir el paso de la corriente
2	Capacitores	100nF / 200V	Elemento pasivo para almacenamiento de energía
2	TRIAC	BTA16 600B	Dispositivo semiconductor capaz de conmutar corriente AC

Tabla. 6.25. Elementos de la Etapa de Potencia

Las hojas técnicas tanto del Optoacoplador MOC 3031 y del TRIAC BTA16 600B, se encuentran en los anexos A4, y A5, respectivamente.

Los elementos antes citados han sido dispuestos en la tarjeta de acondicionamiento de entradas y salidas, como se muestra en la Figura. 6.5, la misma que indica los circuitos empleados para ensamblar estas etapas.

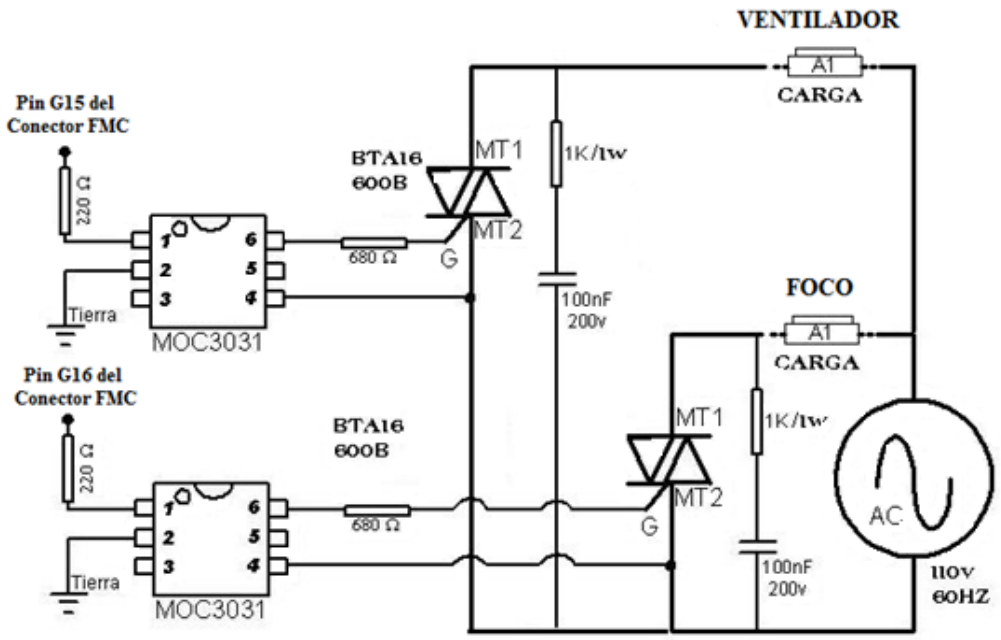

Figura. 6.5. Diagramas de los Circuitos de las Etapas de Potencia

El conector FMC (*FPGA Mezzanine Card*), que se menciona en la figura anterior proporciona un mecanismo para acoplar pines de entrada y salida del FPGA hacia una interfaz exterior, como es el caso de la tarjeta de acondicionamiento de entradas y salidas (Figura. 6.6).

	K	J	H	G	F	E	D	C	B	A
1	NC	NC	VREF_A_M2C	GND	NC	NC	PG_C2M	GND	NC	NC
2	NC	NC	PRSNT_M2C_L	CLK1_M2C_P	NC	NC	GND	DP0_C2M_P	NC	NC
3	NC	NC	GND	CLK1_M2C_N	NC	NC	GND	DP0_C2M_N	NC	NC
4	NC	NC	CLK0_M2C_P	GND	NC	NC	GBTCLK0_M2C_P	GND	NC	NC
5	NC	NC	CLK0_M2C_N	GND	NC	NC	GBTCLK0_M2C_N	GND	NC	NC
6	NC	NC	GND	LA00_P_CC	NC	NC	GND	DP0_M2C_P	NC	NC
7	NC	NC	LA02_P	LA00_N_CC	NC	NC	GND	DP0_M2C_N	NC	NC
8	NC	NC	LA02_N	GND	NC	NC	LA01_P_CC	GND	NC	NC
9	NC	NC	GND	LA03_P	NC	NC	LA01_N_CC	GND	NC	NC
10	NC	NC	LA04_P	LA03_N	NC	NC	GND	LA06_P	NC	NC
11	NC	NC	LA04_N	GND	NC	NC	LA05_P	LA06_N	NC	NC
12	NC	NC	GND	LA08_P	NC	NC	LA05_N	GND	NC	NC
13	NC	NC	LA07_P	LA08_N	NC	NC	GND	GND	NC	NC
14	NC	NC	LA07_N	GND	NC	NC	LA09_P	LA10_P	NC	NC
15	NC	NC	GND	LA12_P	NC	NC	LA09_N	LA10_N	NC	NC
16	NC	NC	LA11_P	LA12_N	NC	NC	GND	GND	NC	NC
17	NC	NC	LA11_N	GND	NC	NC	LA13_P	GND	NC	NC
18	NC	NC	GND	LA16_P	NC	NC	LA13_N	LA14_P	NC	NC
19	NC	NC	LA15_P	LA16_N	NC	NC	GND	LA14_N	NC	NC
20	NC	NC	LA15_N	GND	NC	NC	LA17_P_CC	GND	NC	NC
21	NC	NC	GND	LA20_P	NC	NC	LA17_N_CC	GND	NC	NC
22	NC	NC	LA19_P	LA20_N	NC	NC	GND	LA18_P_CC	NC	NC
23	NC	NC	LA19_N	GND	NC	NC	LA23_P	LA18_N_CC	NC	NC
24	NC	NC	GND	LA22_P	NC	NC	LA23_N	GND	NC	NC
25	NC	NC	LA21_P	LA22_N	NC	NC	GND	GND	NC	NC
26	NC	NC	LA21_N	GND	NC	NC	LA26_P	LA27_P	NC	NC
27	NC	NC	GND	LA25_P	NC	NC	LA26_N	LA27_N	NC	NC
28	NC	NC	LA24_P	LA25_N	NC	NC	GND	GND	NC	NC
29	NC	NC	LA24_N	GND	NC	NC	TCK	GND	NC	NC
30	NC	NC	GND	LA29_P	NC	NC	TDI	SCL	NC	NC
31	NC	NC	LA28_P	LA29_N	NC	NC	TDO	SDA	NC	NC
32	NC	NC	LA28_N	GND	NC	NC	3P3VAUX	GND	NC	NC
33	NC	NC	GND	LA31_P	NC	NC	TMS	GND	NC	NC
34	NC	NC	LA30_P	LA31_N	NC	NC	TRST_L	GA0	NC	NC
35	NC	NC	LA30_N	GND	NC	NC	GA1	12P0V	NC	NC
36	NC	NC	GND	LA33_P	NC	NC	3P3V	GND	NC	NC
37	NC	NC	LA32_P	LA33_N	NC	NC	GND	12P0V	NC	NC
38	NC	NC	LA32_N	GND	NC	NC	3P3V	GND	NC	NC
39	NC	NC	GND	VADJ	NC	NC	GND	3P3V	NC	NC
40	NC	NC	VADJ	GND	NC	NC	3P3V	GND	NC	NC

Figura. 6.6. Pines de Salida del Conector FMC LPC

Fuente: XILINX, Inc., 2010

La figura anterior no sería útil sin la siguiente tabla que indica la relación de cada uno de estos pines con los pines de entrada/salida del FPGA. Solo basta con reconocer los pines que se desea utilizar, asociar el pin con la columna y la fila deseada de la Figura. 6.6, y relacionarlo con los pines del FPGA que se muestran en la Tabla. 6.26.

Columna C		Columna D		Columna G		Columna H	
Pin de FMC LPC	**Pin del FPGA**	**Pin de FMC LPC**	**Pin del FPGA**	**Pin de FMC LPC**	**Pin del FPGA**	**Pin de FMC LPC**	**Pin del FPGA**
C2	B16	D1	V13	G2	E16	H2	Y16
C3	A16	D4	E12	G3	F16	H4	H12
C6	D15	D5	F12	G6	G9	H5	G11
C7	C15	D8	F14	G7	F10	H7	G8
C10	D4	D9	F15	G9	B18	H8	F9
C11	D5	D11	C4	G10	A18	H10	C19
C14	H10	D12	A4	G12	B20	H11	A19
C15	H11	D14	F7	G13	A20	H13	B2
C18	C17	D15	F8	G15	H13	H14	A2
C19	A17	D17	G16	G16	G13	H16	H14
C22	T12	D18	F17	G18	C5	H17	G15
C23	U12	D20	Y11	G19	A5	H19	D18
C26	AA10	D21	AB11	G21	R9	H20	D19
C27	AB10	D23	U9	G22	R8	H22	R11
C30	T21	D24	V9	G24	V7	H23	T11
C31	R22	D26	U14	G25	W8	H25	V11
		D27	U13	G27	W14	H26	W11
				G28	Y14	H28	AA14
				G30	T15	H29	AB14
				G31	U15	H31	AA16
				G33	U16	H32	AB16
				G34	V15	H34	Y15
				G36	Y17	H35	AB15
				G37	AB17	H37	W17
						H38	Y18

Tabla. 6.26. Relación de Pines entre Conector FMC y FPGA

Fuente: XILINX, Inc., 2010

Para este proyecto se ha utilizado los siguientes pines (Figura. 6.5):

- G15 para la salida de *gpio_ventilador* a la etapa de potencia del elemento de enfriamiento.
- G16 para la salida del DAC a la etapa de potencia del elemento de

calefacción.

- G18 es la entrada del comparador externo LM311 al ADC.
- G19 es la salida del *DACout* del ADC que se conecta al filtro pasa bajos.

6.2.2 ACONDICIONAMIENTO DE LA SEÑAL DEL SENSOR

Como se mencionó anteriormente en la descripción del *IP Core Delta-Sigma ADC*, para implementar la conversión análogo-digital mediante modulación *delta-sigma* se necesita un comparador externo y un filtro pasa bajos. El comparador externo determinará si la señal análoga a convertir es mayor o menor a la referencia generada por el DAC interno (*DACout*) que es reconstruida en el filtro pasa bajos.

El pin del conector FMC usado como salida de referencia del ADC está conectado al banco 0 del FPGA, que trabaja con un voltaje de riel de 2.5V. Este voltaje determina el rango analógico que puede ser convertido por el ADC (0-2.5V), utilizando su salida de referencia, la cual se conecta a una de las entradas del comparador externo. Por lo tanto la señal del sensor LM35, que se conecta a la otra entrada del comparador externo, también debe variar de 0 a 2.5V.

Tomando en cuenta que el *IP Core delta-sigma ADC* no trabaja linealmente en todo su rango y que la temperatura no será controlada por debajo de la temperatura ambiente, se ha determinado un rango de trabajo del sensor LM35 de 0 a 100 grados centígrados. El sensor LM35 proporciona 10mV por cada grado centígrado, por tanto si el rango de medición de temperatura es de 0 a 100 grados C, el sensor proporcionará una señal que variará entre 0 y 100mV. Para aprovechar el rango de voltaje analógico que puede ser convertido por el ADC la ganancia de esta etapa deber ser de 2.5.

Se va a implementar una etapa de acondicionamiento basada en amplificador operacional en configuración no inversor, como se muestra en la Figura. 6.7.

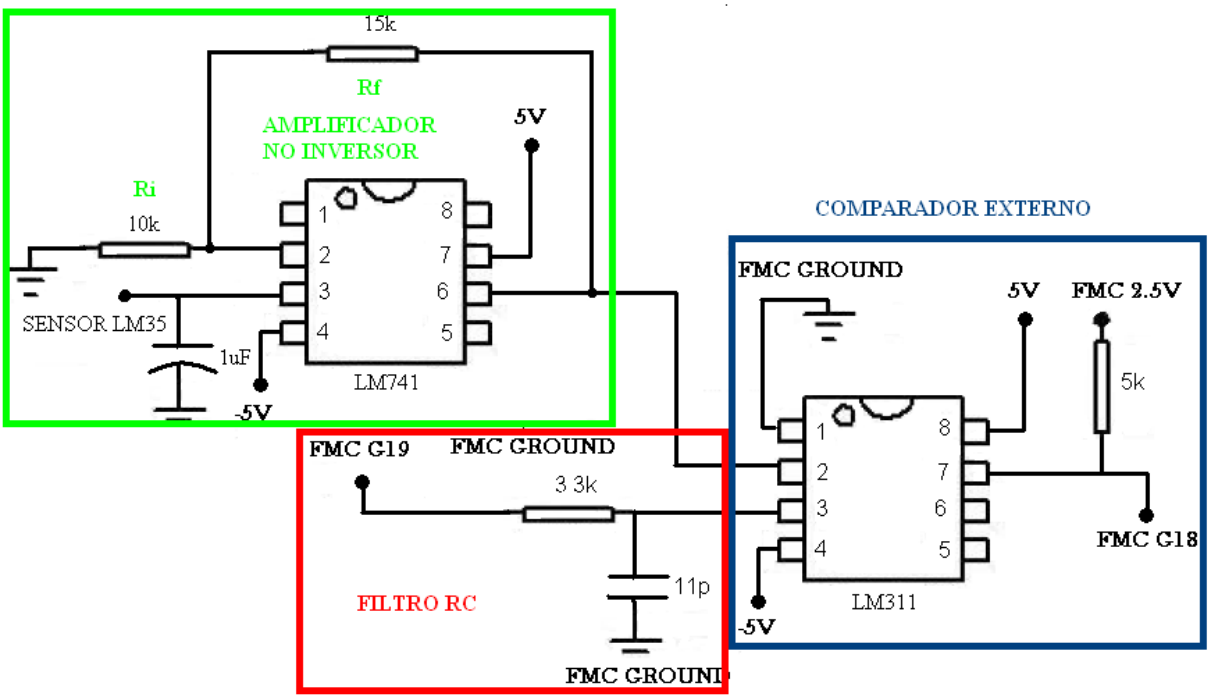

Figura. 6.7. Amplificador No – Inversor, Comparador Externo del ADC, y Filtro Pasabajos RC

Para obtener la ganancia mencionada en el amplificador no inversor, se realizaron los siguientes cálculos:

$$Vout = \left(1 + \frac{Rf}{Ri}\right) Vin$$

Ecuación 6.1

Datos:

Vout = 2.5V;

Vin = 100mV (100 gradosC);

Se asume: Ri = 10 Kohms

$$\left(1 + \frac{Rf}{10000\Omega}\right) = 2.5$$

$$\frac{Rf}{10000\Omega} = 1.5$$

$$Rf = 15000\Omega$$

Con estos valores de resistencias se asegura que el ADC ocupe todo el rango deseado para la adquisición de datos.

Cabe mencionar que después de implementar el conversor análogo-digital basado en el *IP Core delta - sigma ADC*, se determinó que el rango donde este conversor tiene el menor error, está aproximadamente entre 15 y 70 grados centígrados (Figura. 6.8). Por tanto, este *IP Core* tiene lo necesario para cumplir con las especificaciones de diseño de esta aplicación, donde el rango a controlar es de 40 a 65 grados centígrados.

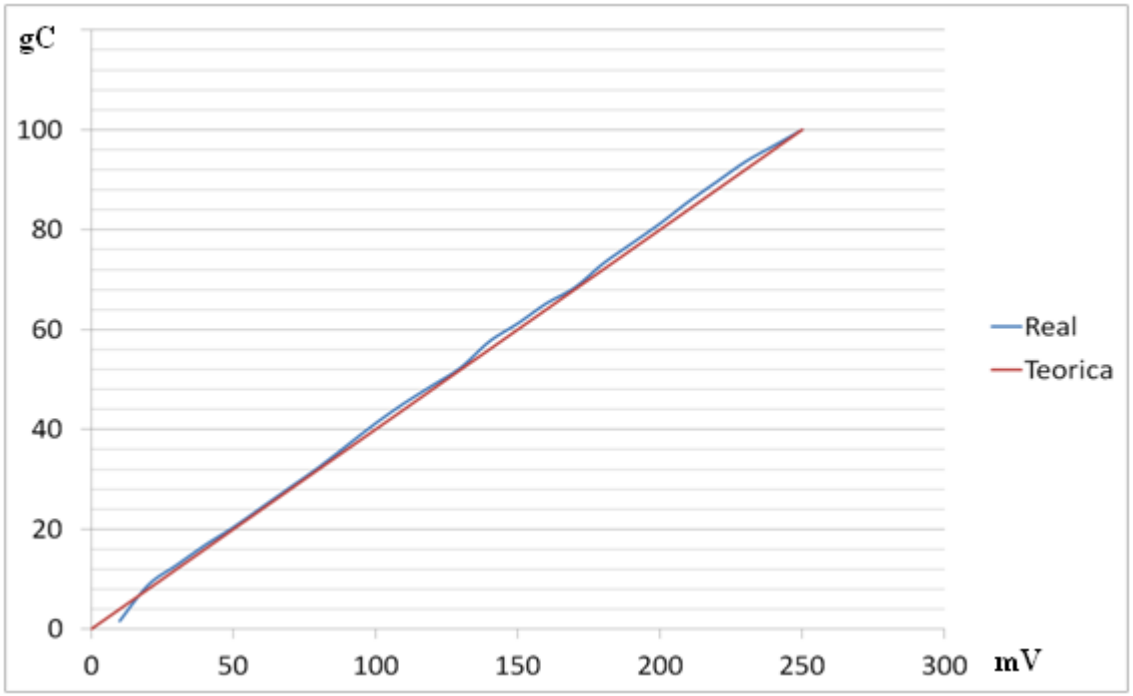

Figura. 6.8. Respuesta real y teórica del ADC.

6.3 DISEÑO DE SOFTWARE

6.3.1 CREACIÓN DE UN WORKSPACE EN SDK

El proyecto en SDK de esta aplicación de control contiene las tres capas básicas de un SoC dentro del *SDK_Workspace*, como se muestra en la Figura. 6.9.

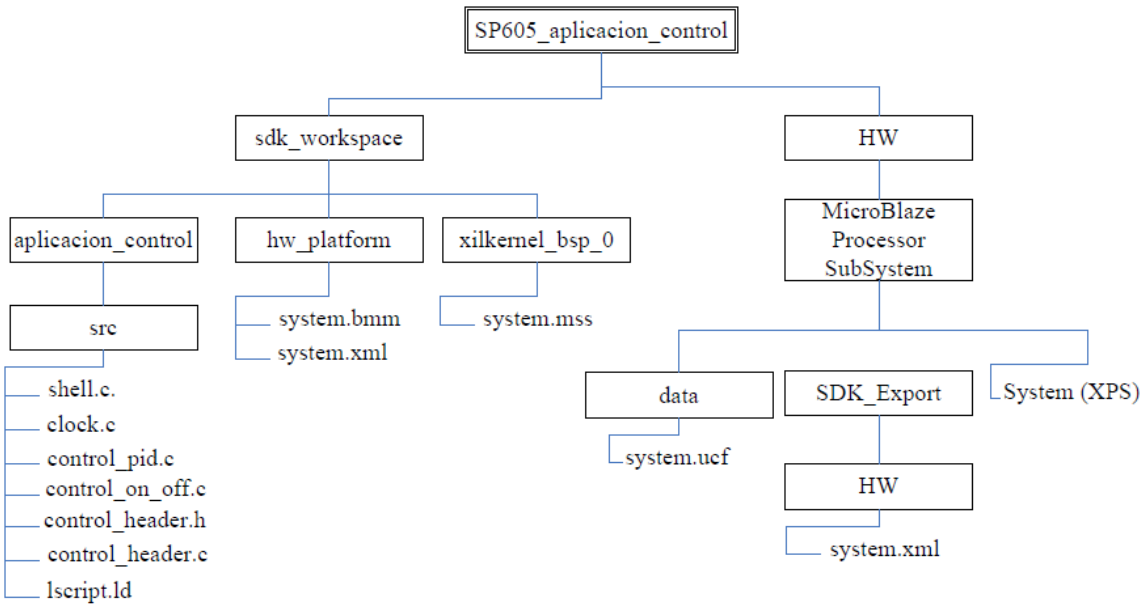

Figura. 6.9. Directorio del proyecto

En el Workspace se debe importar la plataforma de *hardware* generada previamente, además de crear un nuevo *Board Support Package* para facilitar el uso de drivers y librerías correspondientes a los *IP Cores* añadidos anteriormente, utilizando Xilkernel como RTOS. La Figura. 6.10, muestra el contenido del Workspace del SDK y su representación en capas del proyecto.

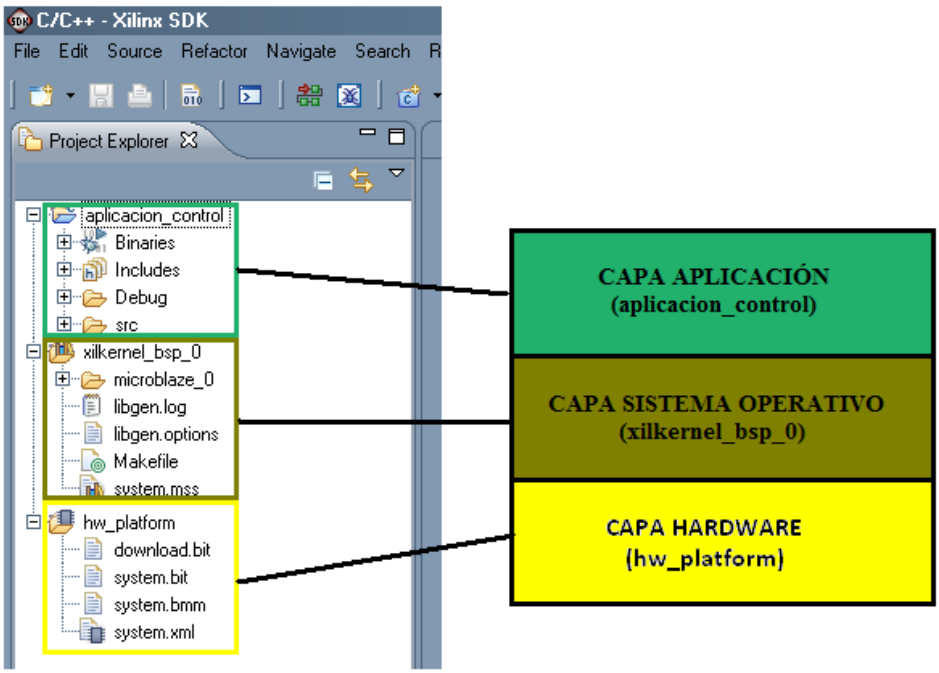

Figura. 6.10. Proyecto en SDK (izquierda) y modelo de capas del sistema (derecha)

6.3.2 IMPORTACIÓN DE LA PLATAFORMA DE HARDWARE

CAPA DE HARWARE (hw_platform)

La carpeta de esta capa contiene la plataforma de *harware* importada desde el XPS, junto con el mapa de memoria y el *bitstream* del sistema.

6.3.3 CREACIÓN Y CONFIGURACIÓN DEL BOARD SUPPORT PACKAGE

CAPA DE SISTEMA OPERATIVO (xilkernel_bsp_0)

La carpeta de esta capa contiene el RTOS Xilkernel junto con los drivers y librerías necesarios para el trabajo con los dispositivos de *hardware*, hilos, semáforos e interrupciones. Cabe mencionar, que el archivo *system.mss* de esta carpeta contiene, entre otras cosas, la configuración del RTOS Xilkernel. Esta configuración es la siguiente:

```
BEGIN OS
 PARAMETER OS_NAME = xilkernel
 PARAMETER OS_VER = 5.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
 PARAMETER SYSTMR_SPEC = true
 PARAMETER SYSTMR_DEV = timer_xilkernel
 PARAMETER SYSINTC_SPEC = Interrupt_Cntlr
 PARAMETER SYSTMR_INTERVAL = 100
 PARAMETER PTHREAD_STACK_SIZE = 10000
 PARAMETER SCHED_TYPE = SCHED_PRIO
 PARAMETER N_PRIO = 6
 PARAMETER CONFIG_SEMA = true
 PARAMETER STATIC_PTHREAD_TABLE = ((shell_main,1))
END
```

DRIVERS

En la herramienta *Board Support Package* BSP, se encuentra toda la información necesaria acerca de los drivers para los dispositivos de *hardware* del sistema. En la Tabla. 6.27, se encuentra una lista de los *IP Cores* con sus respectivos *drivers*.

COMPONENTE	TIPO DE COMPONENTE	DRIVER	VERSION DEL DRIVER
microblaze_0	Microblaze	Cpu	1.12.b
ADC	xps_deltasigma_adc	Dsadc	2.00.a
DAC	xps_deltasigma_dac	Dsdac	2.00.a
DAC2	xps_deltasigma_dac	Dsdac	2.00.a
DDR3_SDRAM	Mpmc	Mpmc	4.00.a
Debug_Module	Mdm	uartlite	2.00.a
Dual_Timer_Counter	xps_timer	Tmrctr	2.00.a
FLASH	xps_mch_emc	Emc	3.00.a
GPIO_2bits2_DAC2	xps_gpio	Gpio	3.00.a
GPIO_4bits	xps_gpio	Gpio	3.00.a
Interrupt_Controller	xps_intc	Intc	2.00.a
LocalMemory_Cntrl_D	lmb_bram_if_cntrl	Bram	2.00.a
LocalMemory_Cntrl_I	lmb_bram_if_cntrl	Bram	2.00.a
RS232_Uart_1	xps_uart16550	uartns550	2.00.a
SPI_FLASH	xps_spi	Spi	3.00.a
Soft_TEMAC	xps_ll_temac	Lltemac	3.00.a
clock_timer	xps_timer	Tmrctr	2.00.a
sample_timer	xps_timer	Tmrctr	2.00.a

Tabla. 6.27. Drivers del Proyecto de Automatización y Control

Esta lista de drivers se la puede encontrar en *Xilinx Tools* → *Board Support Package Settings* (Figura. 6.11). En este panel se puede elegir el *driver* que se desea asignar a un determinado periférico, se recomienda dejar los *drivers* elegidos por defecto.

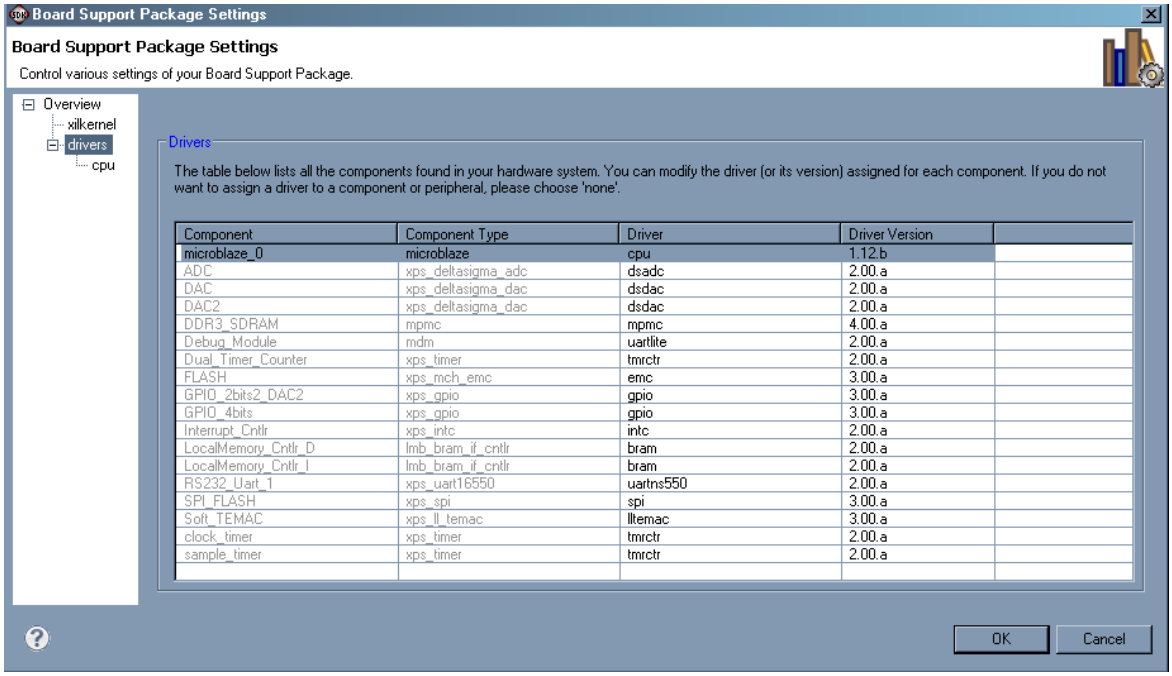

Figura. 6.11. Drivers en BSP Settings

La información necesaria de cada *driver* se detalla en el archivo *system.mss*. Para esto, dar *click* en la documentación (*Documentation*) del *driver* que se desea estudiar, como lo indica la Figura. 6.12.

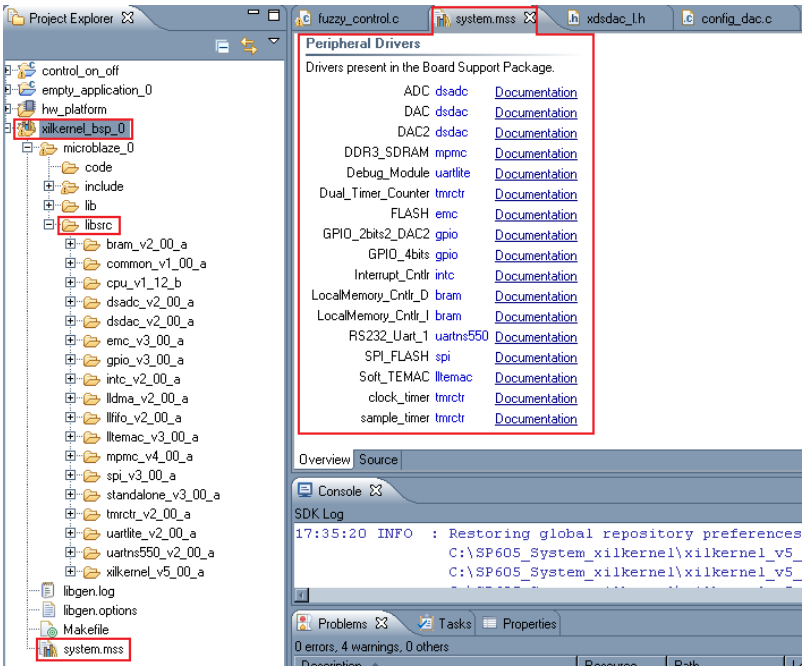

Figura. 6.12. Drivers en xilkernel_bsp_0 del workspace del SDK

LIBRERÍAS

Cuando una plataforma de software es construida en el SDK, los siguientes componentes de software se incluyen automáticamente en la librería:

- Librerías estándar C (lib.a, libm.a)
- Drivers para todos los periféricos del proyecto de diseño (libxil.a)

Además, dependiendo de la aplicación existen otras librerías opcionales que pueden trabajar junto a la colección de librerías de software de Xilinx:

- **LibXil MFS** – Sistema de Archivos de Memoria
- **LibXil FATFS** – Sistema de archivos de Xilinx SysAce basados en FAT.
- **LibXil Flash** – Una librería que proporciona lectura/escritura/borrado/bloqueo/desbloqueo y funcionalidades específicas para dispositivos paralelos flash.
- **LibXil Isf** – La librería *In-System-Flash* admite el hardware de Xilinx *In System Flash.*
- **lwIP** – Librería ligera para redes TCP/IP [58]

6.3.4 CREACIÓN DEL PROYECTO DE SOFTWARE

CAPA DE APLICACIÓN

En la carpeta de esta capa se encuentran los ficheros con el código de la aplicación y el "***linker script***" correspondiente (Tabla. 6.28). El "linker script" especifica en que memoria se almacenará las secciones de código, datos, pila y almacenamiento dinámico, al descargar la compilación del programa al FPGA.

DESCRIPCION	ARCHIVO
shell.c	Archivo propiedad de Xilinx tomado de ug758_example_design_files\xilkernel_demo\src\shell.c y adecuado para esta aplicación. Este código implementa una Shell CLI (Comand Line Interface) que contiene comandos básicos para interactuar con el sistema operativo, es capaz de cargar y ejecutar nuevos hilos. Se le ha cargado los hilos para control *on-off* y PID.
clock.c	Archivo propiedad de Xilinx tomado de ug758_example_design_files\xilkernel_demo\src\clock.c y adecuado para esta aplicación. Este archivo implementa un hilo que sirve de reloj del sistema capaz de ser seteado. Se ejecuta siempre en paralelo al resto de hilos.
control_on_off.c	Este archivo contiene el hilo para control on-off de temperatura y su bucle de control. Se lo ejecuta a través del ingreso del comando "run 0" en la Shell, y termina con la interrupción externa del pulsador de la tarjeta SP605.
control_pid.c	Este archivo contiene el hilo para control pid de temperatura y su bucle de control. Se lo ejecuta a través del ingreso del comando "run 1" en la Shell, y termina con la interrupción externa del pulsador de la tarjeta SP605.
control_header.c	Este archivo contiene funciones para realizar *setup* de *hardware*, lectura del sensor a través del ADC, inicializar el *timer* de muestreo y para el ingreso de números enteros. Estas funciones son usadas en diversas partes del código por lo que conforman una librería extra, asociada a la aplicación a través del archivo de cabecera "control_header.h"
control_header.h	Archivo de cabecera que contiene los prototipos de las funciones del archivo control_header.c, además de definiciones generales de la aplicación.

DESCRIPCION	ARCHIVO
lscript.ld	*Linker Script* de la aplicación. Asocia todas las secciones del programa a la memoria DDR3 externa de la tarjeta SP605. Define un tamaño de pila (stack size) de 2Kb necesario para ejecutar los hilos de control.

Tabla. 6.28. Descripción de los ficheros de la capa de aplicación

El código de cada uno de los archivos mencionados en la tabla anterior se encuentra en el Anexo A13. A continuación se muestra una breve descripción de las rutinas principales de código y su respectivo diagrama UML.

Función *main():* (Figura. 6.13) Esta rutina es la primera en ejecutarse y realiza el *setup* de *hardware* y la inicialización de Xilkernel. Al iniciar Xilkernel se crean dos hilos que se ejecutan en paralelo:

- *shell_main():* Hilo principal definido por el programador.
- *idle_task():* Hilo definido por Xilkernel.

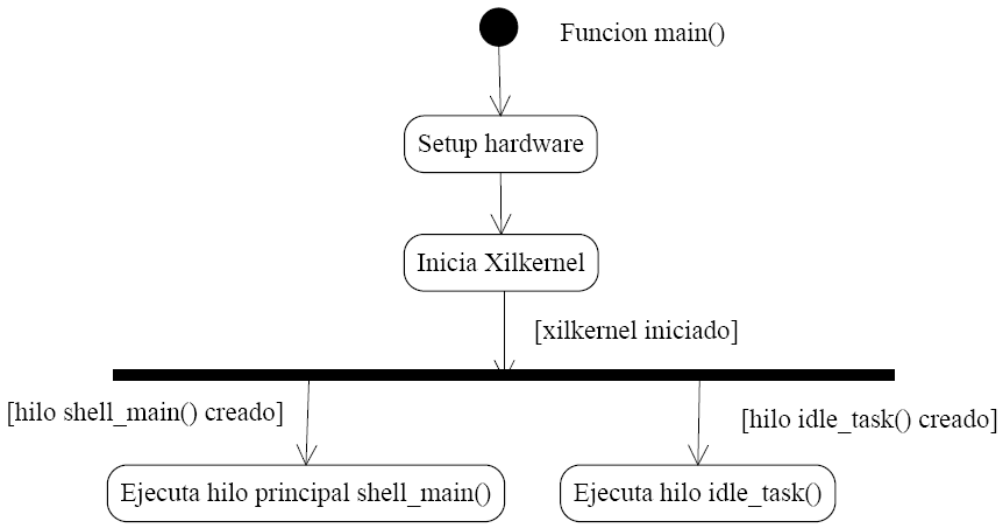

Figura. 6.13. DiagramaUML función *main().*

Hilo *shell_main():* (Figura. 6.14) Constituye el hilo principal, y se encarga de crear el hilo *clock_main(),* y de ejecutar la Shell CLI que contiene los siguientes comandos:

- *help:* Imprime lista de comandos.

- *clear:* Limpia la pantalla
- *list:* Imprime lista de hilos cargados.
- *run X:* Crea hilo X (*run 0* crea hilo *on_off_main() – run 1* crea hilo *pid_main()*).
- *time:* Imprime hora del sistema.
- *time HHMM:* Setea la hora del sistema en HH horas y MM minutos.
- *exit:* Finaliza hilo *Shell_main().*

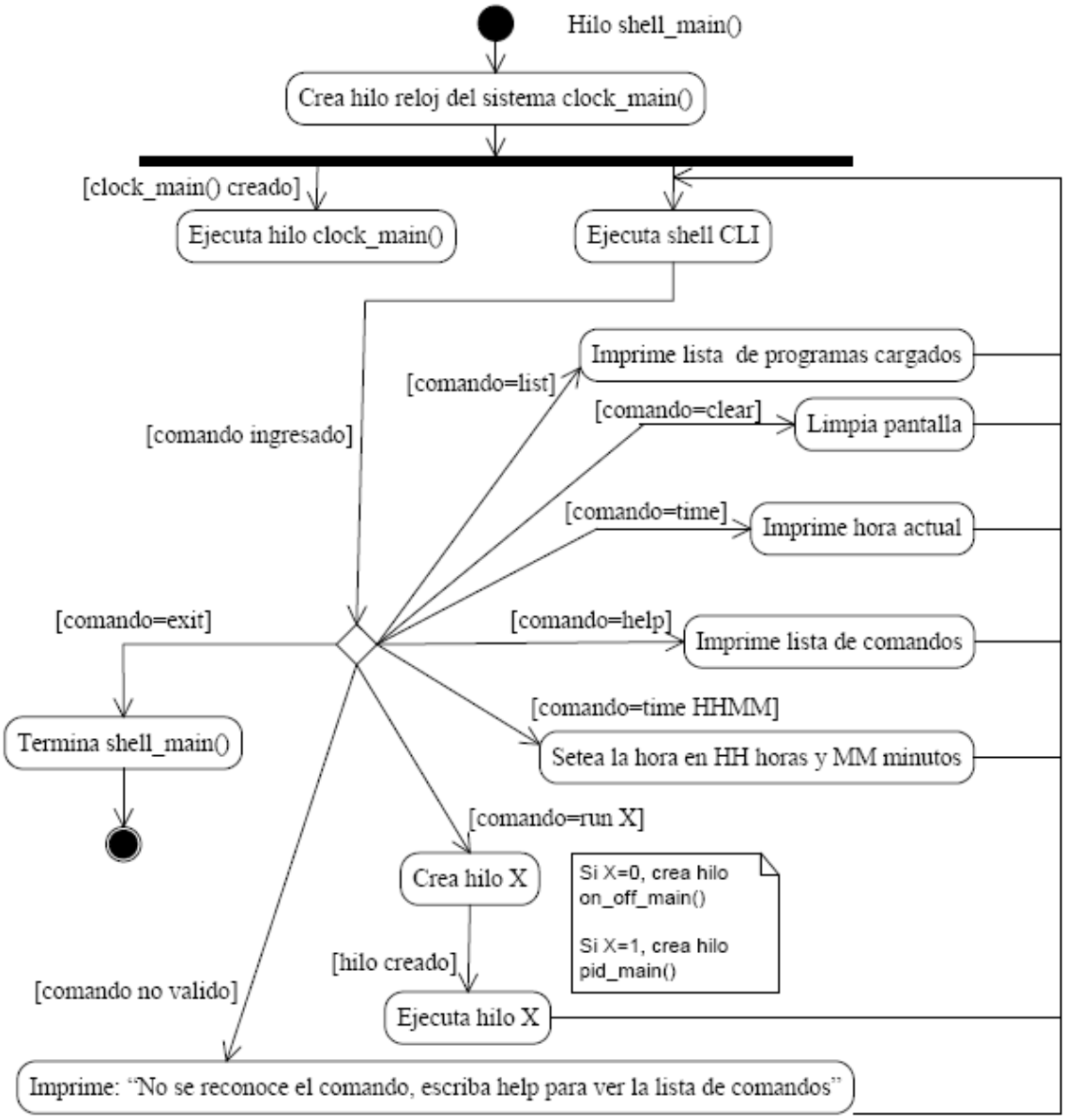

Figura. 6.14. Diagrama UML hilo *shell_main()*

Hilo *clock_main():* (Figura. 6.15). Este hilo constituye el reloj del sistema y se ejecuta en paralelo al resto de hilos. Se encarga de actualizar y setear la hora del sistema.

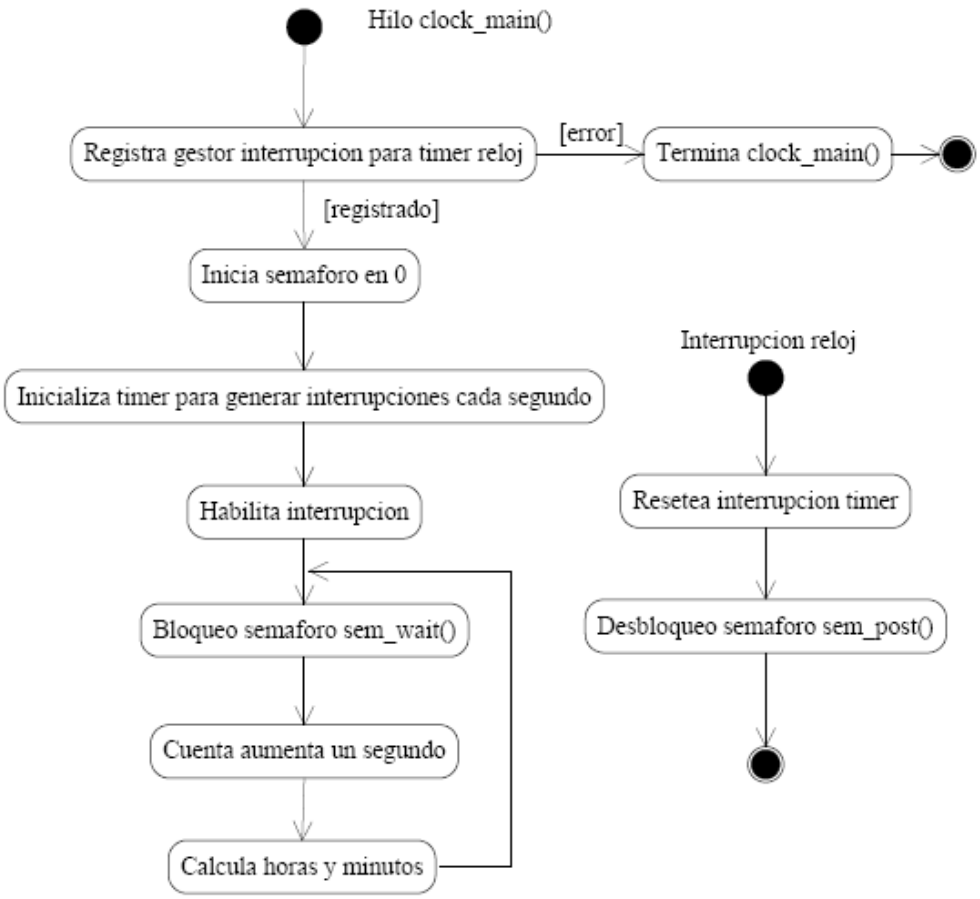

Figura. 6.15. Digrama UML hilo c*lock_main()*.

Hilo *on_off_main():* (Figura 6.16) Este hilo realiza el ingreso del *set point* y el control *on-off* de temperatura. Al ejecutarse deja al hilo principal s*hell_main()* en espera y termina con la interrupción externa del pulsador de la tarjeta SP605.

Hilo *pid_main():* (Figura. 6.17) Este hilo realiza el ingreso del *set point* y el control PID de temperatura. Al ejecutarse deja al hilo principal s*hell_main()* en espera y termina con la interrupción externa del pulsador de la tarjeta SP605.

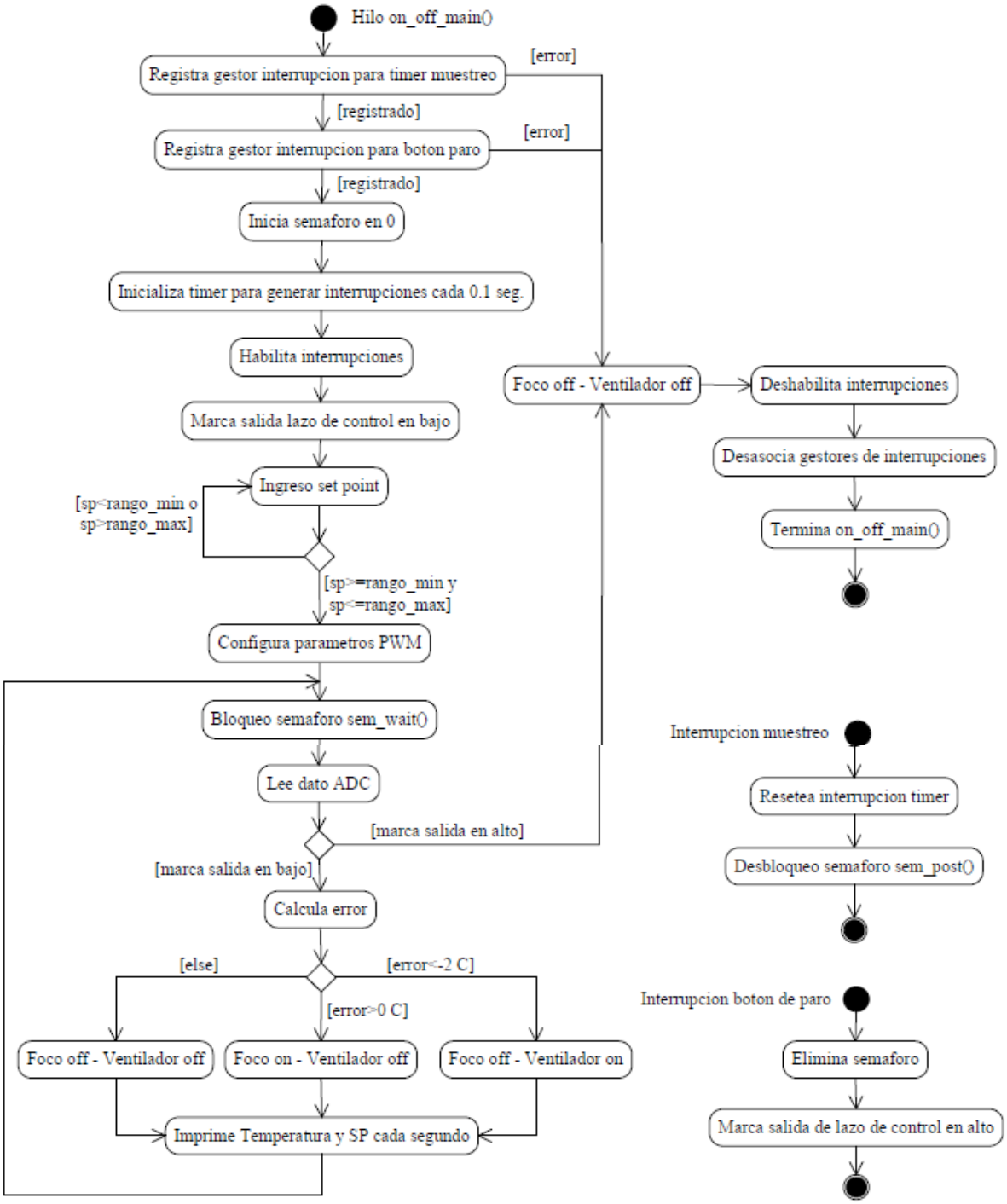

Figura. 6.16. Diagrama UML hilo *on_off_main().*

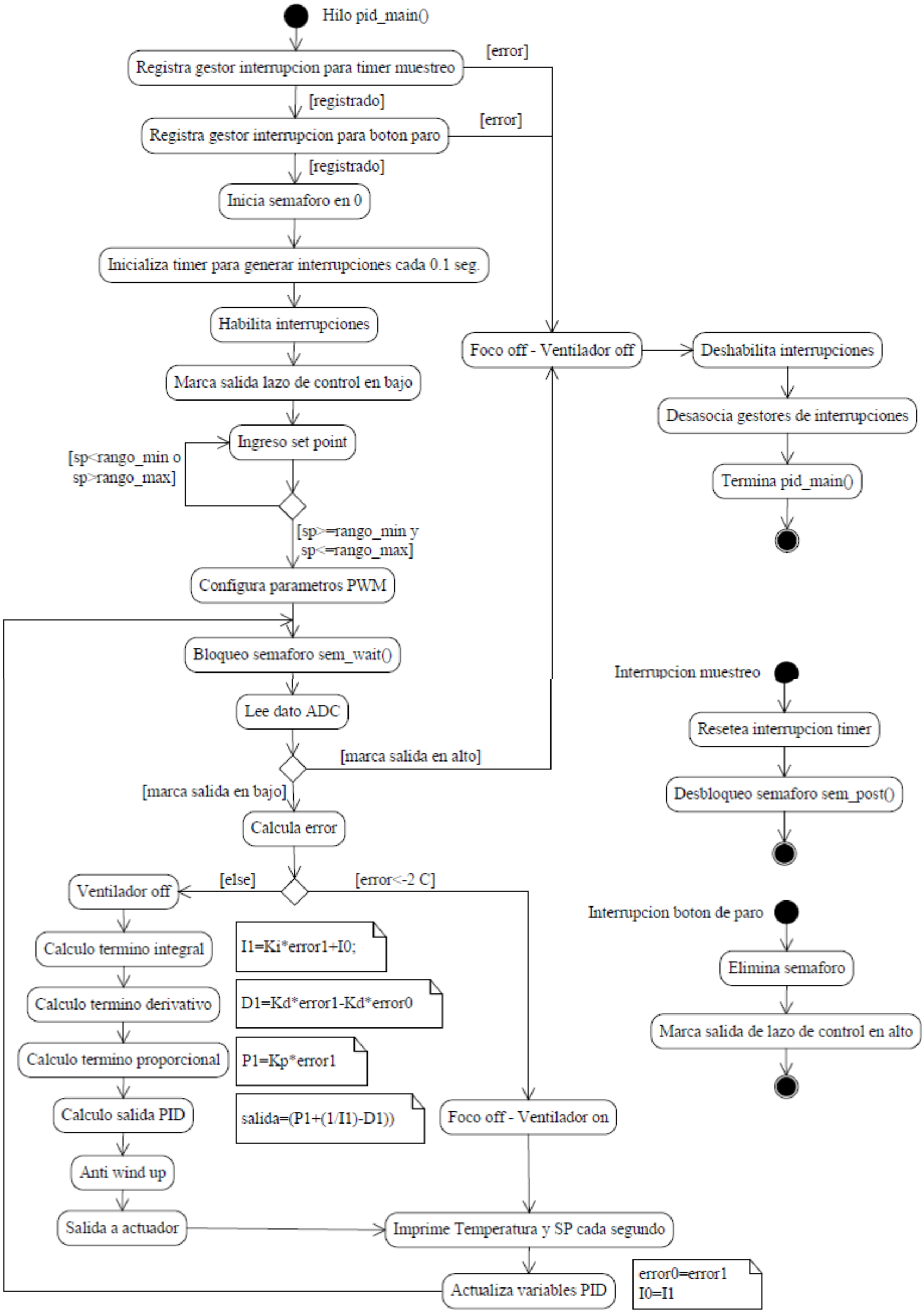

Figura. 6.17. Diagrama UML hilo *pid_main()*.

Para finalizar con la descripción del *software,* cabe mencionar, que esta aplicación tiene la capacidad de adjuntar nuevos hilos a su lista de programas cargados. Para esto se debe incluir la librería *xmk.h* al inicio del código del hilo y modificar/agregar líneas especificass del archivo shell.c, como se muestra en el siguiente ejemplo.

```
#define PROGS_LOADED 3 (aumenta en 1 por cada hilo adicional)
static char * progs_loaded[PROGS_LOADED] = {
    "0: ON-OFF          : Iniciar controlador ON-OFF" ,
    "1: PID             : Iniciar controlador PID" ,
    "3: NuevoHilo       : Descripcion del nuevo hilo" ,
};
extern void* NuevoHilo_main (void *);
static prog_info_t proginfo [PROGS_LOADED] = {
                        {on_off_main, 0} ,
                        {pid_main, 0},
                        { NuevoHilo_main, 0},
};
```

CAPÍTULO 7

RESULTADOS OBTENIDOS

Aplicando el concepto de co-diseño de hardware y software, y la metodología PBD se diseñó un SoC para controlar la temperatura de una planta. El proceso de diseño del SoC, explicado en el capitulo 2, se completó como estaba previsto hasta la etapa de emulación sobre la tarjeta SP605. A continuación se presentarán los resultados que describen la configuración final de cada capa del SoC empleado como sistema de control de temperatura.

7.1 CAPA DE HARDWARE

El mapa de memoria a través del cual el procesador microblaze_0, accede a los registros internos de cada uno de estos *IP Cores* se muestra en la Figura. 7.1. Además, los *IP cores* utilizados en la capa de *hardware* se muestran en la Figura. 7.2.

Mapa de Memoria para microblaze_0

Nombre Módulo	Dirección Base	Dirección Superior
LocalMemory_Cntlr_D	0x00000000	0x00001fff
LocalMemory_Cntlr_I	0x00000000	0x00001fff
Debug_Module	0x84400000	0x8440ffff
RS232_Uart_1	0x83e00000	0x83e0ffff
DDR3_SDRAM	0x88000000	0x8fffffff
Interrupt_Cntlr	0x81800000	0x8180ffff
timer_xilkernel	0x83c00000	0x83c0ffff
timer_clock	0x83c40000	0x83c4ffff
timer_sample	0x83c20000	0x83c2ffff
PWM	0x83c60000	0x83c6ffff
ADC	0x80400000	0x8040ffff
gpio_ventilador	0x81400000	0x8140ffff

Figura. 7.1. Mapa de memoria para *microblaze_0*.

IP Cores presentes en el diseño

Nombre Módulo	IP Core	Versión
Debug_Module	mdm	1.00.g
microblaze_0	microblaze	7.30.a
mb_plb	plb_v46	1.04.a
ilmb	lmb_v10	1.00.a
dlmb	lmb_v10	1.00.a
LocalMemory_Cntlr_D	lmb_bram_if_cntlr	2.10.b
LocalMemory_Cntlr_I	lmb_bram_if_cntlr	2.10.b
lmb_bram	bram_block	1.00.a
RS232_Uart_1	xps_uart16550	3.00.a
clock_generator_0	clock_generator	4.00.a
DDR3_SDRAM	mpmc	6.00.a
proc_sys_reset_0	proc_sys_reset	2.00.a
Interrupt_Cntlr	xps_intc	2.01.a
timer_xilkernel	xps_timer	1.02.a
timer_clock	xps_timer	1.02.a
timer_sample	xps_timer	1.02.a
PWM	xps_timer	1.02.a
ADC	xps_deltasigma_adc	1.01.a
gpio_ventilador	xps_gpio	2.00.a

Figura. 7.2. *IP Cores* presentes en la capa de *hardware*.

Con los *IP Cores* mostrados anteriormente y el SoC *Microblaze Processor Subsystem* como plataforma base, se implementó el SoC final (Figura. 7.3):

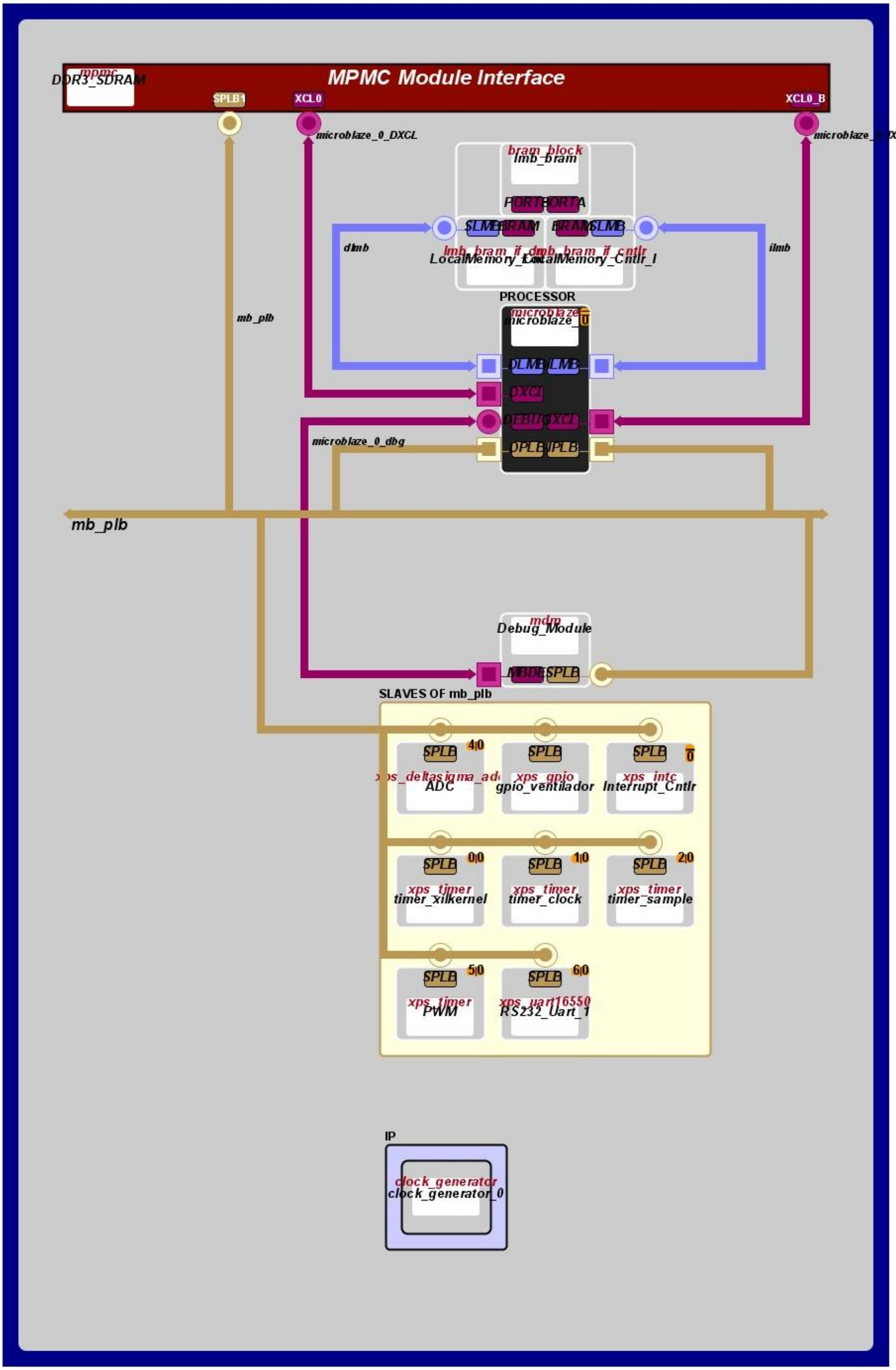

Figura. 7.3. Vista RTL del SoC Final

La figura anterior muestra que se cuenta con dos *IP Cores* adicionales, en comparación al diagrama de de bloques esperado. El primero es el *IP Core clock_generator* que recibe la señal de reloj externa de 200MHz y genera la señal de reloj del sistema de 100 MHz. El segundo es el *IP Core microblaze debug module (mdm)* que permite la depuración del sistema a través del JTAG. Los componentes restantes del *hardware* son los que se esperó obtener en el capítulo 5.

Los resultados de la síntesis de este *hardware* se muestran en la Figura. 7.4 y los resultados de la utilización del FPGA en la Figura. 7.5. Se puede apreciar que se ha utilizado aproximadamente un 25% de la capacidad del FPGA Spartan 6. Esto brinda la posibilidad de agregar gran cantidad de *hardware* que aumente las funcionalidades del sistema. Cabe mencionar que estas figuras son pantallasos tomados de los resultados de la aplicación que ofrece la plataforma de software.

XPS Synthesis Summary					[-]
Report	**Generated**	**Flip Flops Used**	**LUTs Used**	**BRAMS Used**	**Errors**
system	mar 18. oct 21:40:17 2011	5403	5882	14	0
clock_generator_0_wrapper	mar 18. oct 21:39:25 2011		1		0
gpio_ventilador_wrapper	mar 18. oct 20:35:26 2011	77	44		0
adc_wrapper	mar 18. oct 20:35:16 2011	217	183		0
pwm_wrapper	mar 18. oct 20:34:59 2011	361	344		0
timer_sample_wrapper	mar 18. oct 20:34:45 2011	361	344		0
timer_clock_wrapper	mar 18. oct 20:34:28 2011	361	344		0
timer_xilkernel_wrapper	mar 18. oct 20:34:15 2011	361	344		0
interrupt_cntlr_wrapper	mar 18. oct 20:34:01 2011	192	162		0
proc_sys_reset_0_wrapper	mar 18. oct 20:33:50 2011	67	52		0
ddr3_sdram_wrapper	mar 18. oct 20:33:43 2011	918	1083		0
rs232_uart_1_wrapper	mar 18. oct 20:32:07 2011	370	423		0
lmb_bram_wrapper	mar 18. oct 20:31:48 2011			4	0
localmemory_cntlr_i_wrapper	mar 18. oct 20:31:39 2011	2	6		0
localmemory_cntlr_d_wrapper	mar 18. oct 20:31:30 2011	2	6		0
dlmb_wrapper	mar 18. oct 20:31:22 2011	1	1		0
ilmb_wrapper	mar 18. oct 20:31:15 2011	1	1		0
mb_plb_wrapper	mar 18. oct 20:31:10 2011	160	423		0
microblaze_0_wrapper	mar 18. oct 20:30:49 2011	1833	2001	10	0
debug_module_wrapper	mar 18. oct 20:29:45 2011	119	120		0

Figura. 7.4. Resultados de la Síntesis del SoC.

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	4,810	54,576	8%	
Number used as Flip Flops	4,809			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	1			
Number of Slice LUTs	5,455	27,288	19%	
Number used as logic	4,343	27,288	15%	
Number used as Memory	263	6,408	4%	
Number of occupied Slices	2,076	6,822	30%	
Number of LUT Flip Flop pairs used	5,753			
Number with an unused Flip Flop	2,167	5,753	37%	
Number with an unused LUT	298	5,753	5%	
Number of fully used LUT-FF pairs	3,288	5,753	57%	
Number of unique control sets	384			
Number of slice register sites lost to control set restrictions	1,500	54,576	2%	
Number of bonded IOBs	58	296	19%	
Number of LOCed IOBs	58	58	100%	
IOB Flip Flops	1			
Number of RAMB16BWERs	14	116	12%	
Number of RAMB8BWERs	0	232	0%	
Number of BUFIO2/BUFIO2_2CLKs	2	32	6%	

Figura. 7.5. Resumen de Utilización del dispositivo FPGA.

7.2 CAPA DE SISTEMA OPERATIVO

La capa de sistema operativo del sistema, constituye un BSP que contiene el RTOS Xilkernel y los *drivers* para manejar los periféricos presentes (Figura. 7.6). El RTOS Xilkernel facilitó el desarrollo de la aplicación debido a que contiene una amplia cantidad de APIs con estándar POSIX para el desarrollo de aplicaciones en tiempo real.

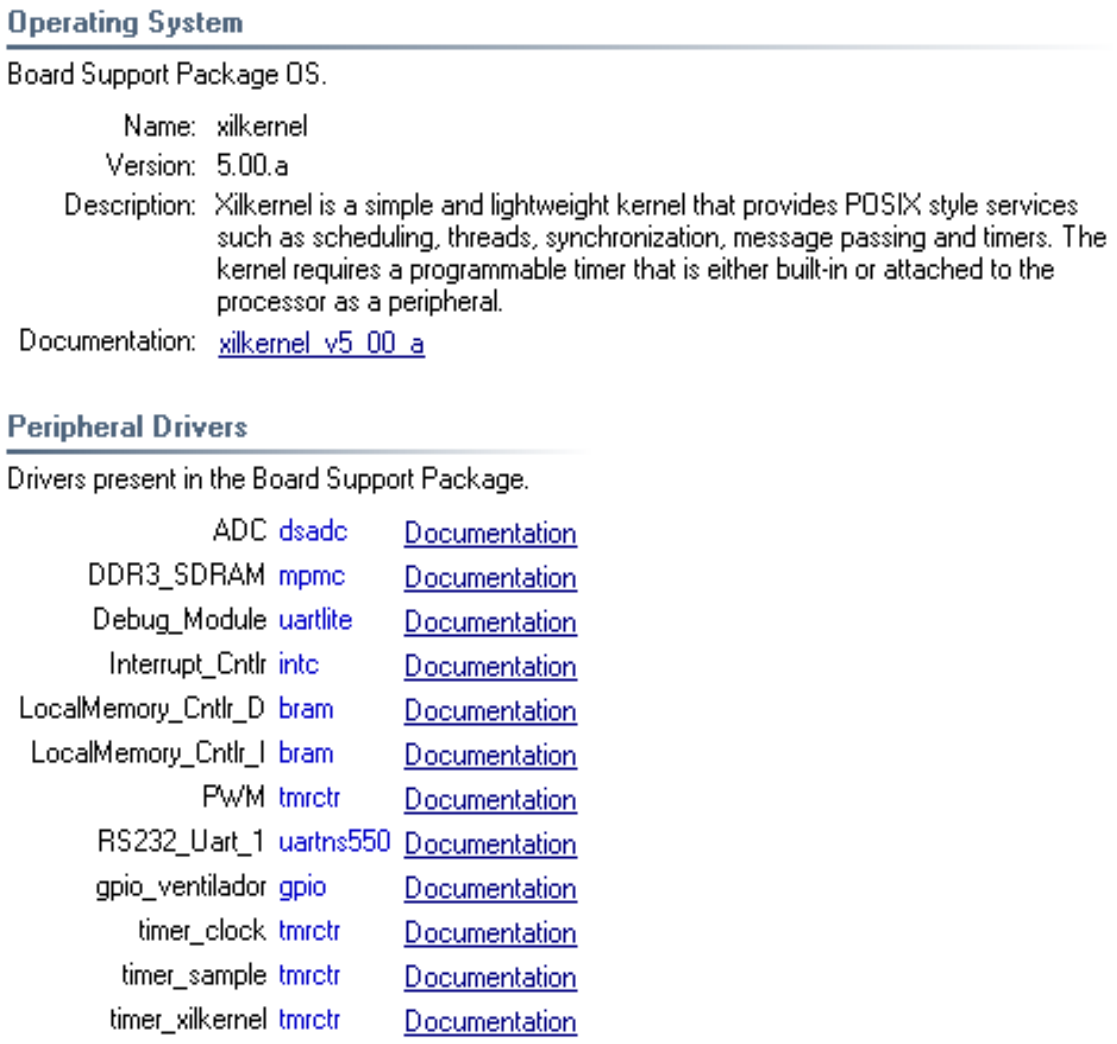

Figura. 7.6. Sistema Operativo y *Drivers* del BSP.

La configuración de los módulos del RTOS Xilkernel es la siguiente:

```
BEGIN OS
 PARAMETER OS_NAME = xilkernel
 PARAMETER OS_VER = 5.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
 PARAMETER SYSTMR_SPEC = true
 PARAMETER SYSTMR_DEV = timer_xilkernel
 PARAMETER SYSINTC_SPEC = Interrupt_Cntlr
```

```
 PARAMETER SYSTMR_INTERVAL = 100
 PARAMETER PTHREAD_STACK_SIZE = 20000
 PARAMETER SCHED_TYPE = SCHED_PRIO
 PARAMETER N_PRIO = 6
 PARAMETER CONFIG_SEMA = true
 PARAMETER STATIC_PTHREAD_TABLE = ((shell_main,1))
END
```

7.3 CAPA DE APLICACIÓN

En la capa de aplicación del sistema se desarrolló un proyecto de *software* que contiene una Shell CLI. Esta Shell ejecuta varios comandos para interactuar con el usuario y permite enlistar y ejecutar hilos previamente probados. En este caso facilitó la integración de los hilos de control On-Off y PID a la aplicación. A continuación se muestra el funcionamiento de este programa al iniciar el sistema.

```
SETUP: Plataforma de hardware inicializada correctamente.
Iniciando Xilkernel...
SHELL: Xilkernel inicializado
SHELL: Inicializando reloj...
RELOJ: Registrado gestor de interrupciones para el timer del reloj.
RELOJ: Configurando timer del reloj para generar interrupciones cada segundo ..
RELOJ: Interrupcion de reloj habilitada ...

shell>
```

El funcionamiento de los comandos es el siguiente:

Comando ***help:***

```
shell>help
help
```

Lista de comandos

run <program_num> : Ejecuta un programa. Ejemplo. "run 0" ejecuta primer programa.
time ?HHMM? : Setea/muestra hora actual.
clear : Limpia la pantalla
list : Lista de los programas cargados
help : Muestra esta pantalla de ayuda
exit : Salir

Comando ***time:***

shell>time 0637
shell>time
La hora es: 06:37:01.
shell>time 0954
shell>time
La hora es: 09:54:06

Comando ***list:***

shell>list
list

Lista de programas cargados en el sistema

0: ON-OFF : Iniciar controlador ON-OFF
1: PID : Iniciar controlador PID

Comando ***run:***

Ingresando ***run 0*** se ejecuta el hilo de control On-Off.

shell>run 0

```
run 0
-- shell ingresando a modo de espera para unirse al programa inicializado.
ON-OFF: Registrado gestor de interrupiones para el timer de muestreo.
ON-OFF: Registrado gestor de interrupiones para el boton de paro.
CONTROL: Configurando timer del reloj para generar interrupiones cada 0.1 segundos ....
ON-OFF: Interrupciones habilitadas...
ON-OFF: Ingrese el setpoint
ON-OFF: SP [grados C]=
```

Para seguir con la rutina de control On-Off ingresar el set point (SP), y para detenerla activar el pulsador SW4 de la tarjeta SP605.

```
ON-OFF: Ingrese el setpoint
ON-OFF: SP [grados C]=55
55
ON-OFF: SP=  55.
ON-OFF: Rutina de control iniciada
ON-OFF:  26 / 55 [grados C]
ON-OFF:  29 / 55 [grados C]
ON-OFF:  27 / 55 [grados C]
ON-OFF:  30 / 55 [grados C]
ON-OFF: Rutina de control finalizada.
```

Ingresando ***run 1*** se ejecuta el hilo de control PID.

```
shell>run 1
run 1
-- shell ingresando a modo de espera para unirse al programa inicializado.
PID: Registrado gestor de interrupiones para el timer de muestreo.
PID: Registrado gestor de interrupiones para el boton de paro.
CONTROL: Configurando timer del reloj para generar interrupiones cada 0.1 segundos ....
PID: Interrupciones habilitadas...
PID: Ingrese el setpoint
PID: SP [grados C]=
```

Para seguir con la rutina de control PID ingresar el set point (SP), y para detenerla activar el pulsador SW4 de la tarjeta SP605.

PID: Ingrese el setpoint
PID: SP [grados C]=55
55
PID: SP= 55.
PID: Rutina de control iniciada
PID: 26 / 55 [grados C]
PID: Salida PID=100.
PID: 26 / 55 [grados C]
PID: Salida PID=100.
PID: 26 / 55 [grados C]
PID: Salida PID=100.
PID: 28 / 55 [grados C]
PID: Salida PID=100.
PID: 28 / 55 [grados C]
PID: Salida PID=100.
PID: Rutina de control finalizada.

Comando ***exit***.

shell>exit
exit
Programa Finalizado
Adios!

La implementación de estas tres capas, constituyen un SoC que permite controlar la temperatura de una planta utilizando técnicas de control On-Off y PID. Los resultados obtenidos de la ejecución de las rutinas de control On-Off y PID, a través de las sentencias antes indicadas se muestran en la Figura. 7.7 y Figura. 7.8 respesctivamente.

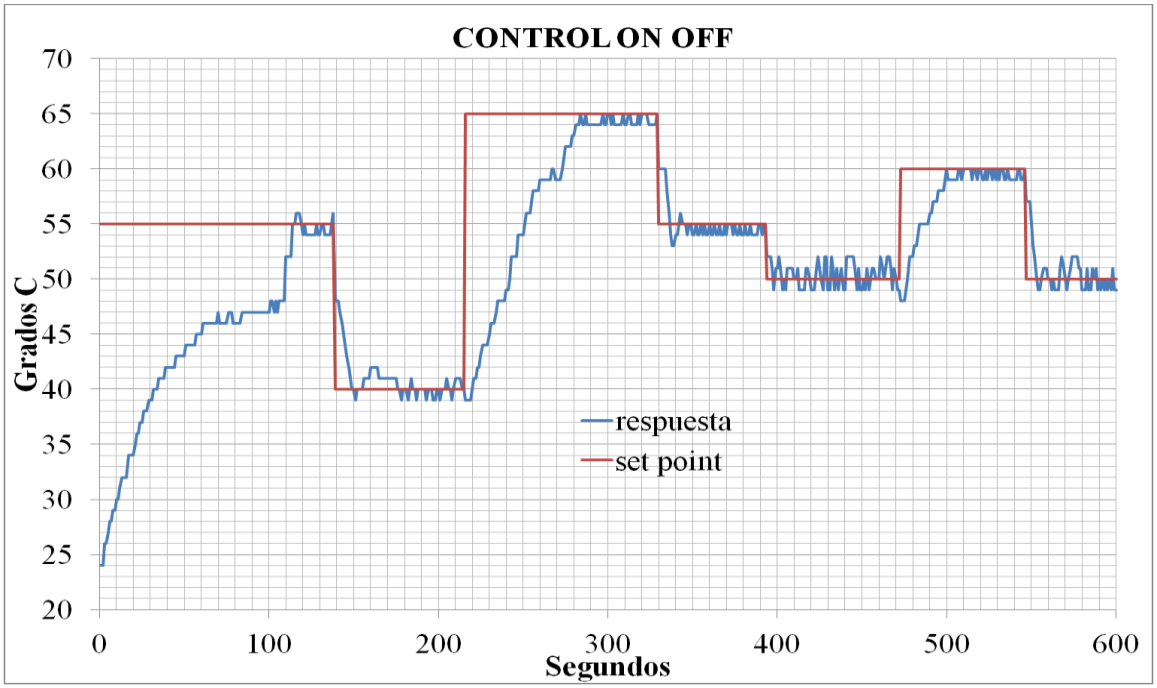

Figura. 7.7. Resultado control On-Off

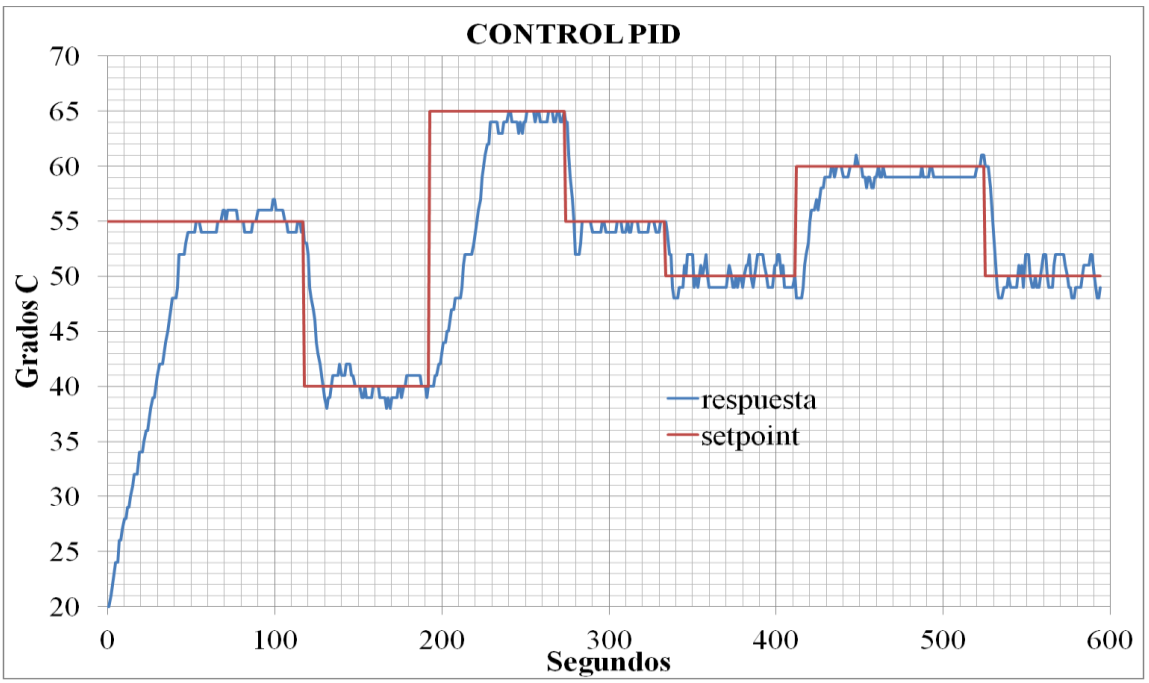

Figura. 7.8. Resultado control PID

En este caso, ambas técnicas de control cumplen con las especificaciones de diseño propuestas en el capítulo 5, donde el error máximo es de +- 2 grados C. y el rango de control es de 40 a 65 grados C. Se puede observar que la mejor respuesta se la obtiene en el control PID, ya que presenta menor tiempo de establecimiento (50 segundos) y y menor número de oscilaciones. Sin embargo, el control On-Off también presenta un tiempo de

establecimiento aceptable (110 segundos) considerando que es una planta de grado 1 y su respuesta es lenta, y a pesar de tener un mayor número de oscilaciones, logra mantener en -1 grado C. el error al enfriarse. A diferencia del control PID, donde este error alcanza los -2 grados C en varias ocasiones. Además, cabe destacar también que ambos controladores presentan seguimiento de la señal.

CAPÍTULO 8

CONCLUSIONES Y RECOMENDACIONES

Antes de comenzar con la presentación de las conclusiones y recomendaciones, cabe recordar que el objetivo principal de este proyecto de grado fue crear un *System on Chip* (SoC) orientado a una aplicación de automatización y control, mediante el co-diseño de *hardware* y *software*, empleando herramientas de Xilinx. Este objetivo fue cumplido a cabalidad al implementar sobre la tarjeta de desarrollo SP605, un SoC que incluye un Sistema Operativo en Tiempo Real (RTOS) para un sistema de control de temperatura.

Este proyecto se prolongó más de lo esperado debido a la inexperiencia y falta de conocimiento previo, en el diseño de SoCs ya que es la primera vez que se trabaja con este tipo de tecnología en el Departamento de Eléctrica y Electrónica (DEEE) de la ESPE. Sin embargo ya superada la etapa de familiarización con los conceptos de co-diseño se podrá implementar varias aplicaciones basadas en SoCs en periodos de tiempo cortos.

Se concluye que el sistema de control diseñado es fácilmente escalable para el desarrollo de nuevas aplicaciones de control, debido a la versatilidad que ofrece diseñar sobre *Field Programmable Gate Array* (FPGA). A esto se suma la flexibilidad de la arquitectura *CoreConnect* de IBM y del RTOS Xilkernel. Es decir que, usando como base el proyecto contenido en este trabajo se puede iniciar el diseño de sistemas de control más complejos.

Se determinó que el factor clave en la disminución del tiempo de diseño de *hardware* de un sistema, es la reutilización de *IP* Cores y plataformas. Esto permite diseñar *hardware* que incluya desde bloques sencillos como GPIOs hasta bloques complejos como

microprocesadores, cuyo diseño desde cero implicaría años de trabajo y gran cantidad de conocimiento.

Además, en comparación al flujo de diseño convencional sobre *hardware* no reconfigurable, se demostró que el concepto de co-diseño y el uso FPGAs permiten al diseñador intervenir en la implementación de cada capa de un sistema embebido (hardware, sistema operativo, y aplicación). Esta característica permite realizar optimizaciones en cualquier capa y en cualquier momento del diseño, de forma económicamente fiable.

Otra conclusión a la que se llegó a través de la experiencia adquirida, es que el involucrarse en el diseño e implementación de la capa de *hardware* y de sistema operativo permite una mayor comprensión de los conceptos de arquitecturas procesadas. Puesto que durante el proceso de co-diseño se personalizó y configuró elementos como: arquitectura de *hardware*, *drivers*, mapa de memoria, capacidad de almacenamiento dinámico y pila, RTOS, Shell CLI, etc.

Por otro lado, se concluye que las herramientas de diseño con altas prestaciones, como las de Xilinx, constituyen un factor clave en la rápida implementación de SoCs. La ayuda que brindan al generar automáticamente la arquitectura de *hardware*, el mapa de direcciones, la asignación de pines, el Board Support Package (BSP) y los *test* de memorias y periféricos, disminuye enormemente la complejidad de un diseño.

En base a la experiencia obtenida en el diseño de sistemas microprocesados basados en SoCs y basados en microcontroladores, se concluyen las siguientes ventajas a favor de los SoCs:

- Permiten personalizar la capa de *hardware* de un sistema a una sola aplicación sin desperdiciar recursos.
- Brindan gran flexibilidad y escalabilidad de hardware.
- Ofrecen mayor capacidad de memoria, velocidad de procesamiento y cantidad de entradas y salidas.

- Usan RTOS diseñados para ejecutarse en múltiples plataformas. Mientras que los usados en microcontroladores soportan únicamente determinados dispositivos.
- Soportan RTOS de mayor funcionalidad y capacidad.
- Facilitan el desarrollo de aplicaciones complejas en menor tiempo.

Así mismo, en comparación a los sistemas basados en PLC se concluyen las siguientes ventajas a favor de los SoCs:

- Permiten personalizar la capa de *hardware* de un sistema a una sola aplicación sin desperdiciar recursos.
- Proporcionan la capacidad de modificar la capa de sistema operativo. A diferencia de los PLCs, donde a pesar de ser la responsable de la ejecución de rutinas de gran importancia, no puede ser modificada por el usuario.
- Facilitan el desarrollo de la capa de aplicación a través de lenguajes de programación de alto nivel como C o C++.

Por último, se concluye que el presente proyecto fue el primer paso del DEEE en la incursión hacia un nuevo modelo de negocios, que se basa en el desarrollo y exportación de tecnología SoC. Dentro de este ámbito se encuentran empresas desarrolladoras de *IP Cores*, arquitecturas y plataformas de hardware, RTOS, y soluciones basadas en FPGA.

Durante el desarrollo de este proyecto se presentaron varios problemas, relacionados con los drivers de los *IP Cores*, y la limitada experiencia en la depuración de errores durante el desarrollo de un SoC. A fin de evitar posibles complicaciones y disminuir el tiempo de diseño se recomienda considerar los siguientes aspectos.

Se recomienda que antes de iniciar con el diseño de aplicaciones básicas se realice un estudio del Estado del Arte de los SoCs, seguido de una familiarización con las herramientas de diseño de Xilinx. Esto puede realizarse mediante la revisión del presente documento, puesto que recoge toda la experiencia obtenida en el cumplimiento de los objetivos de este proyecto de grado. Además, constituye un buen soporte, ya que a través de guías de estudio y tutoriales muestra paso a paso y de manera detallada el

procedimiento de co-diseño de hardware y software embebido de un *System on Chip*, con las herramientas XPS y SDK de Xilinx.

Durante el proceso de diseño de hardware en XPS, tomar en cuenta posibles cambios en la configuración de los bloques que interactuaban con un IP *Core* eliminado. Por ejemplo, al eliminar el modulo TEMAC del *MicroBlaze Processor Subsystem* se debe desasociar sus interrupciones del controlador de interrupciones e inhabilitar el puerto SDMA del MPMC.

En caso de presentar problemas que se presumen provienen de las señales internas a través de las cuales los *IP Cores* interactúan. Se recomienda usar la herramienta de depuración *ChipScope Pro*. Esta permite visualizar las señales internas del FPGA a manera de osciloscopio, supervisar buses y medir el error en las comunicaciones, insertar generadores de señales virtuales, etc.

En caso de presentarse problemas en el manejo de un *IP Core* a través de su driver, se recomienda profundizar en el estudio de su hoja de datos para poder manipular directamente sus registros de *hardware*. En el desarrollo de este proyecto se encontraron dos drivers que presentaron complicaciones. Primero, el *driver* ***dsdac v2_00_a*** del *IP Core XPS Delta-Sigma DAC,* contenía un API de *reset* que hacía referencia a un registro de *hardware* inexistente. Segundo, el *driver* ***tmrctr v2_00_a*** del *IP Core XPS Timer/Counter* no contaba con APIs para trabajar en su modo de operación PWM.

Se recomienda tomar muy en cuenta el archivo ***linker script*** de un proyecto de software. Ya que en este se especifica la memoria en la que se colocará las secciones de código, datos, pila y almacenamiento dinámico de una aplicación. Si la memoria escogida es insuficiente para almacenar una de estas secciones, el error cometido no es detectado por el compilador y se presenta deteniendo la ejecución de una aplicación sin razón aparente.

Además, se recomienda tener cuidado en la configuración de los parámetros de Xilkernel, en especial aquellos que limitan la cantidad de recursos de trabajo, como por ejemplo: *ptheard_stack_size, max_sem_waitq, max_sem, num_msgqs, max_tmrs, max_readyq, max_pthreads,* etc. Así, se recomienda aumentar el valor del parámetro

ptheard_stack_*size* (por defecto 1000) en caso de implementar hilos de gran tamaño que ejecuten interrupciones. Un error en esta configuración no es detectado por el compilador y detiene la ejecución de una aplicación.

Cabe mencionar que a pesar de que se implementó un SoC que ejecuta Xilkernel, no se ha terminado de explotar todas las capacidades de este RTOS. Por esta razón se recomienda el desarrollo de aplicaciones más complejas que incluyan Xilkernel, y que logren demostrar todas las funcionalidades de este RTOS.

Por otro lado, si bien las herramientas de Xilinx demostraron facilitar y agilizar el diseño de SoCs, se recomienda el estudio de otras opciones para el diseño de *chips* utilizando *software*, *IP Cores* y arquitecturas gratuitas. Para esto se recomienda revisar la pagina web *http://opencores.org/*.

Para finalizar, se presenta una lista de líneas de trabajo en el campo de diseño de SoCs que se recomiendan abordar. Utilizando como base el presente proyecto. Estas son:

- Bus CAN con *IP Core XPS_CAN_Controller*.
- Protocolos Ethernet Industriales con *IP Core XPS Ethernet Lite MAC*.
- *Networking* con *IP Core XPS LL TEMAC*.
- Comunicación USB con *IP Cores XPS USB Host Controller* y *XPS USB2 Peripheral*.
- Bus PCI con *IP Core PCIPLBv46 RC/EP Bridge for PCI Express*.
- Sistema con dos procesadores utilizando *IP Cores XPS Mailbox* y *XPS Mutex*.
- Sistema que incluya *IP Cores* creados por el usuario, lo que incluye la creación de *drivers*.
- Depuración de *hardware* con *ChipScope Pro*
- Profundizar en el uso de Xilkernel
- Estudio de otros RTOS como Petalinux.
- Implementar técnicas de control adaptativo, difuso, o por redes neuronales en un SoC.
- Desarrollar control y monitoreo de procesos con HMI sobre un SoC.
- Estudio de la arquitectura AMBA de ARM.

CAPÍTULO 9

BIBLIOGRAFÍA

[1] MARTIN, grant y CHANG, henry, *Winning the SoC Revolution - Experiences in Real Design*, Kluwer Academic Publisher, Estados Unidos 2003, 311 páginas.

[2] NAVAS, Byron, *Chips Diseñados en Ecuador,* Revista E-Ciencia ESPE, Edición 2, Diciembre 2009.

[3] SOCCENTRAL, *Design Reuse - Its Time for New IP-Creation Tools*, http://www.soccentral.com/results.asp?CatID=488&EntryID=31279, 2010, Fecha de Consulta: Junio 20, 2010.

[4] MARTIN, grant, *System-on-Chip and Network-on-Chip Design*, Embedded Systems Handbook, Editor: ZURAWSKI, richard, Taylor & Francis Group, 2006.

[5] CHU, pong, *FPGA Prototyping by VHDL Examples*, John Wiley & Sons Inc., 2008, 440 páginas.

[6] SMITH, michael, *Application-Specific Integrated Circuits*, Addison-Wesley, 1997, 1040 páginas.

[7] XILINX, Inc., *Platform Studio and the Embedded Development Kit*, http://www.xilinx.com/tools/platform.htm, 2010, Fecha de Consulta: Febrero 13, 2010

[8] ESTEVES KRASTEVA, yana, *Computación reconfigurable basada en FPGAs comerciales. Soluciones para el diseño e implementación de sistemas parcialmente reconfigurables*, http://biblioteca.universia.net/html_bura/ficha/params/id/49289992.html, 2009, Fecha de Consulta: Marzo 13, 2010

[9] MARTIN, grant, *State-of-the-Art SoC Communication Architectures*, Embedded Systems Handbook, Editor: ZURAWSKI, richard, Taylor & Francis Group, 2006.

[10] KARLSRUHE, *Programa De Maestría En Ingeniería De Sistemas Embebidos, Alemania*, http: //www.master-maestrias.com, Fecha de Consulta: Marzo 21, 2010

[11] BOEMO, eduardo, *Estado del Arte de la Tecnología FPGA*, http://utic.inti.gov.ar/publicaciones/cuadernilloUE/CT_Microelectronica17_FPGA.pdf, 2005, Fecha de Consulta: Marzo 05, 2010

[12] ALEXANDROV, oleg, *System-on-a-Chip*, http://en.wikipedia.org/wiki/System-on-a-chip, 2010, Fecha de Consulta: Junio 22, 2010

[13] SANDER, ingo, *SoC Architectures*, http://www.imit.kth.se/courses/2B1448/, 2006, Fecha de Consulta: Junio 25, 2010

[14] ARENA Solutions, *Improving Time to Market*, http://www.arenasolutions.com/time-to-market.html, Fecha de Consulta May 4, 2010

[15] PROBELL, jonas, *Semiconductor Intellectual Property Core*, http://en.wikipedia.org/wiki/Semiconductor_intellectual_property_core&anno=2, 2006, Fecha de Consulta: Septiembre 23, 2010

[16] UNIVERSIDAD DE CONCEPCIÓN, *Plataforma*, http://www.educ.cl/index.php?option=com_content&task=view&id=21&Itemid=20#P. Fecha de Consulta: Septiembre 25, 2010

[17] XILINX, Inc., *Xilinx*, www.xilinx.com/about/index.htm, 2010, Fecha de Consulta: Oct. 28. 2010

[18] ZHENG, li-rong, *SoC Trends and Advanced SoC Enablers*, Laboratory of Electronics and Computer Systems, Royal Institute of Technology (KTH), http://www.ict.kth.se/courses/IL2210/Lectures/Lecture1_SoC_trends.pdf, Fecha de Consulta: Junio 25, 2010

[19] MARTNEZ, germán, *Capa de abstracción*, http://es.wikipedia.org/wiki/Nivel_de_abstraccion, 2010, Fecha de Consulta: Octubre 29, 2010

[20] LI, qing, *Real-Time Concepts for Embedded Systems*, CMP Books, 2003, 294 páginas.

[21] BERGER, arnold, *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*, CMP Books, Estados Unidos 2002, 237 páginas.

[22] VILLARROEL, josé, *Sistemas Empotrados*, Departamento de Informática e Ingeniería de Sistemas, Centro Politécnico Superior Universidad de Zaragoza, http://webdiis.unizar.es/~joseluis/SE.pdf., Fecha de Consulta: Diciembre 27, 2010.

[23] MARTIN, grant y CHANG, henry, *Surviving the SoC Revolution - A Guide to Platform – Based Design*, Kluwer Academic Publisher, Estados Unidos 1999, 235 páginas.

[24] MARTIN, grant, *Real-Time in Embedded Systems*, Embedded Systems Handbook, Editor: ZURAWSKI, richard, Taylor & Francis Group, 2006.

[25] XILINX, Inc., *Getting Started with the Spartan-6 FPGA SP605 Embedded Kit*, documento UG727 (v1.1), June 21, 2010.

[26] XILINX, Inc., *Hardware and Demonstration Setup Guide*, documento UG526 (v1.4), Septiembre 24, 2009.

[27] XILINX, Inc., *EDK Concepts, Tools, and Techniques*, documento UG683, 2009.

[28] XILINX, Inc., *Xilinx Platform Studio* (*XPS)*, http://www.xilinx.com/tools/xps.htm, 2011, Fecha de Consulta: Febrero 06, 2011

[29] XILINX, Inc., *Software Development Kit (SDK)*, http://www.xilinx.com/tools/sdk.htm, 2011, Fecha de Consulta: Febrero 06, 2011

[30] XILINX, Inc., *MicroBlaze Soft Processor Core*, http://www.xilinx.com/tools/microblaze.htm, 2011, Fecha de Consulta: Febrero 06, 2011

[31] XILINX, Inc., *Embedded Processing Peripheral IP Cores*, http://www.xilinx.com/ise/embedded/edk_ip.htm, 2011, Fecha de Consulta: Febrero 06, 2011

[32] GRIMALDOS, ¿Qué es GNU?, http://www.grimaldos.es/index2.php?option=com_ content&do_pdf=1&id=37, 2011, Fechad Consulta: Febrero 06, 2011

[33] AGUAYO estanislao, GONZÁLEZ, ivan, y BOEMO, eduardo, *Tutorial Xilinx MicroBlaze*, http://arantxa.ii.uam.es/~ivan/microblaze-jcra04.pdf, Fecha de Consulta: Septiembre 23, 2010

[34] XILINX, Inc., *MicroBlaze Processor Subsystem Datasheet*, documento DS757, Junio 21, 2010.

[35] XILINX, Inc., *LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a)*, http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf., Fecha de Consulta: Febrero 07, 2011

[36] XILINX, Inc., *MicroBlaze Processor Subsystem Hardware Tutorial*, documento DS728, Junio 21, 2010.

[37] XILINX, Inc., *MicroBlaze Processor Subsystem Software Tutorial*, documento DS729, Junio 21, 2010.

[38] UNIVERSIDAD DE SEVILLA, *SoC Basados en Sistemas Abiertos*, http://www.us.es/estudios/master/master_M089/asignatura_50890013, Fecha de Consulta: Mayo 28, 2011

[39] LOCKWOOD, john, *Reconfigurable System On Chip Design*, http://www.arl.wustl.edu/~lockwood/class/cs536/lecture/cs536_lecture1_reconfigurable_SOC.pdf, Fecha de Consulta: Mayo 28, 2011

[40] Chu, yu, *Introduction to System-On-Chip Design*, http://140.122.71.78:8000/speech/920430.pdf, Fecha de Consulta: Noviembre 28, 2010

[41] STANNERED, *Design Flow for a System-on-a-Chip*, http://es.wikipedia.org/wiki/Archivo:SoCDesignFlow.svg, Fecha de Consulta: Mayo 29, 2011

[42] *Entorno de Desarrollo Integrado*, http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado, Fecha de Consulta: Juni. 22, 2010

[43] *Definición de Kernel*, Diccionario de Informática, http://www.alegsa.com.ar/Dic/kernel.php, Fecha de Consulta: Mayo 30, 2011

[44] XILINX, Inc., *Configurable Embedded System Design with Xilinx FPGAs*, http://www.xilinx.com/products/technology/embedded-processing/index.htm, Fecha de Consulta: Mayo 30, 2011

[45] XILINX, Inc., *ISE User Constraints File (UCF)*, http://www.xilinx.com/itp/3_1i/data/fise/xug/chap11/xug11005.htm., Fecha de Consulta: Mayo 31, 2011

[46] STANDFORD UNIVERSITY, *Xilinx ChipScope ICON/VIO/ILA Tutorial*, http://www.stanford.edu/~phartke/chipscope_tutorial.pdf, Fecha de Consulta: Junio 06, 2011

[47] XILINX, Inc., *Workspaces*, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/SDK_doc/concepts/sdk_c_workspace.htm., Fecha de Consulta: Junio 06, 2011

[48] XILINX, Inc., *Utility Bus Split*, documento DS484 (v1.1), Abril 24, 2009.

[49] XILINX, Inc., *Utility Flip Flop*, documento DS483 (v1.1), Abril 24, 2009.

[50] *Flip Flops*, http://www.youblisher.com/p/154819-FLIP-FLOPS/, Fecha de Consulta: Junio 15, 2011

[51] XILINX, Inc., *XPS Delta-Sigma DAC*, documento DS588 (v1.01a), Abril 19, 2010.

[52] XILINX, Inc., *Hardware User Guide*, documento DS526 (v1.4), Septiembre 24, 2010.

[53] XILINX, Inc., *XPS Delta-Sigma ADC*, documento DS587 (v1.01a), Abril 19, 2010.

[54] XILINX, Inc., *Virtex Analog to Digital Converter*, documento XAPP155 (v1.1), Septiembre 23, 1999.

[55] XILINX, Inc., *XPS 16550 UART*, documento DS577 (v3.00a), Mayo 03, 2010.

[56] XILINX, Inc. *Processor Local Bus PLBv4.6*, documento DS531 (v1.04a), Abril 24, 2009.

[57] XILINX, Inc., *Local Memory Busv10*, documento DS445 (v1.00a), Abril 24, 2009.

[58] XILINX, Inc., *Software Platforms*, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/SDK_doc/concepts/sdk_c_platforms.htm, Fecha de Consulta: Junio 15, 2011

[59] XILINX, Inc., *OS and libraries Documentation Collection*, document UG643, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf, Fecha de Consulta: Junio 26, 2011

[60] XILINX, Inc. *Xilkernel*, documento UG646 (v5.00.a), Abril 19, 2010

[61] SEARCHENTERPRISELINUX, *POSIX (Portable Operating System Interface)*, http://searchenterpriselinux.techtarget.com/definition/POSIX, Fecha de Consulta: Junio 27, 2011

[62] *Semáforo (Informática)*, http://es.wikipedia.org/wiki/Sem%C3%A1foro_(inform%C3%A1tica), Fecha de Consulta: Junio 28, 2011

[63] *Message Queue*, http://en.wikipedia.org/wiki/Message_queue, Fecha de Consulta: Junio 28, 2011

[64] SEARCHCIO, *Shared Memory*, http://searchcio-midmarket.techtarget.com/definition/shared-memory, Fecha de Consulta: Junio 29, 2011

[65] SYMBIAN DEVELOPER LIBRARY, *Diference between Mutex and Semaphore*, http://geekswithblogs.net/shahed/archive/2006/06/09/81268.aspx, Fecha de Consulta: Junio 30, 2011

[66] *Drivers*, http://es.wikipedia.org/wiki/Drivers, Fecha de Consulta: Julio 4, 2011

[67] *Library*, http://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica), Fecha de Consulta: Julio 5, 2011

[68] XILINX, Inc., *Coorporative Information*, http://www.xilinx.com. Fecha de Consulta: Julio 10, 2011

[69] RODRIGUEZ, marina, *Diseño de un Sistema de Lectura y Procesado para Múltiples Sensores Embebidos en un FPGA*, http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20071220MarinaAparicio.pdf, 2007, Fecha de Consulta: Julio 11, 2011

[70] KATSUHIKO, ogata, *Ingeniería de Control Moderna*, 4ª edición, Prentice Hall, 2003.

[71] LA BARRA, *Hogar Digital*, http://www.labarradyr.com.ar/2006/abr06/La_barra_Empresas.htm, Fecha de Consulta: Julio 15, 2011

[72] NORBEY, jose, *Sistemas Distribuidos*, http://hstechelectronica.blogspot.com/2011/08/sistemas-distribuidos.html, Fecha de Consulta: Julio 15, 2011

[73] VEGA, josé, SANCHEZ, roberto, SALGADO, gerardo, SANCHEZ, luis,

Arquitectura RISC vs CISC, Universidad Autónoma Metropolitana - Unidad Atzcapotzalco, http://www.azc.uam.mx/publicaciones/enlinea2/num1/1-2.htm, octubre 2011, Fecha de Consulta: Julio 16, 2011

[74] GARCIA, almudeber, *Evolución de la Metodología de Diseño y la Tecnología*, http://upcommons.upc.edu/pfc/bitstream/2099.1/6277/5/3.EVOLUCI%C3%93N%20DE%20LA%20TECNOLOG%C3%8DA%20Y%20LA%20METODOLOG%C3%8DA%20DE%20DISE%C3%91O.pdf, 2011, Fecha de Consulta: Julio 16, 2011