UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERIA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA VIRTUAL PARA SISTEMAS EMBEBIDOS UTILIZANDO UN PROCESADOR SOFT-CORE EMBEBIDO EN UN FPGA

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:
INGENIERO ELECTRÓNICO

PRESENTADO POR
PEDRO PASCUAL ABERGA FARRO

PROMOCIÓN
2010- II

LIMA, PERÚ
2014

DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA VIRTUAL PARA SISTEMAS EMBEBIDOS UTILIZANDO UN PROCESADOR SOFT-CORE EMBEBIDO EN UN FPGA

DEDICATORIA:

Dedico este trabajo a Dios, mis padres, hermanos y amigos, quienes de una u otra manera me estuvieron apoyando para lograr culminar este trabajo. A Dios por haberme dado fuerzas y entusiasmo para poder empezar y acabar esta carrera, a mis padres por su apoyo incondicional en todo momento, a mis hermanos por sus palabras de ánimo, y a mis amigos que en realidad vendrían a ser todos, por los buenos momentos de compañerismo compartidos.

SUMARIO

En la presente tesis se realiza el estudio y diseño de un Sistema Digital, basado en un procesador embebido en un FPGA, que permite verificar a distancia el funcionamiento de otro dispositivo. El sistema diseñado debe ser capaz de enviar y recibir señales usando diferentes protocolos de comunicación, tales como: paralelo, vía convertidor A/D (para el ingreso de información analógica al sistema) y vía convertidor D/A (para el envío de datos digitales desde el sistema). Asimismo se puede enviar y recibir información del sistema, vía Ethernet.

Se hace uso de la metodología SoC (*System on Chip*), la cual es una creciente metodología de diseño en VLSI. Un tipo de SoC particular es el SoPC (*System on a Programmable Chip*), el cual es más flexible y apropiado para pequeños lotes de producción, ya que combina las ventajas del diseño SoC y PLD (Dispositivos de Lógica Programable)

Se implementa una interface de usuario en Java SE. A esta interface se puede ingresar vía acceso remoto para realizar el seguimiento a distancia, también se puede implementar una LAN para acceder al sistema desde diferentes usuarios mediante la interface de Java.

Una de las aplicaciones de este sistema es el ahorro en costos de equipamiento para dictar cursos que involucren dispositivos de alto costo, ya que éste dispositivo puede ser compartido, vía la interface de usuario, cuándo esté disponible la comunicación.

INDICE

Contenido

INDICE DE FIGURAS

INDICE DE TABLAS

GLOSARIO DE TÉRMINOS

FPGA	*:*	*Field Programmable Gate Array*
SOC	*:*	*System On Chip*
PLD	*:*	*Programmable Logic Device*
SOPC	*:*	*System on Chip Programmable*
VHDL	*:*	*“Very High Speed Circuit” Hardware Description Software.*
SPI	*:*	*Serial Periferical Interface*
EDA	*:*	*Electronic Design Automation*
CAD	*:*	*Computer Assistant Design*
MUX	*:*	*Multiplexor*
ADC	*:*	*Analog to Digital Converter*
DAC	*:*	*Digital to Analog Converter*
ASCII	*:*	*American Standard Code for Information Interchange*
Soft-Core	*:*	*Soft-Core Processor*
IP	*:*	*Intelectual Property*
IP-Core	*:*	*Intelectual Property Core*
LUT	*:*	*Look Up Table*

PRÓLOGO

El propósito general de la presente tesis es diseñar un “Laboratorio Virtual”, el cual nos permitirá ahorrar costos de equipamiento en el entrenamiento de diseño digital, así como también servirá como sistema de verificación a distancia de diferentes proyectos adaptables al módulo. La metodología seguida será la SoC que está basada en el diseño y desarrollo Hardware y Software, contando el sistema con un procesador Soft-Core embebido.

El sistema ha sido verificado para un sistema de red pequeña, ya que para hacer este sistema de mayores prestaciones se requiere una red más amplia en número, así como también contar con un servidor, el cual tendrá en ella el software instalado y así poder brindar mejores prestaciones. Pero actualmente el sistema puede sin ningún problema servir para el uso de un salón de clases.

El primer capítulo tiene por fin mostrar la problemática encontrada y que desea ser resuelta mediante la presente tesis, los objetivos y más puntos introductorios. El segundo capítulo muestra los aspectos teóricos requeridos para el desarrollo de la tesis. El tercer capítulo muestra de manera más concisa el problema y un esquema solución ya concepcionado. El cuarto capítulo ya muestra el análisis y diseño requerido para el desarrollo de la tesis. En el quinto capítulo ya se implementan los bloques que fueron analizados en el capítulo anterior. En el sexto capítulo se muestra la evaluación y pruebas del sistema diseñado. Finalmente se enumeran las conclusiones y recomendaciones para trabajos futuros.

CAPÍTULO I

CONCEPTOS GENERALES

1.1 Problemática

A lo largo de estos últimos años, se han implementado diferentes clases de sistemas remotos para múltiples ambientes de trabajo, partiendo desde el nivel doméstico en nuestros hogares, con los sistemas de control remoto, hasta el nivel industrial, como por ejemplo para adquisición y envío de datos a distancia para el control de una planta, en los cuáles se utilizan diferentes protocolos de comunicación, así como plataformas software y hardware.

Los sistemas de Monitoreo y Control remoto, son actualmente utilizados en múltiples aplicaciones, tales como:

- Sistemas de Medición y Predicción de parámetros climáticos.
- Sistemas de Seguridad Electrónica.
- Monitoreo y Control de Información.
- Monitoreo y Control de Procesos.
- Monitoreo y Control de Equipos, etc.

Estos sistemas son desarrollados a la medida de las necesidades, utilizando las últimas tecnologías de comunicación.

Bajo esa perspectiva, este trabajo está orientado a resolver la dificultad que existe para la programación y verificación (o monitoreo) de equipos (sistemas o dispositivos electrónicos) a distancia.

Para tal fin es requerido el desarrollo e implementación de lo siguiente:

- Plataforma Hardware.
- Interface de Usuario.
- Módulos de comunicación requeridos.

La implementación del Sistema(o Plataforma) Hardware está basada en la metodología SoC (*System on Chip*), la cual es una creciente metodología de diseño en VLSI. Un tipo de SoC particular es el SoPC (*System on a Programmable Chip*), el cual es más flexible y apropiado para pequeños lotes de producción, ya que combina las ventajas del diseño SoC y PLD (Dispositivos de Lógica Programable). [1]

El diseño de la Interface de Usuario es desarrollado en plataforma Java, en ella se implementa la interface gráfica, para esquematización de data enviada y recibida por el sistema, así como también se implementan los módulos de comunicación (mediante el uso de sockets) requeridos por el sistema.

Los módulos de comunicación requeridos se implementan tanto en el Sistema Hardware como en la Interface de Usuario, para así lograr un correcto entendimiento entre ambos sistemas.

Para entender claramente ésta problemática, se analizará el siguiente caso ilustrado:

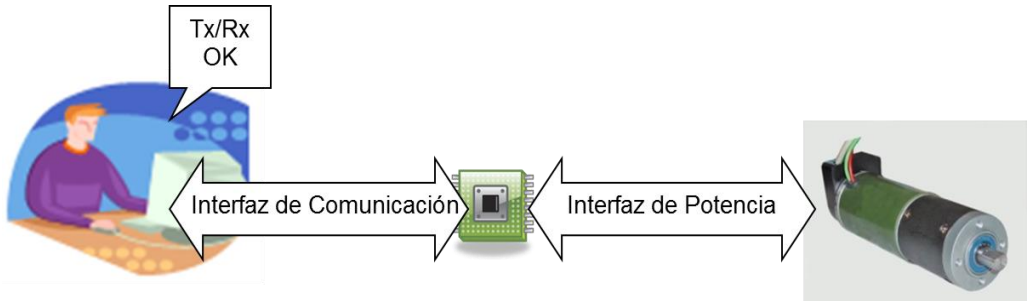

Figura 1. 1 Estudiante diseñando sistema de control de velocidad de motor DC.

Un estudiante desea diseñar un sistema de control de velocidad de un motor DC, el cual cuenta con un encoder.

Para resolver este problema el estudiante cuenta con opciones de dispositivos electrónicos programables tales como:

- Microcontroladores.
- DSP, etc.

El estudiante tendrá que diseñar una tarjeta electrónica para realizar la interfaz de potencia requerida. El algoritmo de control será descrito en el dispositivo electrónico programable elegido, el cual requiere leer señales que provengan del encoder, para poder conocer la velocidad del motor, asimismo deberá contar con señales de PWM para poder enviar a la tarjeta electrónica y así de esta manera poder cerrar el lazo del sistema.

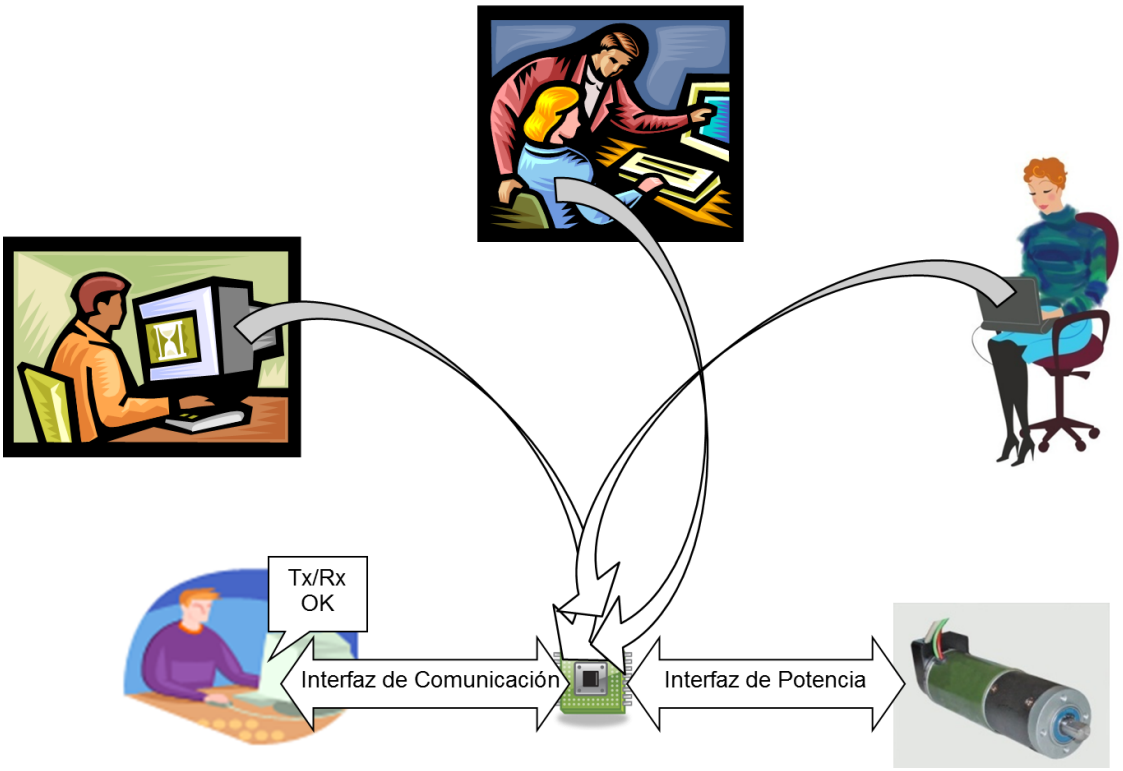

Figura 1. 2 Grupo de estudiantes que requieren hacer el mismo estudio.

Ahora imaginémonos que así cómo este estudiante existen otras personas, quiénes también desean diseñar un sistema de control similar, pero que lamentablemente no cuentan con el dispositivo electrónico digital que eligió este estudiante. Otra posibilidad es que cuentan de este dispositivo pero en un laboratorio al que sólo se puede acceder durante un horario poco accesible para ellos.

De acuerdo a esta problemática caben las siguientes preguntas:

¿Existe alguna posibilidad para que el dispositivo usado por el alumno pueda ser compartido de manera remota por otras personas que lo requieran?

Se cuenta con proyectos dónde sólo se puede programar o trabajar con cierto dispositivo a distancia, algunos hasta pueden ser monitoreados vía video con el uso de cámaras.

Pero sin embargo, no se ha encontrado evidencia de un sistema que haya sido diseñado para poder virtualizar cualquier tipo de sistema y sea manejado a distancia vía remota, es decir de la manera que se plantea, luego de mencionarse esto surgen otras interrogantes.

¿Qué características debería tener la solución electrónica que permita la programación a distancia de una plataforma digital tal como un DSP, FPGA, Microcontrolador, etc.?

- El sistema tiene que ser confiable, ya que se supone que el sistema será autónomo y no habrá nadie supervisando las 24 horas del día.
- El sistema tiene que tener bajo consumo de potencia y contar con modos de ahorro de energía, ya que habrá momentos en que el sistema no tendrá mucho requerimiento de uso.
- El sistema tiene que ser versátil a las reconfiguraciones a distancia, es decir totalmente reconfigurable de tal manera que pueda pensarse en más prestaciones para proyectos futuros.

¿Qué ventajas tendría diseñar este sistema con respecto a otros existentes?

- Sistemas actuales utilizan envío de data de verificación en video y es por ello que tienen un gran consumo de ancho de banda.
- El sistema tiene la gran ventaja de usar un reducido ancho de banda, el cual es controlado según el tamaño de la data que se quiera enviar; a comparación de sistemas que envían data del sistema del punto final vía data en video, lo cual consume un gran ancho de banda.

¿Qué tan confiable es el sistema?

- El sistema es altamente confiable, en medida del uso adecuado de cada componente físico en la comunicación. Se tendrá que respetar los protocolos de comunicación definidos para el usuario.
- La confiabilidad del sistema dependerá directamente de la confiabilidad de la comunicación punto a punto.
- Adicionalmente se podría usar cierta codificación para la protección contra errores.

¿Cuáles serían las aplicaciones de este sistema?

Se pueden presentar aplicaciones muy diversas tales como:

- Enseñanza de cursos a distancia, aún contando con recursos hardware limitado.
- Diseño de sistemas electrónicos con equipos de uso a distancia.
- Para mantenimiento y verificación de señales de equipos de difícil acceso.

- Para brindar facilidad de horario a alumnos que requieren entrenamiento en el uso de cierto dispositivo.
- Para ahorro de energía y tiempo en conexiones entre equipos a analizar, etc.

1.2 Objetivos Generales

- Diseñar un Sistema Hardware capaz de recibir (leer) data de un dispositivo (o sistema) electrónico, y que pueda enviar esta data vía Ethernet a una PC(o equipo destino).
- El sistema mediante el uso de una Interface de Usuario, debe ser capaz de enviar data a este mismo dispositivo vía Ethernet, es decir debe haber tanto envío como recepción de data.
- Diseñar una Interface de Usuario que nos permita visualizar la data recibida, así como también la data que deseamos enviar vía protocolo Ethernet.
- Programar y monitorear a distancia al dispositivo electrónico seleccionado para puesta en funcionamiento del sistema.

1.3 Objetivos Específicos

- Diseño y configuración del procesador Soft-Core Microblaze. Conexiones a IP cores de fabricante y adición de IP cores de usuario necesarias para el manejo de los diferentes periféricos.
- Manejo del dispositivo convertidor de Analógico a Digital brindado por la tarjeta de desarrollo Spartan 3E Starter Board para la recepción y control de las señales analógicas del dispositivo electrónico a distancia.
- Manejo del dispositivo convertidor de Digital a Analógico brindado por la tarjeta de desarrollo Spartan 3E Starter Board para el envío de señales analógicas al dispositivo electrónico a distancia.
- Manejo de la IP Ethernet Lite de Xilinx para poder establecer la comunicación vía Ethernet en el sistema.
- Diseño de Interfaces de usuario en Plataforma Java para poder visualizar la data del sistema.

1.4 Actualidad del Diseño de Sistemas basados en FPGA's

En la actualidad el ingeniero desarrollador cuenta con varias herramientas Software y Hardware para poder trabajar cada uno de sus nuevos diseños, con diferentes opciones de dispositivos hardware para realizar Diseño Electrónico Digital, tales como: FPGA's, CPLD's, DSP's, uCo, etc. Cada uno de los cuales deberá elegir según las circunstancias y rigurosidades del problema que se presente.

Además de los sistemas hardware con los que cuenta, en la actualidad se tiene que:

- La realidad del diseño lógico actual exige el uso de herramientas de diseño electrónico del tipo EDA.
- En cada herramienta EDA se tiene asociada una metodología de diseño.
- Herramientas EDA con soporte de dispositivos del fabricante o independiente del fabricante.
- Posibilidad de diseñar sistemas completos en chip programable (*SoPC = System on Programmable Chip*).

Los Sistemas embebidos están compuestos por hardware y software que trabajan juntos para realizar una función específica. Usualmente ellos contienen procesadores embebidos que frecuentemente están en la forma de procesadores Soft-Core que ejecutan código de software.

Para ver de una manera más general a los procesadores, enumeramos lo siguiente:

1. Los núcleos de procesadores se pueden implementar en hardware o software.

2. Procesadores en hardware (*Hard-Core*): Usan un núcleo de procesador embebido en silicio dedicado y ofrecen ventajas híbridas entre FPGA y ASIC.

3. Procesadores en software (*Soft-Core*): Usan elementos lógicos programables existentes en el FPGA para implementar la lógica del procesador. [2]

1.4.1 Proyectos relacionados a la Tesis

Los proyectos relacionados a este trabajo de tesis son fundamentalmente de diseño de sistemas digitales basados en procesadores Soft-Core.

- IP cores de Usuario para el manejo de diferentes dispositivos.
- Sistemas de comunicación Digital.
- Plataformas de comunicación Hardware y Software.
- Procesamiento Digital de Señales.

Un ejemplo de este tipo de aplicaciones se puede encontrar en [3], este proyecto diseña una interface que habilita a una tarjeta de desarrollo para FPGA's la comunicación con otros dispositivos vía conexión Ethernet, como se muestra en la Figura 1.3, siguiendo varios protocolos de redes establecidos. La tarjeta usada es la DE2-115 de Altera, la cual cuenta con FPGA Cyclone IV, así como también con dos chips Marvell 88ee1111 (*Ethernet PHY*) que permiten hacer las conexiones Ethernet gigabit.

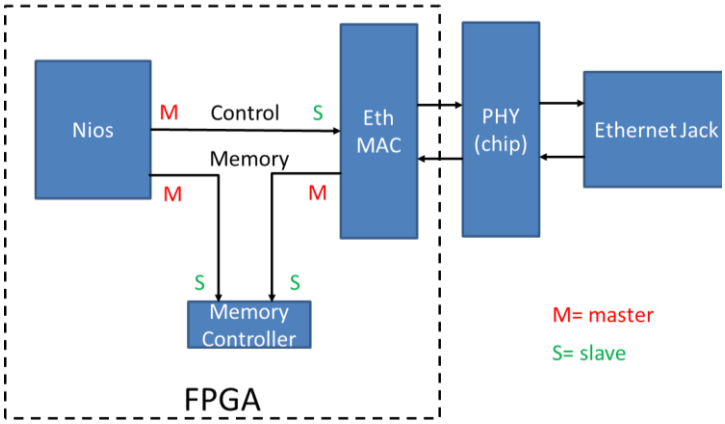

Figura 1. 3 Conexiones entre FPGA y el chip para uso del módulo Ethernet.

Otra aplicación que hace uso de comunicación Ethernet, la encontramos en [4]. Sistema digital basado en FPGA, el cual utiliza comunicación Ethernet y que hace uso del protocolo UDP en la capa de transporte, asimismo hace uso de la interface RMII (reduced media-independent interface) en la capa física. Diseña una interface de usuario en Windows para visualización de data.

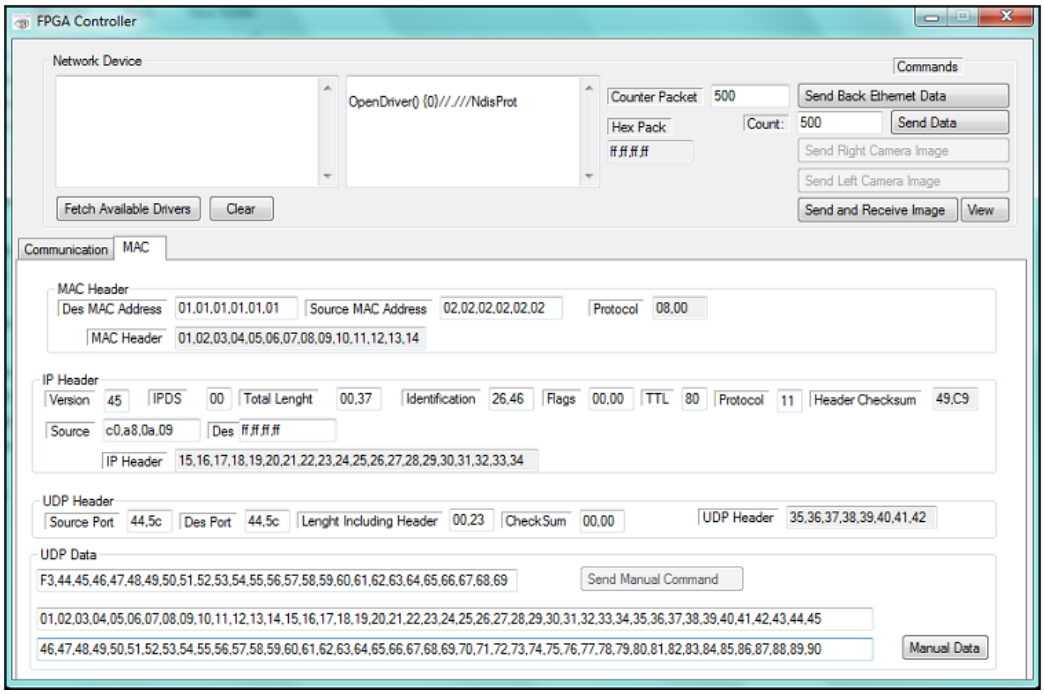

Figura 1. 4 Ventana de aplicación.

Un proyecto de aplicación de un sistema que hace uso de un servidor basado en comunicación digital basada en FPGA, lo encontramos en [5]. Sistema diseñado hace uso de un servidor basado en comunicación USB y utiliza cámaras web para verificación de data, este sistema es denominado Sistema RemoteFPGA, ver Figura 1.5.

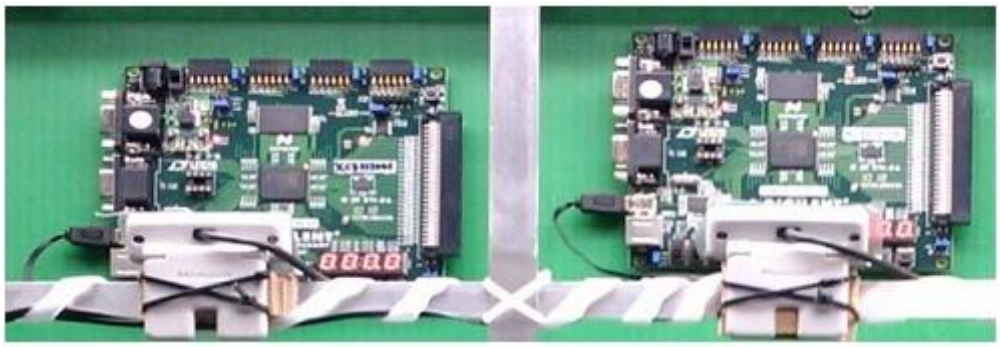

Figura 1. 5 Dos sistemas de desarrollo para FPGA y cámaras web (RemoteFPGA).

Una aplicación de comunicación a distancia, que realiza pruebas entre universidades de diferentes países puede verse en [6]. En este trabajo se presenta el monitoreo y control usando Internet2 desde un punto remoto ubicado en España, de un proceso localizado en el laboratorio de Control Automático de la Universidad de La Frontera.

Un proyecto interesante de procesamiento de voz puede verse en [7]. En este proyecto se utiliza: Codificador de Audio (*Audio Codec*) para adquisición de datos, Algoritmo LPC para compresión de datos, Modelo Oculto de Márkoc (HMM) para reconocimiento de patrones y un procesador Soft-Core Nios II para el procesamiento. El sistema reconoce las señales de voz y luego almacena lo reconocido en forma de texto, tal como se muestra en el diagrama de la Figura 1.6.

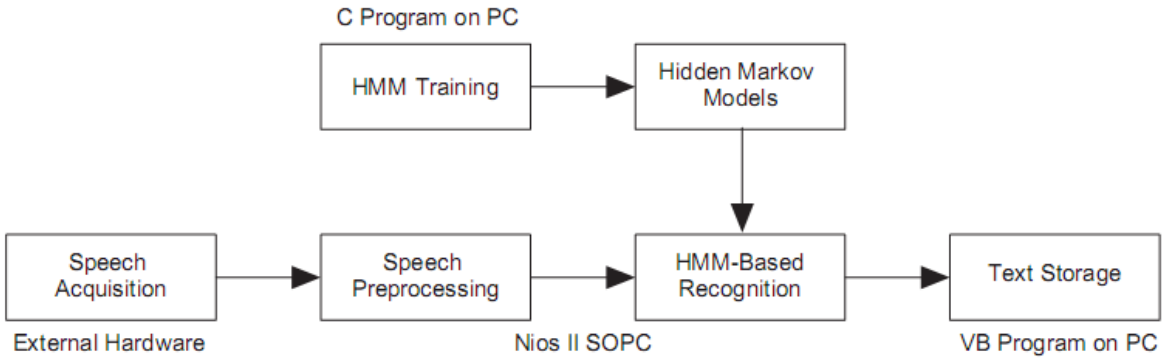

Figura 1. 6 Diagrama de bloques funcional.

En la adquisición de las señales de voz se realiza un preprocesamiento, el diagrama de la Figura 1.7 describe esta etapa.

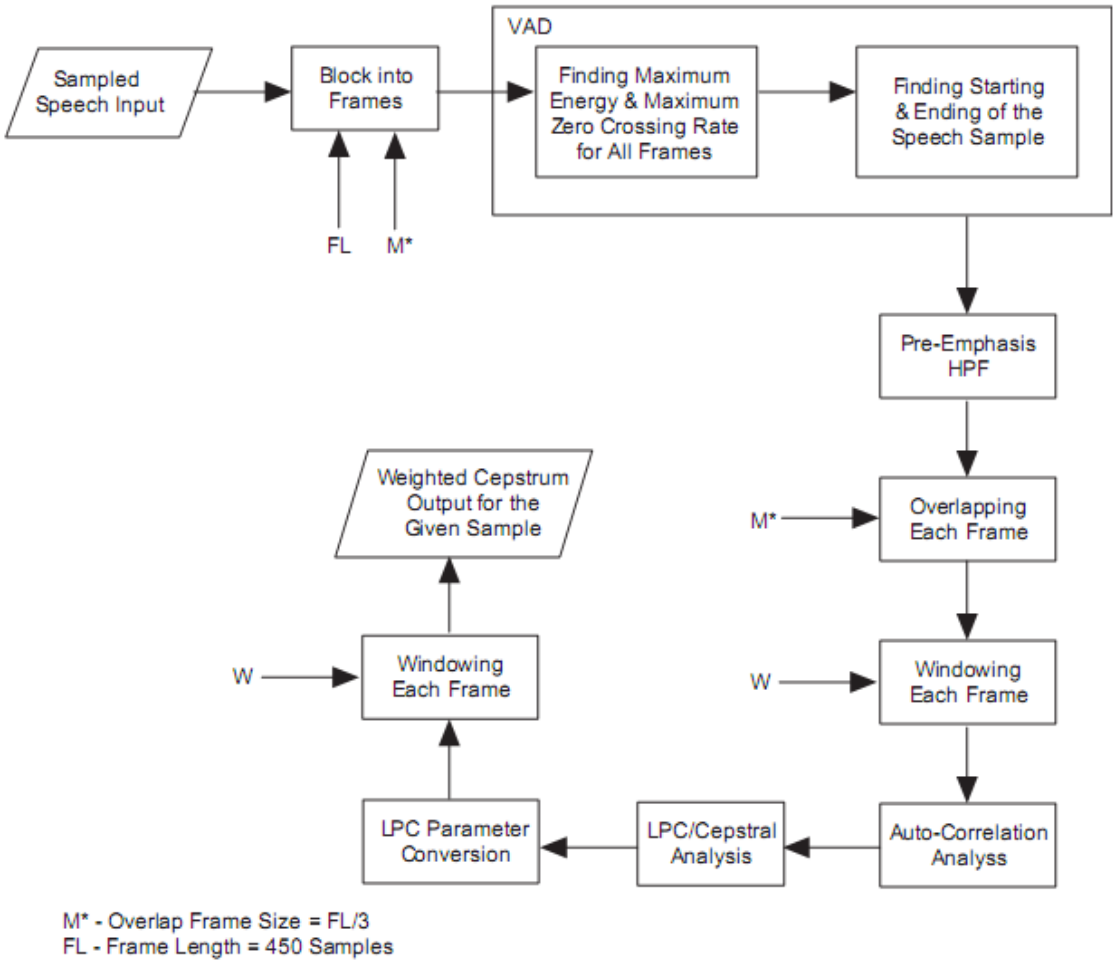

Figura 1. 7 Diagrama de bloques de Preprocesamiento.

Por último, una aplicación interesante a mencionar y detallar brevemente es la aplicación *Hardware in The Loop Simulation* (*HIL Simulation*). Experimentos con una configuración mecatrónica básica muestran que el sistema HIL simulado se comporta como un sistema real, y puede ser considerado una adición útil en Diseño de Sistemas. Esto puede disminuir el tiempo de desarrollo considerablemente, especialmente cuándo el producto mecatrónico a ser diseñado es complejo.

Hardware in the Loop Simulation [8]

Hardware in the Loop Simulation se utiliza como herramienta de ayuda en el diseño y testeo de sistemas complejos, en ingeniería de diferentes disciplinas, en el presente trabajo pueden mencionarse a los sistemas electrónicos y mecatrónicos dónde las condiciones de prueba o de uso están fuera de parámetros estándares, o de condiciones ambientales normales.

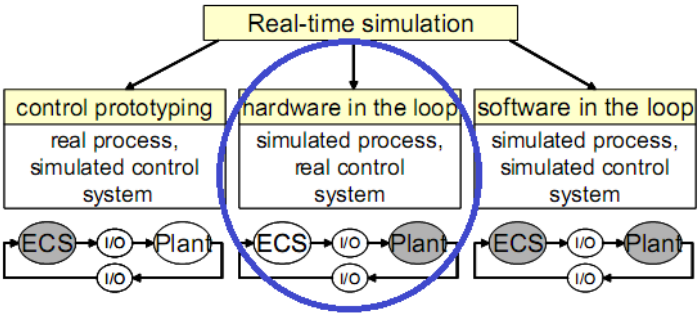

Figura 1. 8 Simulación en Tiempo Real

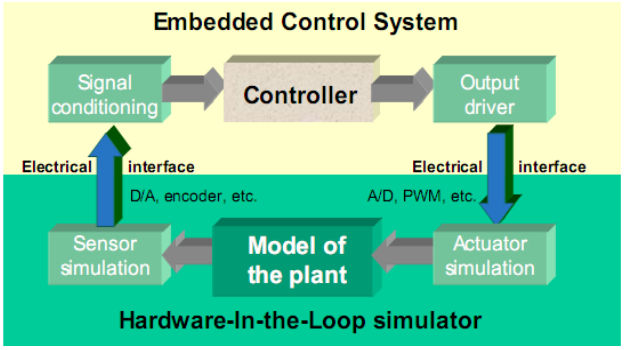

Figura 1. 9 Sistema de Control Embebido

La Figura 1.9 muestra que este sistema requiere módulos A/D, D/A, y de otras interfaces para su implementación.

De manera aplicativa podemos mencionar el siguiente sistema: Imaginemos que deseamos diseñar y controlar un péndulo invertido de manera virtual, tal como se muestra en la Figura 1.10. Entonces para hacer uso de este tipo de simulación se deberá ingresar de manera virtual el modelo del péndulo y realizar el control virtualmente, vía comunicación FPGA y PC. Luego de controlar virtualmente y realizar una correcta simulación usando esta metodología, podremos tener una referencia para poder controlar un sistema real con estas mismas características, con los ajustes requeridos.

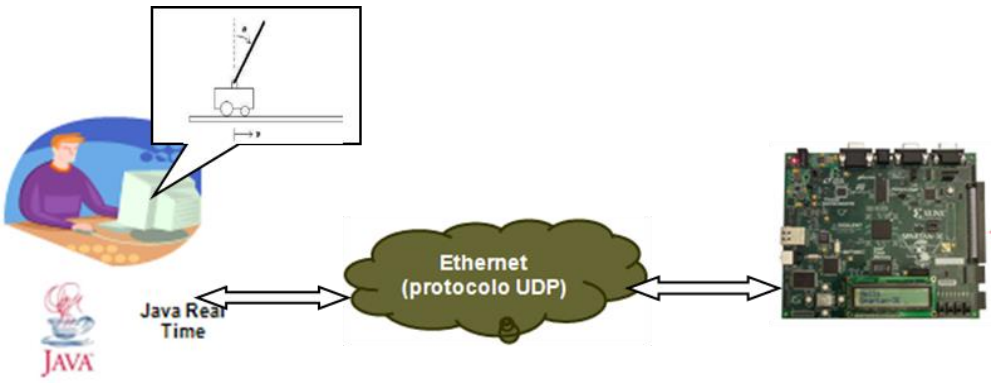

Figura 1. 10 Diseño Aplicativo de Sistema con péndulo invertido virtual.

1.5 Organización del sistema

El sistema tiene por objetivo lograr la verificación de un sistema digital (o que pueda digitalizarse), el cual se encuentra a distancia. Un esquema del sistema puede ser representado según se muestra en la siguiente figura:

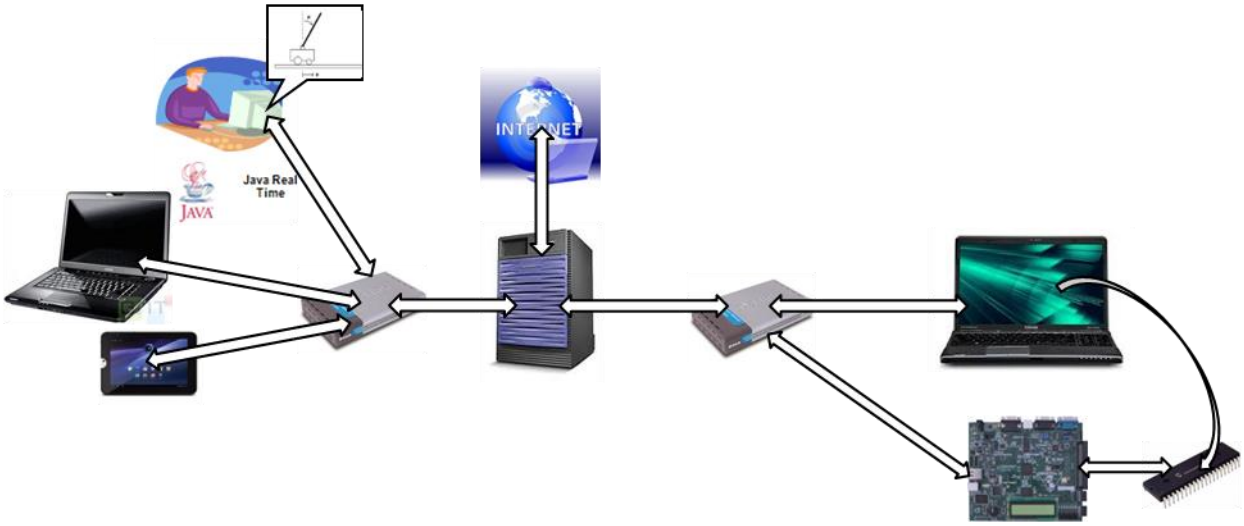

Figura 1. 11 Plataforma para un ambiente de aplicación complejo.

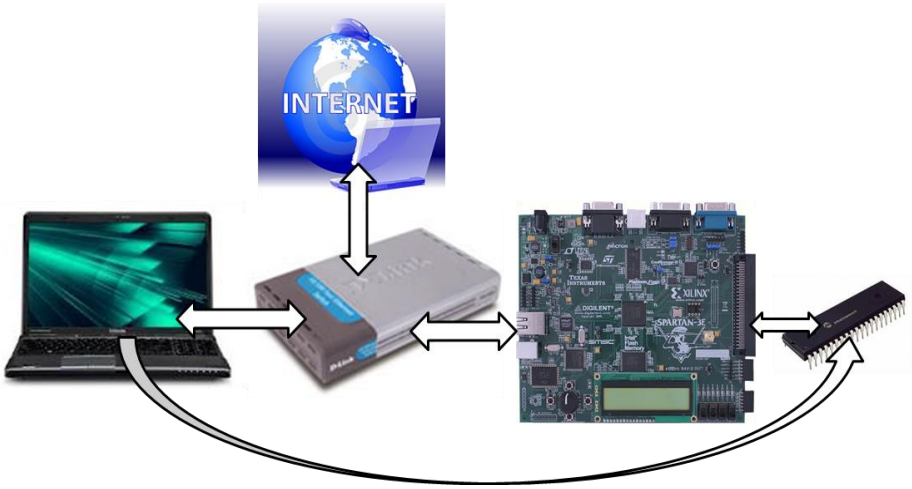

Figura 1. 12 Representación de la plataforma para una aplicación sencilla.

En la Figura 1.12 se muestran todos los componentes que conforman el sistema requerido para este proyecto. Esta representación es para un sistema no tan complejo como el que se muestra en la Figura 1.11, sin embargo nos permitirá observar las cualidades fundamentales del sistema.

CAPÍTULO II

MARCO TEÓRICO

2.1 Arquitectura de un FPGA

Los FPGA's (*Field Programmable Gate Arrays*) son dispositivos semiconductores programables que están basados por una matriz de bloques lógicos configurables (CLB's) conectados vía interconexiones programables. A diferencia de los Circuitos Integrados de Aplicación Específica (ASIC's) donde el dispositivo es configurado para el diseño en particular, los FPGA's pueden ser configurados para una aplicación deseada o según los requerimientos de funcionalidad. [9]

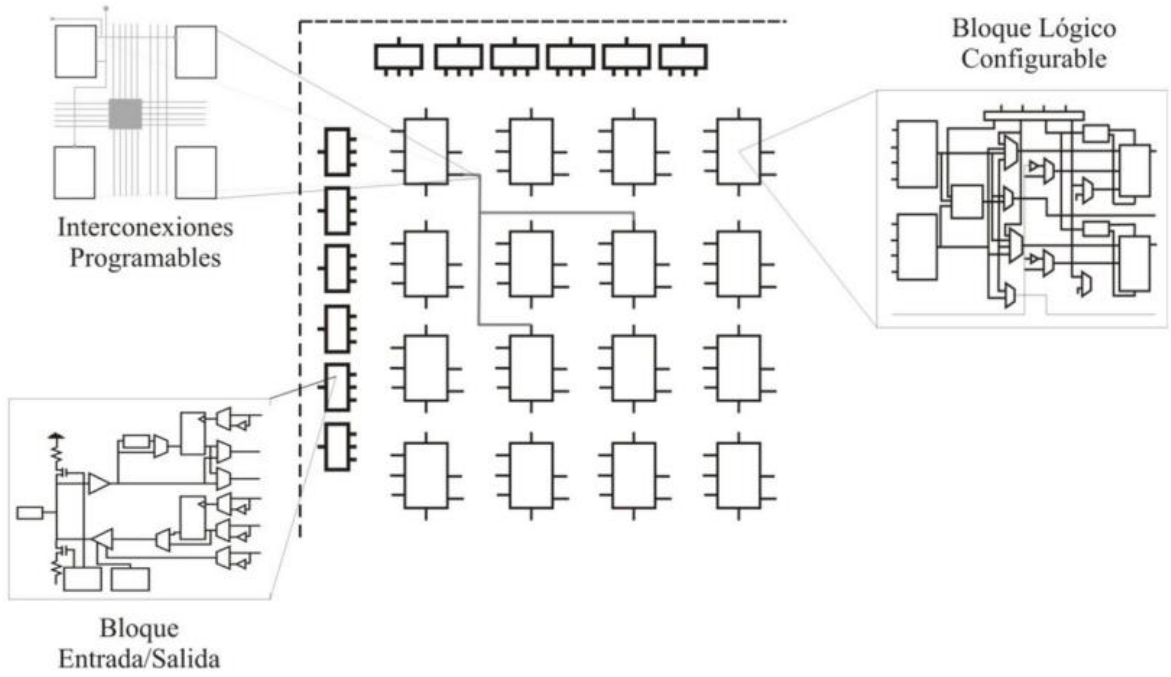

Figura 2. 1 Elementos básicos en la arquitectura de un FPGA. [10]

En la Figura 2.1 se muestra la arquitectura básica de un FPGA, la cual se encuentra formada por tres elementos básicos:

- CLBs (Bloques de Lógica Configurable), proporcionan los elementos funcionales para construir la mayoría de la lógica así como el almacenamiento de datos. Su complejidad puede variar desde un par de transistores hasta un conjunto de memorias de acceso aleatorio denominadas tablas de consulta (LUT, *Look-Up-Tables*).

- IOBs (Bloques de Entrada/Salida), estos recursos lógicos proveen el control de flujo de datos entre lógica interna y las terminales de entrada/salida del dispositivo.
- Recursos de interconexión, las entradas y salidas de los CLB e IOB se interconectan mediante líneas e interruptores programables que se ubican en los canales de cableado entre las filas y columnas de los bloques.

Los FPGA's son particularmente adecuados para la implementación de algoritmos paralelos. Sin embargo, los algoritmos secuenciales, especialmente aquellos que no exigen gran capacidad de procesamiento, son más fáciles de implementar como un programa para un microprocesador, o mediante el uso de procesos en lenguaje de descripción de hardware.

Desde su primer diseño en el año 1985, por la empresa Xilinx, los FPGA's han venido evolucionando a escalas predichas por la Ley de Moore, y en la actualidad se cuenta con módulos de desarrollo muy complejos de diferentes compañías. Los cuales son usados para realizar prototipos de diferentes sistemas electrónicos.

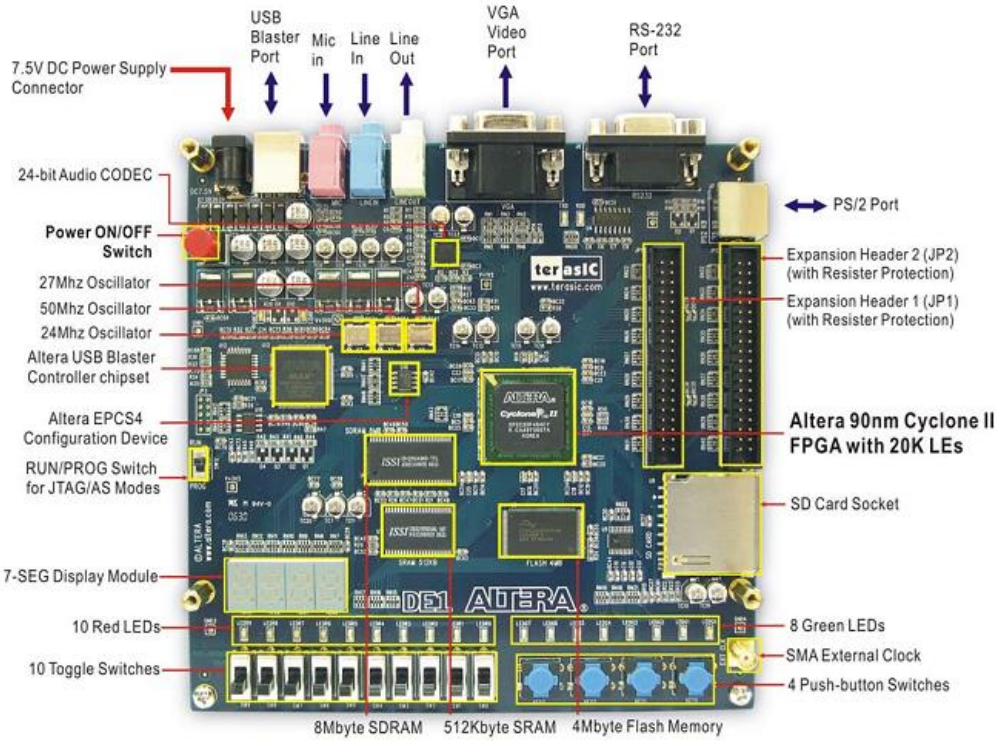

Figura 2. 2 Tarjeta de desarrollo DE1 de Terasic-Altera. [11]

En la Figura 2.2 se muestra una tarjeta de desarrollo que cuenta con el FPGA de la Figura 2.3. Así como esta tarjeta, también se cuenta con otras herramientas para el desarrollo basado en FPGA's, las cuales serán mostradas posteriormente.

Figura 2. 3 FPGA Cyclone II EP2C20F484C7N de Altera

Algunos de los componentes de esta tarjeta de desarrollo son:

- Puerto Serie, Puerto VGA, Puerto PS/2, Memorias.
- Entrada y Salida de Audio.
- Switches, Leds y Displays.

2.2 Lenguaje de Descripción de Hardware

Un HDL (*Hardware Description Language*) o Lenguaje de Descripción de Hardware, como es conocido en el idioma español, es un lenguaje estandarizado que nos permite describir circuitos electrónicos. Es decir, mediante este tipo de lenguaje nos es posible describir el comportamiento, luego mediante el uso de herramientas adecuadas se pueden realizar simulaciones y pruebas, para una posterior implementación en plataformas hardware de verificación y finalmente para una producción a mayor escala, según propósitos del diseñador.

Cada uno de estos lenguajes cuenta con una estructura de código y ciertas reglas, esto es debido a que este tipo de diseño es del tipo *Semi-Custom*. Entre los HDL más conocidos se tiene al VHDL y Verilog, el uso del VHDL, conocido así por sus siglas VHSIC (*Very-High-Speed Integrated Circuits*) y HDL, se encuentra más difundido en Europa, mientras que el Verilog está más difundido en Estados Unidos. Es importante hacer mención que el lenguaje Verilog también es usado para un nivel de diseño más bajo, que es al nivel de transistor y es usado en Software preparado para el diseño en microelectrónica.

Para el desarrollo basado en HDL se utilizan herramientas del tipo CAD y EDA, existen diferentes fabricantes de hardware, contando algunos de ellos con su propio HDL para diseño bajo herramientas que ofrecen.

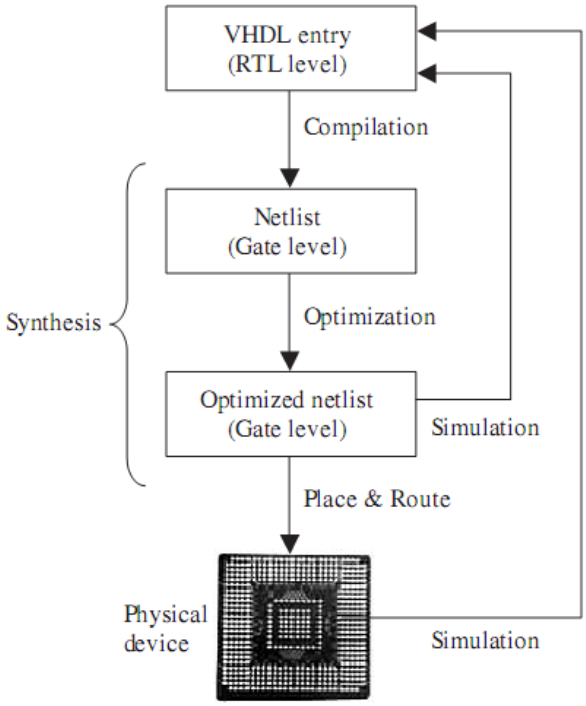

Figura 2. 4 Resumen del Flujo de Diseño VHDL. [12]

En la Figura 2.4 se indica en resumen el flujo de diseño que se realiza al usar el Lenguaje de Descripción de Hardware VHDL, sin embargo mediante el uso de otros lenguajes se cuenta con un flujo similar. Para realizar este flujo de diseño es requerido hacer uso de las herramientas de diseño, tales como las Herramientas CAD y EDA.

Herramientas CAD y EDA

El concepto de CAD (diseño asistido por ordenador, *Computer Aided Design*) significa proceso de diseño que emplea sofisticadas técnicas gráficas por ordenador, apoyadas por paquetes de software para la resolución de problemas de cálculo, de desarrollo, costes, etc. asociados con el trabajo de diseño.

El impacto de las herramientas de CAD sobre el proceso de diseño de circuitos electrónicos y sistemas procesadores es fundamental. No sólo por la adición de interfaces gráficas para facilitar la descripción de esquemas, sino por la inclusión de herramientas como simuladores, que facilitan el proceso de diseño y la verificación de las unidades de diseño.

EDA (*Electronic Design Automation*) es el nombre que se le da a todas las herramientas (tanto software como hardware) que sirven para el diseño de sistemas electrónicos.

El diseño hardware tiene un problema que no existe en la producción software. Este problema es el alto coste de prototipación-testeado-vuelta a empezar, ya que el coste del prototipo suele ser bastante elevado.

El enfoque de las herramientas de diseño ha ido encauzado a la inclusión de la etapa de prototipado solamente al final de las etapas de diseño, evitando la fabricación de varios prototipos a lo largo del proceso de diseño. Para ello se introduce la fase de simulación y comprobación de circuitos utilizando herramientas CAD, de forma que no sea necesaria la realización física del prototipo para la comprobación del funcionamiento del circuito.

A continuación para poder mostrar una ilustración de las herramientas de diseño, enumeraremos las herramientas usadas para el desarrollo de aplicaciones con FPGA's usando FPGA's de Altera y Xilinx. El trabajo con FPGA's de otras marcas es similar, ya que muchas de las herramientas software usadas están hechas por los mismos desarrolladores de software.

Sistema con FPGA de Xilinx

Herramientas Software:

- Para el desarrollo de HDL, Esquemáticos u otro archivo de entrada de diseño de hardware es utilizada la herramienta ISE (*Integrated Software Environment*) de Xilinx.
- Para el diseño del hardware del procesador Soft-Core Microblaze la herramienta usada es el XPS (*Xilinx Platform Studio*).
- El desarrollo del software del procesador Microblaze es desarrollado en el SDK (*Software Development Kit*) aunque también es posible trabajar el software del procesador en XPS, tal vez en el futuro sea quitada esta posibilidad ya que así es mencionada en documentación de Xilinx, con el fin de enfocar a SDK para ese objetivo.
- Adicionalmente también podemos mencionar que Xilinx cuenta con *DSP Tools* en donde tenemos a la herramienta *Xilinx Systems Generator*, esta herramienta trabaja conjuntamente con Simulink de Matlab, lo cual permite trabajar bloques en Simulink que generarán código en HDL.

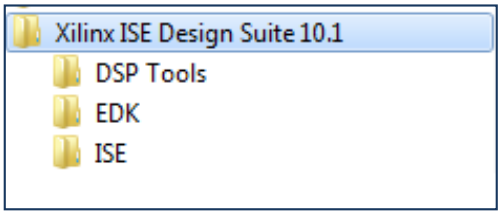

Figura 2. 5 Organización de herramientas CAD y EDA de Xilinx.

Según la organización que se muestra en la Figura 2.5, tenemos que XPS y SDK conforman el EDK (*Embedded Development Kit*). Las cuáles son las herramientas más importantes cuándo se trata de trabajar con el procesador Microblaze. En la Figura 2.6 se muestra la página de inicio del XPS, la cual indica el flujo de diseño.

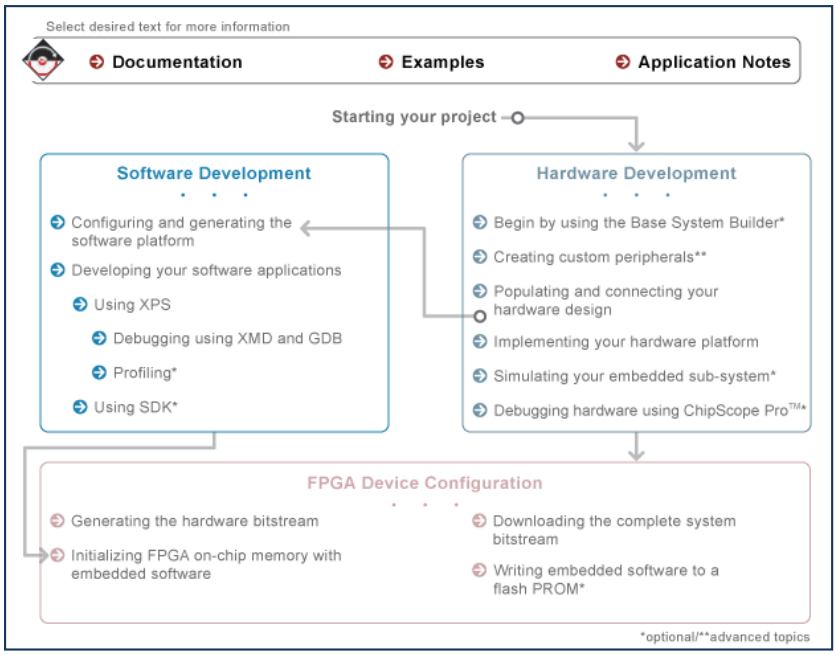

Figura 2. 6 Flujo de diseño en Xilinx Platform Studio (XPS).

Sistema con FPGA de Altera

Herramientas Software:

- Para el desarrollo de HDL, Esquemáticos u otro archivo de entrada de diseño de hardware será usado la herramienta Quartus II de Altera.

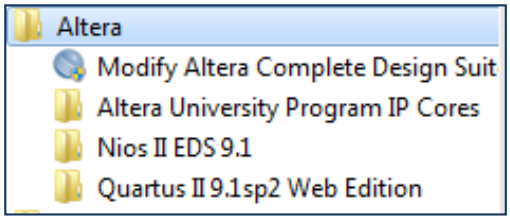

Figura 2. 7 Organización de herramientas CAD y EDA de Altera.

- Para el diseño del hardware del procesador Soft-Core Nios II usaremos la herramienta SOPC Builder, la cual es una herramienta interna del Quartus II.
- Para poder trabajar con SOPC Builder primero debe crearse un proyecto en Quartus II, se debe tener siempre presente que **no es recomendable almacenar proyectos en direcciones que cuenten con espacios y tildes** ya que es muy probable tener problemas debido a ello en la compilación.

- Para el desarrollo del software del procesador Nios II se utiliza el Nios IDE (*Integrated Development Environment*).

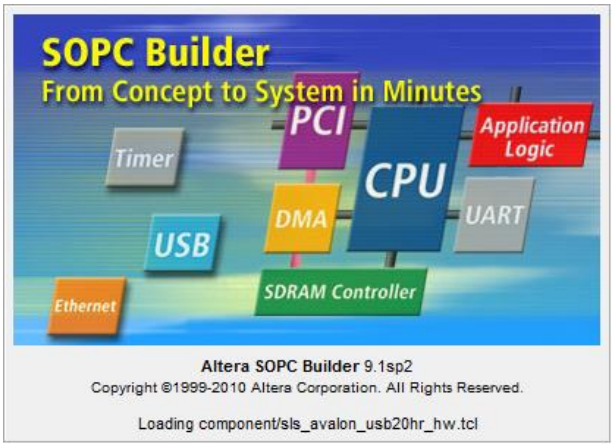

Figura 2. 8 Lanzando SOPC Builder desde Quartus II.

2.3 Procesadores Embebidos (*Soft-Core*) en FPGA's

Un procesador *Soft-Core* es un modelo de lenguaje de descripción de hardware (HDL) de un procesador específico (CPU), que puede ser personalizado o diseñado para una aplicación dada y sintetizado para un ASIC o FPGA objetivo. En muchas aplicaciones, procesadores Soft-Core proveen varías ventajas sobre los procesadores que son diseñados a medida, tales como costo reducido, flexibilidad, independencia de plataformas y gran inmunidad a quedar obsoletos.

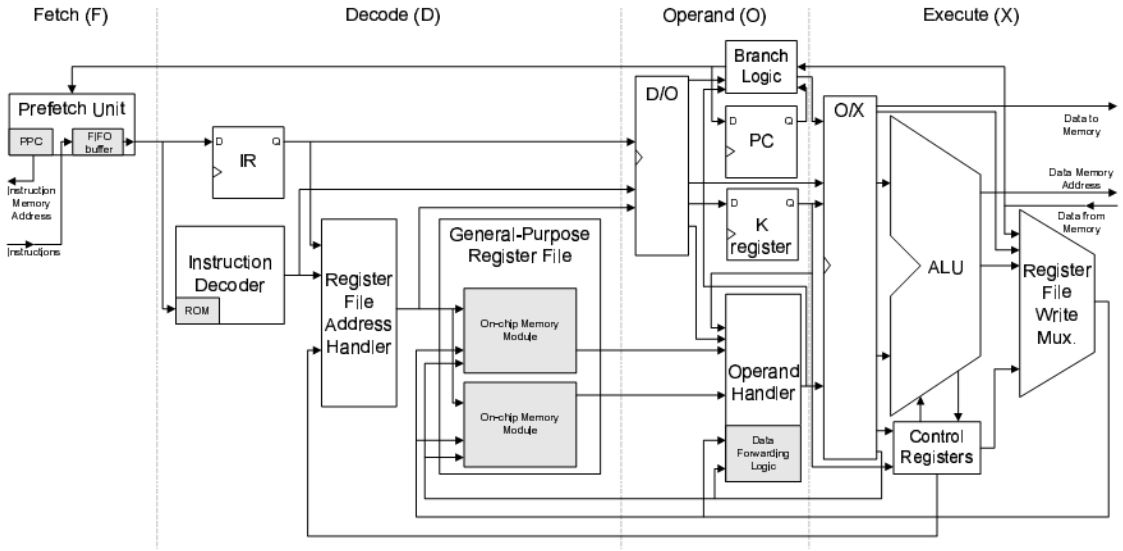

Figura 2. 9 UT Nios *datapath* [13]

La Figura 2.9 muestra el modelo de 4 estados (*Fetch, Decode, Operand, Execute*) en un procesador Nios mediante un esquema *datapath* (camino de datos).

2.3.1 Características de procesadores disponibles

A continuación mostramos 2 tablas comparativas de las características de algunos procesadores Soft-Core disponibles.

Tabla 2. 1 CPUsSoft-Core embebidos para FPGA [14]

CPU core	Architecture	Bits	Pipeline depth	Cycles per instruction[1]	Area (LEs[2])
S1 Core	SPARC-v9	64	6	1	37000 - 60000
LEON3	SPARC-v8	32	7	1	3500
LEON2	SPARC-v8	32	5	1	5000
OpenRISC 1200	OpenRISC 1000	32	5	1	6000
MicroBlaze	MicroBlaze	32	3, 5	1	1324
aeMB	MicroBlaze	32	3	1	2536
OpenFire	MicroBlaze	32	3	1	1928
Nios II/f	Nios II	32	6	1	1800
Nios II/s	Nios II	32	5	1	1170
Nios II/e	Nios II	32	no	6	390
LatticeMico32	LatticeMico32	32	6	1	1984
Cortex-M1	ARMv6	32	3	1	2600
DSPuva16	DSPuva16	16	no	4	510
PicoBlaze	PicoBlaze	8	no	2	192
PacoBlaze	PicoBlaze	8	no	2	204
LatticeMico8	LatticeMico8	8	no	2	200

1 El valor especificado puede ser válido para la mayoría de las instrucciones, pero no para todas. Por ejemplo, la multiplicación a menudo necesita más que un ciclo normal de instrucciones de la ALU. **2** El área se mide en los Elementos Lógicos (LE's), que consisten de un LUT de 4 entradas y un flip-flop. Las estimaciones de superficie son únicamente para fines de referencia.

Tabla 2. 2 Comparativa entre procesador Soft-Core Nios II y Microblaze [1]

Feature	Nios II 5.0	Microblaze 4.0
Datapath	32 bits	32 bits
Pipeline Stages	1 - 6	3
Frequency	Up to 200 MHz	Up to 200 MHz
GateCount	26,000 – 72,000	30,000 – 60, 000
Register File	32**g**eneral **p**urpose& 6 special purpose	32**g**eneral **p**urpose& 6 special purpose
Instruction Word	32 bits	32 bits
Instruction Cache	Optional	Optional
Hardware Multiply& Divide	Optional	Optional
Hardware Floating Point	ThirdParty	Optional

Procesador MicroBlaze:

El procesador Soft-Core MicroBlaze de 32 bits está basado en una arquitectura (o motor) RISC, con instrucciones específicas para datos y acceso a la memoria. Es compatible tanto con el chip BlockRAM y / o memoria externa. Todos los periféricos se aplican sobre la estructura del FPGA.

A continuación mostraremos un diagrama de bloques de este procesador que es desarrollado por Xilinx.

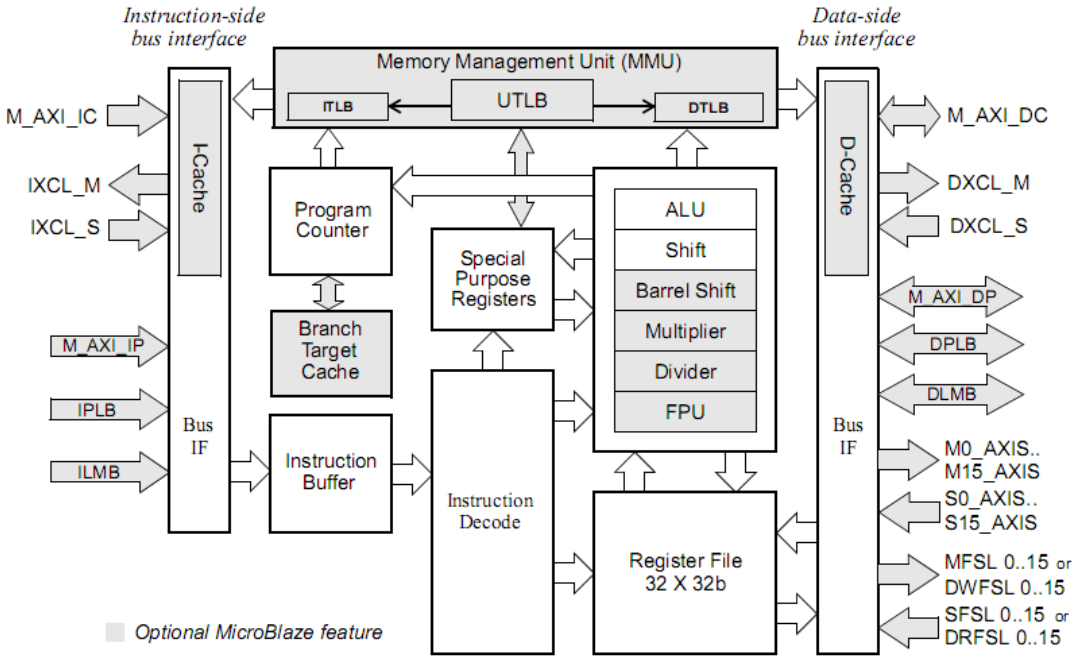

Figura 2. 10 Diagrama de bloques de un procesador Microblaze. [15]

Procesador Nios II:

Nios II es un procesador de propósito general de 32 bits, de arquitectura RISC, desarrollado por Altera para uso en sus FPGA's. Es el sucesor de los antiguos procesadores Nios embebidos de 16 bits.

Nios II Soft-Core puede ser customizado como sigue:

1. Nios II/f – Optimizado para el máximo rendimiento, estos emplean pipeline de 6 estados, ejecutan 1 instrucción por cada ciclo de instrucción e incluye por separado cache de instrucciones y cache de datos, MMU (*Memory Management Unit*) y MPU (*Memory Protection Unit*), hardware multiplicador, divisor y operaciones de desplazamiento.

2. Nios II/s – Diseñado como un trade-off (de negociación), emplea pipeline de 5 estados, ejecuta una instrucción por ciclo de instrucción e incluye una cache de instrucciones (no incluye cache de datos) y hardware multiplicador, divisor y operaciones de desplazamiento (MMU y MPU están ausentes).

3. Nios II/e – Diseñado para ocupar una pequeña área (LE's), no usan pipeline, ejecutan una instrucción cada 6 ciclos y no cuenta con bloques aritmético como en los 2 anteriores casos.

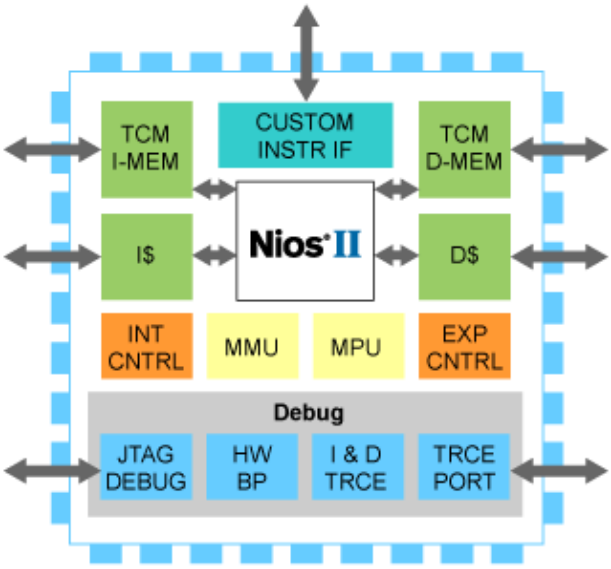

Figura 2. 11 Interconexiones con Procesador Nios II de Altera. [16]

Procesador Cortex-M1:

Cortex-M1 es una implementación de la arquitectura ARMv6 propietaria de 32 bits diseñado para FPGA's. Utilizar Cortex-M1 requiere licencia de *ARM Limited*. Hay chips FPGA's Actel basados en memoria flash que cuentan con Cortex-M1 de licencia incluida (no requiere licencia). Cortex-M1 también puede ser utilizado con FPGA's Xilinx y Altera.

ARM (*Advanced RISC Machine*) es una popular arquitectura RISC especialmente adecuada para aplicaciones que requieren bajo consumo de energía. Muchos sistemas operativos han sido portados a ARM, incluyendo Linux y Windows CE.

2.3.2 Ventajas y desventajas de un Sistema basado en Soft-core

a) Ventajas

- Característica de Diseño Hardware-Software.
- Reconfigurables, así evitan el quedar obsoletos con el tiempo.
- Hardware adicional puede ser creado por el diseñador.
- Software de Procesador puede ser actualizado sin necesidad de modificar Hardware.
- Procesador permite abstracción de diseño.

- Actualmente gran desarrollo en esta metodología de diseño.
- Los sistemas basados en hardware programable son por lo general totalmente versátiles ya que el usuario es quien define la lógica del sistema.

b) Desventajas:

- Estilo de diseño es no completamente libre, ya que procesador define ciertas características a respetar.
- Procesador puede llegar a consumir gran cantidad de recursos, dependiendo de características con las que cuente.

2.4 Núcleos de Propiedad Intelectual o IP Cores

En diseño electrónico una IP Core o bloque de IP es una unidad reutilizable de lógica. Las IP Cores se pueden utilizar como bloques de construcción dentro de los diseños de chips ASIC o FPGA's. Como elementos esenciales de la reutilización de diseños, las IP Cores son parte de la creciente automatización de diseño electrónico (EDA) tendencia de la industria hacia el uso repetido de los componentes previamente diseñados. Idealmente, un núcleo IP debe ser totalmente portátil, es decir, capaz de ser fácilmente insertado en cualquier tecnología o proveedor de la metodología de diseño.

Características de IP's disponibles

- IP cores de Fabricantes

En el caso de el fabricante Altera se cuenta con *Altera University Program IP Cores*, es decir se cuenta con un paquete de un programa universitario, el cual al ser usado con el software que se obtiene libremente desde la página de este fabricante nos permitirá grabar nuestra tarjeta de desarrollo pero con la única limitación de ser una ejecución de tiempo limitado.

Para el caso del fabricante Xilinx en la versión 10.1 se cuenta con la disponibilidad de algunas de las IP, siendo estas tomadas de la IP Catalog que la herramienta EDK (Kit de desarrollo embebido) de Xilinx nos brinda.

- IP cores libres

Las IP cores libres, o conocidas generalmente como "open cores", presentan la característica de ser de código libre y son normalmente independientes de la plataforma de desarrollo, esta es una gran ventaja con respecto a las IP cores de los fabricantes, sin embargo al ser independientes de la plataforma de desarrollo para poder ser usadas en cada una de ellas puede tomar tiempo adicional para adaptarlas a cada una de ellas.

OpenRISC 1200

Opencores [17] es una comunidad de hardware de código abierto que ofrece una gran variedad de IP's entre los que se encuentra su producto estrella: el OpenRISC 1200, un microprocesador RISC de 32 bits que se distribuye como código abierto.

Algunas características de este Soft-Core *Processor* son:

- Arquitectura Harvard.
- Pipeline de 5 etapas.
- Unidad de manejo de memoria (MMU).
- Cachés de instrucciones y de datos de 8 KB, de mapeo directo.
- Unidades opcionales, tales como: unidad de depuración, *tick-timer*, controlador de interrupciones y unidad de manejo de potencia (PMU).
- Interfaz WISHBONE para los buses de instrucciones y de datos.

Estudio de IP's de Xilinx a usar

A continuación se indican algunas de las IP cores de Xilinx requeridas.

IP cores necesarias:

- **Emac lite Ethernet:** Para poder trabajar con el módulo de comunicación Ethernet brindado por la tarjeta de desarrollo.
- **Serial:** Para poder realizar la comunicación serial mediante la plataforma de trabajo.
- **Gpio:** Para poder trabajar con entradas y salidas paralelas de propósito general.

2.5 Redes de Transmisión de Datos

Una red de comunicación trata de resolver diferentes tipos de problemas físicos y técnicos de un sistema de comunicación dividiéndolo en diferentes niveles de abstracción, llamados capas y así poder abordar cada problema por separado.

Cuando se construyeron las primeras computadoras durante la segunda guerra mundial, eran costosas y se encontraban aisladas [18]. Sin embargo, después de unos veinte años, ya que sus precios disminuyeron gradualmente, los primeros experimentos comenzaron a conectar los ordenadores juntos. A principios del decenio de 1960, los investigadores como Paul Baran, Donald Davies y Joseph Licklider publicaron independiente los primeros artículos que describen la idea de construir las redes de ordenadores. Dado el costo de las computadoras, compartirlos a larga distancia era una idea interesante. En los EE.UU., la ARPANET (*The Advanced Research Project Agency*) comenzó en 1969 y continuó hasta mediados de la década de 1980 [19].

En Francia, 1972, Louis Pouzin desarrolló la red informática bautizada como CYCLADES [20]. Muchas otras redes de investigación fueron construidas durante la década de 1970 [21].

En la Figura 2.12, que proporciona la estimación del número de ordenadores conectados a Internet, se nota que Internet ha sufrido un gran crecimiento a lo largo de los últimos 20 años.

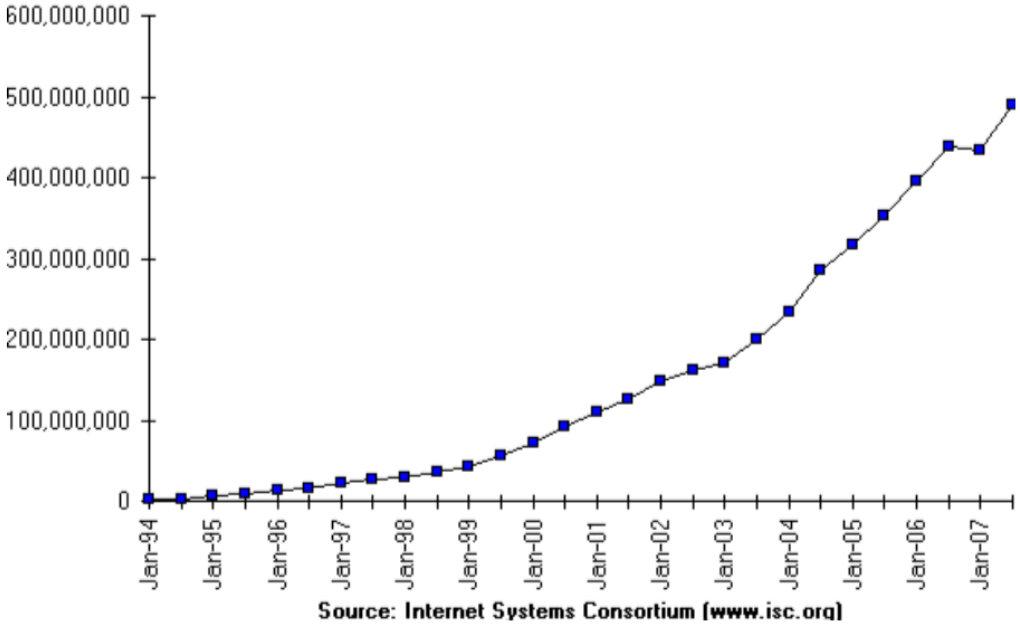

Figura 2. 12 Estimación del número de *host* en la Internet. [18]

Modelos de Redes [22]

Los modelos de red definen un conjunto de capas de red y cómo interactúan. Existen modelos de red diferente, dependientes de la organización o compañía que los crearon.

Los dos más importantes son:

- **Modelo de Red OSI:** La Organización Internacional de Normalización (ISO) ha definido un estándar llamado el modelo de referencia de interconexión de sistemas abiertos (*OSI reference model*). Esta es una arquitectura de 7 capas, tal como se muestra en la Figura 2.13.
- **El modelo TCP/IP:** Este modelo es a veces llamado el modelo DOD (del Departamento de Defensa) ya que fue diseñado para el departamento de defensa de los Estados Unidos. También se conoce como el modelo de Internet, dado que TCP/IP es el protocolo utilizado en Internet. Este modelo consta de 5 capas, tal como se muestra en la Figura 2.14. Los modelos más antiguos a menudo muestran sólo cuatro capas, que combinan las capas de enlace y de datos físicos.

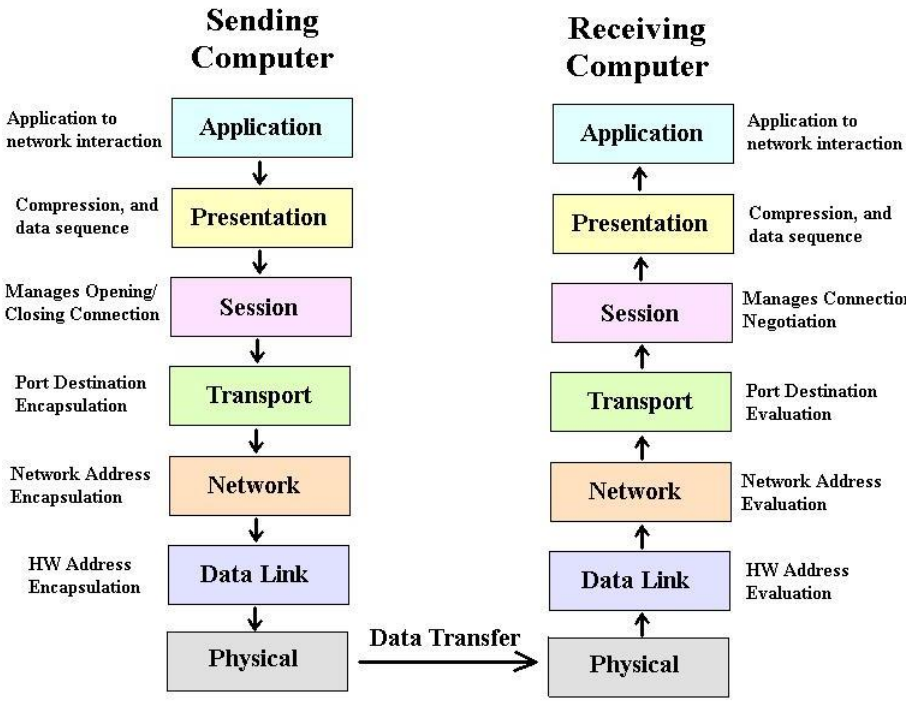

Figura 2. 13 Interacción de la capa de Red. [22]

A continuación mostramos una lista de estas 5 capas de TCP/IP:

- Capa Física
- Capa de Enlace
- Capa de Red
- Capa de Transporte
- Capa de Aplicación

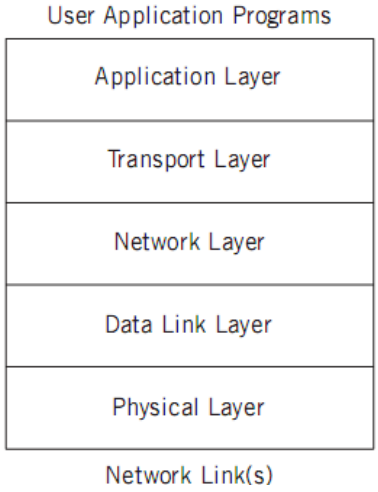

Figura 2. 14 Las 5 capas de TCP/IP. [23]

1.- La primera capa (capa física), se encarga de detalles como los tipos de cables, tipos de señales, codificación, etc.

2.- La segunda capa (capa de enlace), se encarga del procedimiento de acceso a los datos y de la corrección de errores.

3.- La tercera capa (capa de red), se encarga de la transmisión de datos a distancia. Esta capa asegura que los datos encuentren el camino al destinatario a través de diversas redes.

4.- La cuarta capa (capa de transporte), recibe los datos de las aplicaciones en un orden, y asegura su envío, y orden para componer el mensaje original correctamente. Evita la perdida de paquetes.

5.- La quinta capa representa finalmente el procesamiento de datos por parte de la aplicación.

Cada capa necesita una cierta información adicional para poder cumplir con su tarea. Esta información se encuentra en el encabezado (*header*) de cada paquete. Cada capa añade un pequeño bloque de datos (cabeza de protocolo) al paquete.

- El protocolo Ethernet IEEE 802.3 soluciona los problemas de la Capa física y enlace.
- El protocolo IP soluciona los problemas de la Capa de red.
- En la capa de transporte tenemos los protocolos TCP y UDP, el primero es más fiable que el segundo, sin embargo el protocolo UDP proporciona una comunicación más sencilla y puesto que la longitud de cabecera ocupa menos bits este protocolo es más rápido, además que UDP está orientado a datagramas lo cual es una característica deseable en un sistema en tiempo real.

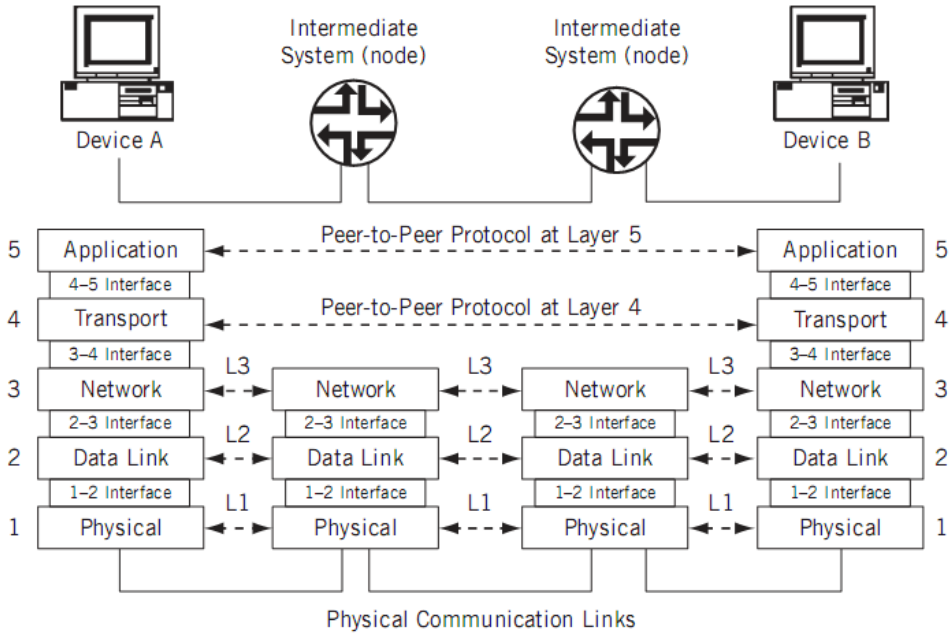

Figura 2. 15 Protocolos e interfaces, que muestran cómo los dispositivos se conectan sólo físicamente en la capa inferior (capa 1). [23]

Es importante tener en cuenta que funcionalmente, nodos intermedios sólo requieren las tres capas inferiores del modelo, tal como se muestra en la Figura 2.15.

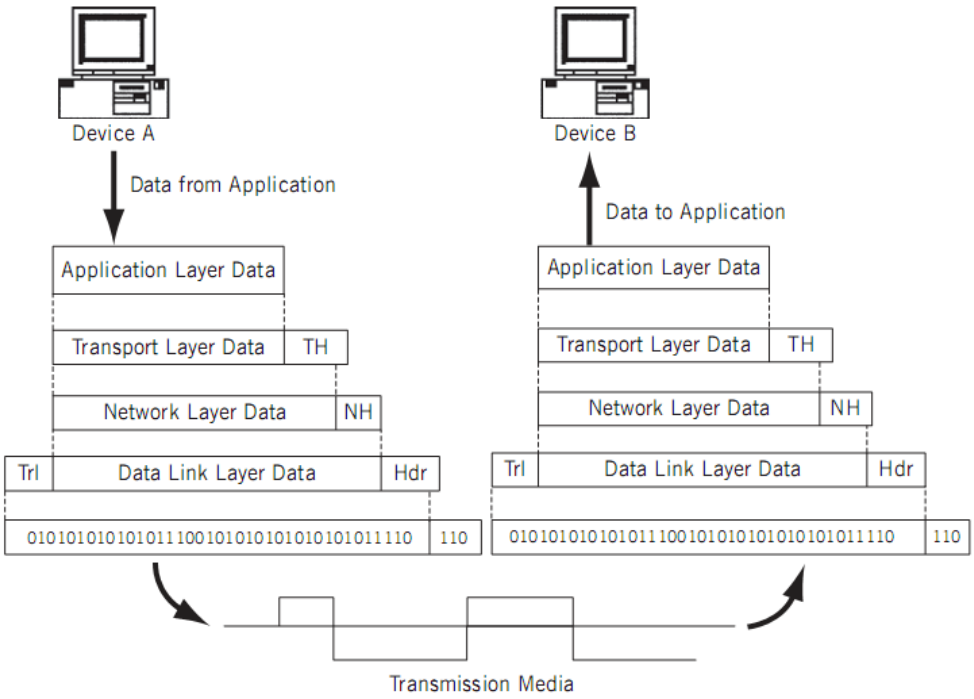

Figura 2. 16 Encapsulación y cabeceras (*headers*) TCP/IP. [23]

En la Figura 2.16 debe tenerse en cuenta que la corriente no estructurada de bits representa marcos con distinto contenido.

2.5.1 Protocolo Ethernet

Este protocolo también es conocido como IEEE 802.3, y usa CSMA/CD (Acceso Múltiple por Detección de Colisiones). Diferencia equipos asignándoles una dirección física de 48 bits, también conocida como MAC (Media Access Control) o dirección Ethernet.

La data es encapsulada según se muestra en la siguiente figura:

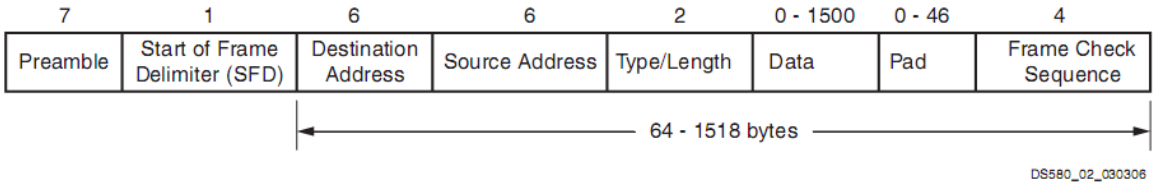

Figura 2. 17 Trama Ethernet (tamaño de campos en bytes). [24]

Siendo cada uno de los campos:

- Preámbulo.

- Delimitador de inicio de trama.
- Dirección destino.
- Dirección fuente.
- Tipo/Tamaño.
- Data.
- Pad.

- **Preámbulo:** Esta trama es usada para la sincronización y debe contener 7 bytes con el patrón “10101010”, el patrón es transmitido de izquierda a derecha.
- **Delimitador de inicio de trama:** Este campo marca el inicio de la trama y debe contener el patrón “10101011”, el patrón es transmitido de izquierda a derecha.
- **Dirección destino:** Este campo es de 6 bytes de longitud y es transmitido con el LSB primero.
- **Dirección fuente:** Este campo es de 6 bytes de longitud y es transmitido con el LSB primero.
- **Tipo/Tamaño:** Este campo es de 2 bytes de longitud y sirve para especificar el protocolo usado en el siguiente nivel de la capa de abstracción, en este caso sería 0x0800 para IPv4, 0x86d0 para IPv6 y 0x0806 para ARP.
- **Data:** Este campo puede tener una longitud desde 0 bytes a 1500 bytes. Este campo es transmitido con el LSB primero. Para el caso de transmisión este campo debe ser siempre proveído y para el caso de recepción este campo es siempre conservado en el paquete de datos.
- **Pad:** Este campo varia de 0 a 46 bytes de longitud y es usado para asegurar que la longitud de la trama sea al menos de 64 bytes en longitud (no se considera ni el preámbulo ni el campo FCS). Para el caso de transmisión este campo es insertado automáticamente y siempre retenido en el caso de recepción.

2.5.2 Protocolo IP

El Protocolo de Internet está diseñado para su uso en sistemas interconectados de redes de comunicación informáticas de conmutación de paquetes [25]. El protocolo de Internet proporciona transmisión de bloques de datos llamados datagramas procedentes de fuentes de destinos, donde las fuentes y los destinos son hosts identificados por direcciones de longitud fija. El protocolo de Internet también proporciona la fragmentación y reensamblaje de datagramas largos, si es necesario, para transmisión a través de redes de "trama pequeña".

Un resumen del contenido de la cabecera IP es el siguiente:

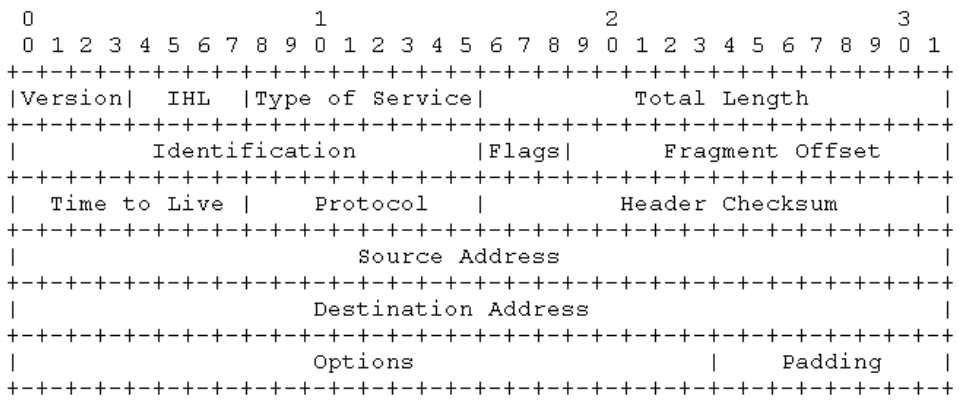

Figura 2. 18 Ejemplo de Cabecera de Datagrama IP. [25]

Versión *(Version)*: 4 bits

El campo *Versión* indica el formato de la cabecera IP. Para la versión IPv.4 este campo debe tener el valor hexadecimal 0x4.

IHL (*Internet Header Length*): 4 bits

Es la longitud de la cabecera en palabras de 32 bits, y por tanto apunta al comienzo de los datos. Nótese que el valor mínimo para una cabecera correcta es 5.

Tipo de Servicio (*Type of Service*): 8 bits

Provee una indicación de los parámetros abstractos de la calidad de servicio deseada. Estos parámetros se usarán para guiar la selección de los parámetros de servicio reales al transmitir un datagrama a través de una red en particular. Algunas redes ofrecen prioridad de servicio, la cual trata de algún modo el tráfico de alta prioridad como más importante que el resto del tráfico (generalmente aceptando sólo tráfico por encima de cierta prioridad en momentos de sobrecarga). La elección más común es un compromiso a tres niveles entre baja demora, alta fiabilidad, y alto rendimiento.

Bits 0-2: Prioridad.

Bit 3: 0 = Demora Normal, 1 = Baja Demora.

Bit 4: 0 = Rendimiento Normal, 1 = Alto rendimiento.

Bit 5: 0 = Fiabilidad Normal, 1 = Alta fiabilidad.

Bits 6-7: Reservado para uso futuro.

Longitud Total (*Total Length*): 16 bits

Es la longitud del datagrama, medida en octetos, incluyendo la cabecera y los datos. Este campo permite la longitud de un datagrama ser hasta de 65, 535 octetos.

Tales longitudes no son prácticas para la mayoría de *hosts* y redes. Todos los *hosts* deben estar preparados para aceptar datagramas de hasta 576 octetos (llegando en su totalidad o en fragmentos). Es recomendado que *hosts* sólo envíen datagramas más largos a 576 octetos si ellos han asegurado que el destinatario está preparado para aceptar datagramas más largos.

Identificación (*Identification*): 16 bits

Valor de identificación asignado por el remitente como ayuda en el ensamblaje de fragmentos de un datagrama.

Flags (indicadores): 3 bits

Son diversos indicadores de control.

Bit 0: reservado, debe ser cero.
Bit 1: (DF) 0 = Puede fragmentarse, 1 = No fragmentar.
Bit 2: (MF) 0 = Último fragmento, 1 = Más Fragmentos

Posición del Fragmento (*Fragment Offset*): 13 bits

Este campo indica a que parte del datagrama pertenece este fragmento. La posición del fragmento se mide en unidades de 8 octetos (64 bits). El primer fragmento tiene posición 0.

Tiempo de Vida (*Time to Live*): 8 bits

Este campo indica el tiempo máximo que el datagrama tiene permitido permanecer en el sistema internet. Si este campo contiene el valor cero, entonces el datagrama debe ser destruido.

Protocolo: 8 bits

Este campo indica el protocolo del siguiente nivel usado en la parte de datos del datagrama internet. En caso el siguiente nivel corresponda al protocolo UDP el número decimal 17 debería llenar este campo.

Suma de Control de Cabecera (*Header checksum*): 16 bits

Suma de Control de la cabecera solamente. Dado que algunos campos de la cabecera cambian (por ejemplo: el tiempo de vida), esta suma es recalculada y verificada en cada punto donde la cabecera internet es procesada.

El algoritmo de la suma de control (*The checksum algorithm*) es:

El campo suma de control es el complemento a uno de 16 bits de la suma de los complementos a uno de todas las palabras de 16 bits de la cabecera. Para los propósitos del cálculo de la suma de control, el valor inicial del campo de suma de control es cero.

Dirección de Origen: 32 bits

La dirección de origen conocida también como dirección IP origen.

Dirección de Destino: 32 bits

La dirección de destino conocida también como dirección IP destino.

***Options*: variable**

Options puede aparecer o no en los datagramas. Ellos deben ser implementados en todos los módulos IP (*host y gateways*). Lo que es opcional es su transmisión en cualquier datagrama en particular, no su implementación. El campo *option* es de tamaño variable.

***Padding*: variable**

Es usado para asegurar que la cabecera de internet finaliza en un límite de 32 bits.

2.5.3 Protocolo UDP

UDP (*User Datagram Protocol*) es un protocolo de capa de transporte sencillo y orientado a datagramas: cada operación de salida de un proceso produce exactamente un datagrama UDP, que causa un datagrama IP para ser enviado [26]. Esto es diferente de un protocolo orientado a flujo tal como el protocolo TCP, donde la cantidad de datos escritos por una aplicación pueden tener poca relación con lo que realmente se envió en un solo datagrama IP. La Figura 2.19 muestra la encapsulación de un datagrama UDP como un datagrama IP.

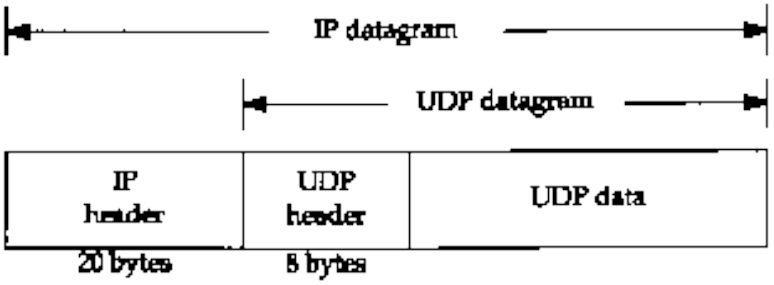

Figura 2. 19 Encapsulación UDP.

UDP no provee confiabilidad: envía los datagramas que la aplicación escribe en la capa IP, pero no hay ninguna garantía de que alguna vez lleguen a su destino [27]. La aplicación tiene que preocuparse por el tamaño de los datagramas IP resultante [28]. Si excede la MTU (*Maximun Transmission Unit*) de la red, el datagrama IP es fragmentado. Esto se aplica a cada red que el datagrama atraviesa desde el origen al destino, no sólo la primera red conectada al host de envío.

Cabecera UDP

La Figura 2.20 muestra los campos en una cabecera UDP. [27]

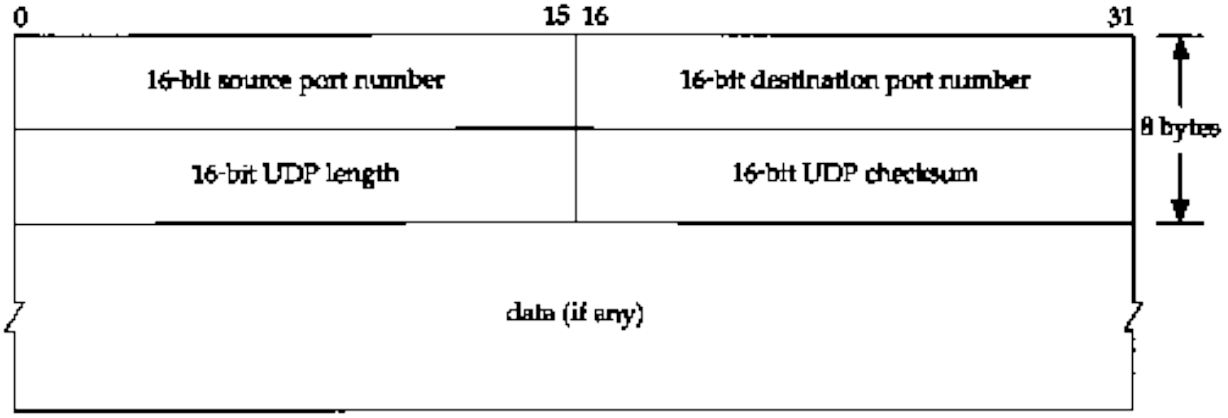

Figura 2. 20 Cabecera UDP. [26]

Campos de la Cabecera UDP

Puerto Origen (*Source Port Number*): 16 bits

Es un campo opcional, cuando tiene significado, éste indica el puerto del proceso que envía (o usado para enviar el datagrama), y puede ser asumido como el puerto al que la respuesta debe ser direccionada en la ausencia de cualquier otra información. Si no es usado, el valor cero es insertado.

Puerto Destino (*Destination Port Number*): 16 bits

Numero de puerto de la máquina destino, tiene sentido dentro del contexto de una dirección de internet particular de destino.

Longitud (*UDP Length*): 16 bits

Es la longitud en bytes (u octetos) del datagrama UDP incluyendo cabecera y datos, lo cual significa que el número mínimo de la longitud es 8.

Suma de verificación (*UDP Checksum*): 16 bits

Es la suma de comprobación de errores del mensaje, viene a ser complemento a uno de 16 bits, de la suma de los complementos a uno de una seudo cabecera de información de una cabecera IP, si la transmisión de este campo contiene solo ceros entonces significa que el transmisor no ha generado esta suma. Es completado con ceros si es necesario para hacer un número múltiplo de dos de octetos.

2.6 Plataforma de Desarrollo Java

Para empezar a describir la metodología de trabajo en esta plataforma, partimos haciéndonos la siguiente pregunta: ¿Por qué Java es una buena idea?

Figura 2. 21 Icono representativo del éxito de la plataforma Java.

A continuación se enlistan algunas de las respuestas:

- Java no es sólo un lenguaje (JLS: Java Language Specification).

- Es una plataforma de ejecución, Java Virtual Machine Specification (JVMS).

- Semántica WORA (Write Once, Run Anywhere)

  ⇨ Lo cual quiere decir “escribes una vez y lo corres en cualquier parte”.

- Soporta concurrencia.

- Modelo de programación orientada a objetos.

- El lenguaje y el modelo de memoria están orientados a la construcción de software más fiable.

- Java empezó como un proyecto de investigación sobre control distribuido de dispositivos de electrónica de consumo.

- Fuerte control de tipos (strong typing).

- Orientado a objetos; tomado de C++.

- ⇨ Pero sin punteros, aritmética de punteros, sobrecarga de operadores, estructuras ni uniones.
- ⇨ No hay herencia múltiple, pero una clase puede implementar múltiples interfaces.

- Librerías de clases.
- Programación de la GUI (Interfaz gráfica de usuario), ficheros, hilos, I/O, red, etc.
- Se puede restringir el conjunto de recursos de la máquina a los que las aplicaciones pueden acceder.

2.6.1 Programación Orientada a Objetos (POO)

La programación orientada a objetos (POO) es un modelo de programación que utiliza objetos ligados mediante mensajes, para la solución de problemas [29]. Puede considerarse como una extensión natural de la programación estructurada en un intento de potenciar los conceptos de modularidad y reutilización del código.

Veamos un ejemplo. Considere una entidad bancaria. En ella identificamos entidades que son cuentas: cuenta del cliente 1, cuenta del cliente 2, etc. Pues bien, una cuenta puede verse como un objeto que tiene unos atributos, nombre, número de cuenta y saldo, y un conjunto de métodos como IngresarDinero, RetirarDinero, AbonarIntereses, SaldoActual, Transferencia, etc. En el caso de una transferencia:

```
cuenta01.Transferencia(cuenta02);
```

Transferencia sería el mensaje que el objeto cuenta02 envía al objeto cuenta01, solicitando le sea hecha una transferencia, siendo la respuesta a tal mensaje la ejecución del método **Transferencia**. Trabajando a este nivel de abstracción, manipular una entidad bancaria resultará algo muy sencillo.

Para poder entender de una manera más ilustrativa cómo es que se trabaja en la POO, partamos por recordar los Tipo de datos manejados por C, el cual maneja tipo de datos primitivos tales como:

- ➢ Int, Float, Double, Char y String.

En cambio en un lenguaje de POO, tal como Java, se manejan estos mismos datos primitivos y además los de tipo objeto.

- Se necesita tener un esqueleto del objeto.

- Por ejemplo si queremos crear un esqueleto, en este caso tendría las siguientes partes: cabeza, tronco y extremidades. El esqueleto en Java tiene el nombre de Clase.

Entonces si se desea crear un Objeto Jorge que sea del tipo humano, asimismo se puede crear un Objeto Pedro, etc.

Para crear un objeto se instancia a partir de la clase, entonces para poder crear un objeto es necesario tener un clase e instanciando esta clase se crean los objetos.

Analogía con lenguaje C:

- Esta declaración: int x=0;
- Es equivalente a: Humano Jorge
  Humano Paris
  Humano José

POO (Programación Orientada a Objetos)

Clase	
Humano	
	- Cabeza
Partes:	- Tronco
	- Extremidades
Acciones	- correr();
(Métodos)	- caminar();
	- hablar();

MECANISMOS BASICOS DE LA POO

Los mecanismos básicos de la programación orientada a objetos son: objetos, mensajes, métodos y clases.

Objetos

Un programa orientado a objetos se compone solamente de *objetos*, entendiendo por objeto una encapsulación genérica de datos y de los métodos para manipularlos. Dicho de otra forma, un ***objeto*** es una entidad que tiene unos atributos particulares, las ***propiedades***, y unas formas de operar sobre ellos, los ***métodos***.

Por ejemplo, una venta de una aplicación de Windows es un objeto. El color de fondo, la anchura, la altura, etc. son propiedades. Las rutinas, lógicamente transparentes al usuario, que permiten maximizar la ventana, minimizarla, etc. son métodos.

Mensajes

Cuando se ejecuta un programa orientado a objetos, los objetos recibiendo, interpretando y respondiendo a ***mensajes*** de otros objetos. Esto marca una clara diferencia con respecto a los elementos de datos pasivos de los sistemas tradicionales. En la POO un ***mensaje*** está asociado con un método, de tal forma que cuando un objeto recibe un mensaje la respuesta a ese mensaje es ejecutar el método asociado.

Por ejemplo, cuando un usuario quiere maximizar una ventana de una aplicación Windows, lo que hace simplemente es pulsar el botón de la misma que realiza esa acción. Eso, provoca que Windows envíe un mensaje a la ventana para indicar que tiene que maximizarse. Como respuesta a este mensaje se ejecutará el método programado para ese fin.

Métodos

Un ***método*** se implementa en una ***clase*** de objetos y determina cómo tiene que actuar el objeto cuando recibe el ***mensaje*** vinculado con ese método. A su vez, un ***método*** puede también enviar ***mensajes*** a otros objetos solicitando una acción o información.

En adición, las ***propiedades*** (atributos) definidas en la clase permitirán almacenar información para dicho objeto.

Cuando se diseña una clase de objetos, la estructura interna del ***objeto*** se oculta a los usuarios que lo vayan a utilizar, manteniendo como única conexión con el exterior, los ***mensajes***. Esto es, los datos que están dentro de un objeto solamente podrán ser manipulados por los ***métodos*** asociados al propio objeto.

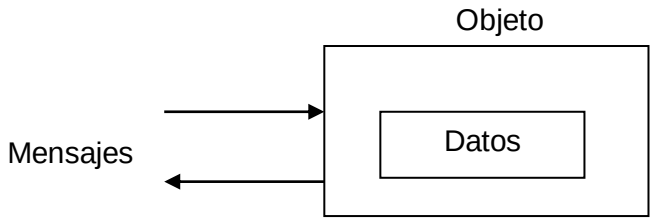

Figura 2. 22 Mensajes en Java.

Según lo expuesto, podemos decir que la ejecución de un programa orientado a objetos realiza fundamentalmente tres cosas:

1. Crea los objetos necesarios.

2. Los mensajes enviados a unos y a otros objetos dan lugar a que se procese internamente la información.
3. Finalmente, cuando los objetos no son necesarios, son borrados, liberándose la memoria ocupada por los mismos.

Clases

Una ***clase*** es un tipo de objetos definido por el usuario. Una ***clase*** equivale a la generalización de un tipo específico de objetos. Por ejemplo, piense en un molde para hacer flanes; el molde es la clase y los flanes los objetos.

Un ***objeto*** de una determinada clase se crea en el momento en que se define una variable de dicha ***clase***. Por ejemplo, la siguiente línea declara el objeto cliente01 de la clase o tipo CCuenta.

```
CCuenta cliente01 = new CCuenta (); // nueva cuenta
```

Algunos autores emplean el término instancia (traducción directa de instance), en el sentido de que una instancia es la representación concreta y específica de una clase; por ejemplo, cliente01 es una instancia de la clase CCuenta. Desde este punto de vista, los términos instancia y objeto son lo mismo.

Cuando se escribe un programa utilizando un lenguaje orientado a objetos, no se definen objetos verdaderos, se definen clases de objetos de objetos, donde una clase se ve como una plantilla para múltiples objetos con características similares.

Afortunadamente no se tienen que escribir todas las clases que se necesitan en un programa, porque Java nos proporciona una biblioteca de clases estándar para realizar las operaciones más habituales que podamos requerir.

Paquetes Java [30]

Un Paquete en Java es un contenedor de clases que permite agrupar las distintas partes de un programa cuya funcionalidad tienen elementos comunes.

Paquetes importantes de Java

Un Paquete en Java es un contenedor de clases que permite agrupar las distintas partes de un programa cuya funcionalidad tienen elementos comunes.

Estos son los paquetes más importantes de la API de Java:

Tabla 2. 3 Paquetes importantes de la API de Java. [30]

Paquete	**Descripción**
java.applet	Contiene clases para la creación de *applets*.
java.awt	Contiene clases para crear interfaces de usuario con ventanas.
java.io	Contiene clases para manejar la entrada/salida.
java.lang	Contiene clases variadas pero imprescindibles para el lenguaje, como *Object, Thread, Math…*
java.net	Contiene clases para soportar aplicaciones que acceden a redes TCP/IP.
java.util	Contiene clases que permiten el acceso a recursos del sistema, etc.
javax.swing	Contiene clases para crear interfaces de usuario mejorando la *AWT*. (*Abstract Window Toolkit*)

2.6.2 Clases usadas para la Interface Gráfica de Usuario (GUI)

El desarrollo de la interface de usuario, cuenta con los siguientes elementos:

- **JButton:** Usado para el ingreso de datos.
- **Ellipse2D:** Usado para mostrar salida de datos.
- **Rectangle2D:** Usado para mostrar salida de datos.

Para el desarrollo de una interface de usuario puede hacerse uso de las herramientas de diseño de un Entorno de Desarrollo Integrado (IDE), tal como Netbeans, y mediante esta interface de diseño agregar los elementos deseados a nuestra interface de usuario. Sin embargo, en el diseño de la presente interface de usuario se prosiguió de la siguiente manera:

El modo de declaración de estos elementos se realizó de la siguiente manera:

- JButton b1 = new JButton("SW7");

- Ellipse2D e2d_8 = new Ellipse2D.Float(400,150,30,30); // LED00

- Rectangle2D r2d_1 = new Rectangle2D.Float(50, 300, 30, 30); // SW07

- Rectangle2D r2d_dpi = new Rectangle2D.Float(195, 570, 10, 10); // DP DEL DISPLAY IZQUIERDO

- Para mostrar los cambios de color de los diferentes elementos de salida, se implementa la lógica necesaria que los colores sean asignados según correspondencia.

- Para el envío de datos es necesario agregar propiedades de “Listener” al elemento JButton, para así poder “escuchar” los valores del puntero del mouse al acercarse al botón o elemento respectivo.

Las clases más importantes para esta interface gráfica son:

- La clase AWT y la clase Swing.

import java.awt.*; [31]

import javax.swing.*; [32]
Un artículo al “pintado” con AWT y Swing se presenta en [33].

2.6.3 Clases usadas para la Comunicación Ethernet

La implementación de la comunicación Ethernet se desarrolló mediante el diseño de sockets, buffers y un datagrama respectivo.

```
DatagramSocket socket = new DatagramSocket(4405);
        //buffer = new byte [1024];
buffer = new byte [12]; // one more (from 3 to 6) to send data to display

        //System.out.println("Data recibida :"+hostDestino);
intpuertoDestino =datagrama.getPort();
data = datagrama.getData();
```

import java.net.*;

Proporciona las clases para la implementación de aplicaciones de red. [34]

import java.io.*;

Proporciona las clases para la implementación de entrada y salida.

CAPÍTULO III

CONCEPCIÓN DEL SISTEMA

3.1 Planteamiento del Problema y Propuesta Solución

Parte del problema es descrito en el capítulo I en la problemática. Sin embargo, en este capítulo se desea plantear de manera clara los puntos que requiere cumplir el sistema a diseñar, y así poder brindar una propuesta solución.

Los requerimientos del sistema pueden indicarse en los siguientes puntos:

- Sistema debe contar con entradas y salidas, tanto digitales como analógicas.
- Sistema debe enviar y recibir data vía Ethernet.
- Sistema debe ser reconfigurable.
- Se debe contar con una interface de usuario para lograr control del sistema.
- Debe ser posible acceder de manera remota al sistema.

El sistema requerido se puede esquematizar como se muestra en la Figura 3.1:

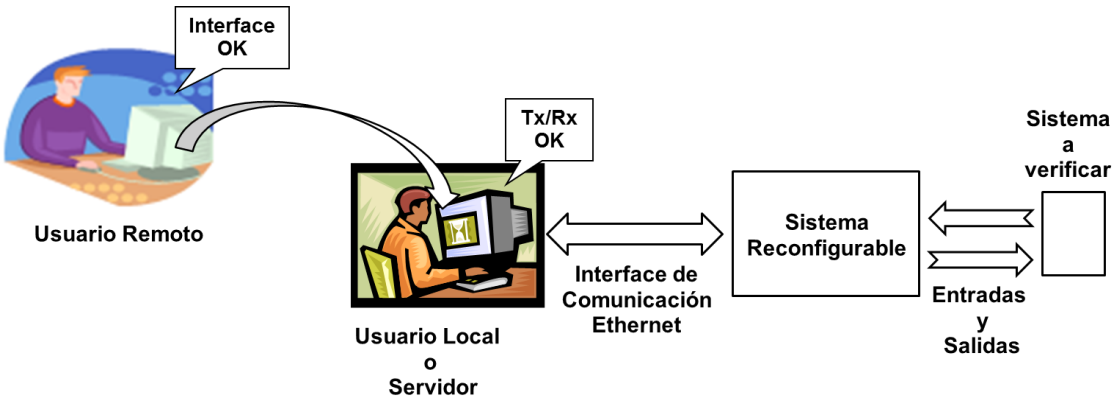

Figura 3. 1 Concepción del Sistema requerido.

La propuesta solución es la siguiente:

- Usar un módulo de desarrollo para FPGA's (dispositivo programable en campo) para diseñar el sistema reconfigurable.
- El sistema tendrá como núcleo principal (corazón del sistema) a un procesador *Soft-Core* embebido en FPGA.
- Diseñar los bloques de hardware (IP Cores) necesarios para controlar todos los periféricos requeridos, con los cuales cuenta el módulo de desarrollo.
- Diseñar Interface de usuario usando plataforma Java, la cual implementará el código necesario para realizar la comunicación Ethernet.
- Usar software compatible a la plataforma de trabajo para realizar el acceso remoto. De manera ideal, este software debe ser independiente de la plataforma.

3.2 Descripción y Funcionamiento del Sistema

El sistema a diseñar será un Sistema Digital basado en Procesador Soft-Core, que contará con los diferentes protocolos de comunicación que requiere. El procesador brindará un nivel de abstracción al sistema, y así será posible combinar las ventajas de abstracción y de diseño a nivel hardware que brindan los dispositivos programables como los FPGA's.

El sistema programará un dispositivo electrónico a distancia, y será capaz de monitorear y controlar los periféricos de este dispositivo vía remota, lo cual nos brindará una plataforma de laboratorio virtual.

El funcionamiento del sistema será de la siguiente manera:

1) En la tarjeta de desarrollo para FPGA (Spartan 3E Starter Board) se diseña el hardware y software necesario, el cual estará basado en procesador Microblaze, para poder controlar las entradas y salidas (digitales y analógicas), así como también los demás periféricos necesarios.
2) La Interface de Usuario en Java realizará comunicación vía Ethernet. Hará uso de sockets para la comunicación y creación del datagrama. Esta interface deberá ser lo más intuitiva posible.
3) El software de acceso remoto puede acceder a la máquina host en donde se encuentra la conexión con el sistema a evaluar, así de ésta manera se podrá verificar el funcionamiento del dispositivo conectado.
4) El usuario ubicado a distancia podrá acceder al sistema encargado de la verificación, así como también podrá programar el dispositivo a verificar.

3.2.1 Arquitectura del Sistema

La arquitectura del sistema es conformada por el diseño Software y Hardware de la plataforma.

A continuación se detallan algunos de los puntos clave de los sistemas Software y Hardware de la plataforma.

Sistema Hardware reconfigurable

El hardware del sistema debe ser reconfigurable como ha sido indicado anteriormente, esto permitirá reconfigurar el sistema cada vez que sea necesario.

En este bloque se diseñaran todos los componentes hardware necesario para manejo de los protocolos de comunicación requeridos.

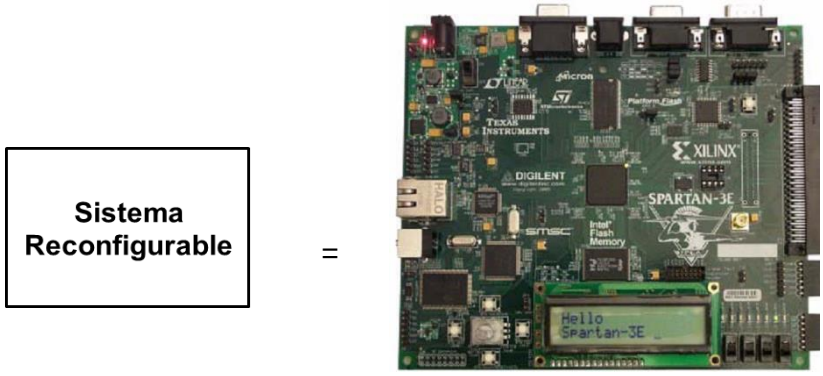

Figura 3. 2 Dispositivo para Sistema Reconfigurable.

Interface de Comunicación Ethernet

Este bloque es el encargado de las conexiones que se realizan entre el sistema reconfigurable y el usuario local. Por lo tanto será variable en tamaño, dependiendo del tamaño de la red construída para el sistema de comunicación.

Figura 3. 3 Dispositivo para Interface de Comunicación Ethernet.

Sistema Software (Plataforma Java)

El diseño del sistema software se realiza en plataforma Java. Particularmente para la interface de usuario se hade uso de las API de Java 2D. Este bloque basado en la plataforma Java, presenta la característica de ser portable e independiente de la plataforma, tal como es requerido en el diseño del sistema.

Figura 3. 4 Icono representativo de la Plataforma Java

Procesador Embebido Microblaze: Este procesador será embebido en el Sistema Reconfigurable para poder establecer un nivel de abstracción más alto, por lo tanto el sistema estará basado en procesador Soft-Core, y luego se hará uso de otros componentes hardware que serán diseñados según sea requerido.

Para realizar el diseño de este procesador y el desarrollo Software del mismo será necesario hacer uso de la herramienta EDK de Xilinx. Es por ello que a continuación se describen algunos pasos generales para el uso de esta herramienta, los cuales no pretenden constituir un manual de usuario, pero serán de ayuda para proyectos en los cuales se haga uso de ella.

Uso de las herramientas del Kit de Desarrollo Embebido (EDK)

Se trabajará con la herramienta Xilinx Platform Studio (XPS). Esta pequeña guía nos muestra el diseño genérico para distintas pruebas de software con un mismo procesador Microblaze.

Figura 3. 5 Kit de Desarrollo Embebido (Herramientas EDK).

Creación de un proyecto en Xilinx Platform Studio

Pasos generales a seguir:

- Abrir entorno de la herramienta XPS, la cual se encuentra ubicada según se muestra en la Figura 3.5.
- Seleccionar directorio donde se almacenará proyecto, así se podrán almacenar todos los archivos generados en el desarrollo del mismo.

- Elegir el hardware que se usará para la descarga del sistema diseñado, es por ello que primero deberá crearse un nuevo diseño en la herramienta y luego se elegirán las características que presenta la tarjeta de desarrollo.
- Implementar procesador del tipo Soft-Core, es decir será implementado y sintetizado usando los bloques lógicos de nuestro FPGA Spartan 3E.
- También se tiene que seleccionar con cuidado las características del FPGA (Spartan 3E, grado de velocidad -4, etc.)
- Luego se selecciona las características relacionadas al procesador Microblaze.
- Luego las características de los periféricos que se están añadiendo (rapidez, flanco de subida o bajada, Ethernet, buses, ...)
- Se selecciona => Memory test y deseleccionamos => Peripherical Self-Test.
- Por último se hace clic en iniciar con la herramienta XPS, según se solicite.

3.2.2 Características principales de los dispositivos empleados

A continuación se enumeran los componentes principales empleados:

1) PC, ésta contiene la interfaz de usuario para poder visualizar la data.
2) Tarjeta de desarrollo para FPGA's, Spartan 3E Starter Board. [24]
3) DES-1008D (SWITCH SOHO 8 PORT). [35]
4) Dispositivo a testear.

El principal componente es la tarjeta de desarrollo para FPGA, ya que en ella se realizará la grabación de nuestro diseño creado. Otros hardwares adicionales son módulos que pueden conectarse a esta tarjeta de desarrollo, tales como por ejemplo un hardware de un puente H para realizar una interfaz de potencia u otros disponibles, según la tarjeta elegida. [36]

A continuación mostramos 2 modelos de tarjetas disponibles del fabricante Xilinx, debido a las facilidades con las que se cuenta se trabajará con la tarjeta Spartan 3E Starter Board, que se muestra en la Figura 3.7.

Figura 3. 6 Avnet Spartan-6 LX9 MicroBoard de Xilinx.

Sin embargo es apropiado mencionar que la tarjeta de la Figura 3.6 también cuenta con la interface hardware para realizar la comunicación Ethernet. Por lo cual para proyectos similares también se podría contar con este dispositivo, el cual es de dimensiones inferiores.

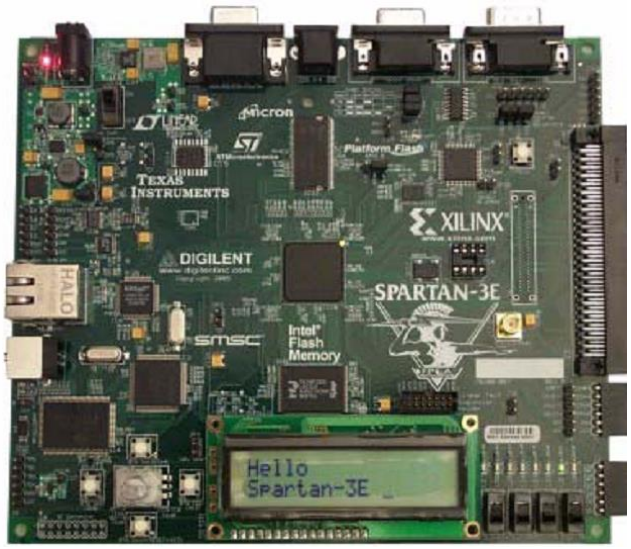

Figura 3. 7 Tarjeta de desarrollo Spartan 3E Starter Board. [24]

La tarjeta de desarrollo Spartan-3E Starter Kit de Xilinx utiliza FPGA Spartan-3E-XC3S500E 4FG320. La placa incluye dos puertos serie RS232, 4 interruptores DIP, 4 botones, 8 LED's, un puerto VGA, pantalla LCD de caracteres, puerto PS/2, SPI convertidor analógico a digital, SPI digital a analógico, 10/100 del puerto Ethernet, 2 MB SPI flash, 16 MB de flash NOR paralela y 32 MB de SDRAM DDR.

Otro componente importante para el diseño del sistema es el Switch, el cual se muestra en la Figura 3.8, ya que mediante este dispositivo se creará la pequeña interfaz de comunicación Ethernet que es requerida en el sistema.

Figura 3. 8 Switch DES-1008D de 8 puertos. [34]

El switch DES-1008D es un switch de alto rendimiento y gran versatilidad. Está diseñado para reforzar el rendimiento del SOHO (*Small Office/Home Office*) y la pequeña empresa, otorgando flexibilidad y manejo a 10/100Mbps. Está provisto de 8 puertas auto-detect, lo que permite que grupos de trabajo aumenten el rendimiento en la red.

8 PUERTAS 10/100MBPS

Este switch provee de 8 puertas con soporte al estándar Nway. Las puertas tienen la capacidad de negociar las velocidades de red entre 10BASE-T y 100BASE-TX, como también el modo de operación en Half o Full Duplex.

CONTROL DE FLUJO

La arquitectura de *Parallel Switching* para el modo de operación *Store&Fordward*, permite la transferencia de datos en forma directa entre las distintas puertas, con Full Error Checking, eliminando en el tráfico de la red el envío de Paquetes Incompletos, Fragmentados o con Errores de CRC, salvaguardando de esta forma la integridad de los datos.

Tabla 3. 1 Ficha Técnica del Switch DES-1008D. [35]

CARACTERÍSTICAS TÉCNICAS	
PUERTAS	8 (10/100Base-TX)
ESTÁNDARES	IEEE 802.3 10Base-T Ethernet Repeater, IEEE 802u 100Base-TX class II Fast Ethernet repeater y ANSI/IEEE Std 802.3 Nway auto-negotiation
CONECTORES	RJ-45
TRANSFERENCIA	10/100 Mbps *Full Duplex, autodetect*
MÉTODO DE ACCESO	CSMA/CD
MÉTODO DE TRANSMISIÓN	Store-and-forward
TOPOLOGÍA	Estrella
FILTERING ADDRESS TABLE	8 K por dispositivo
PACKET FILTERING/ FORWARDING RATES	148.800 pps por puerta (en *full duplex*)
LEDs INDICADORES	Por puerta: link/activity, velocidad 100Mbps, Full-duplex collision. Por switch : Power
FUENTE DE PODER	Externa.
CONSUMO	8 Watts Máximo Modelo Rev. C2 12 Watts Máximo Modelo Rev. D1
CARACTERÍSTICAS FÍSICAS	
TAMAÑO	*Palm-Size*
DIMENSIONES	197 x 115 x 28 mm Modelo Rev. C2 171 x 98 x 29 mm Modelo Rev. D1
PESO	1,5 Kg
TEMPERATURA DE OPERACIÓN	0ºC a 55ºC
TEMPERATURA DE ALMACENAJE	-25ºC a 55ºC
HUMEDAD	5% a 95% no condensada
EMISSION(EMI)	EMI: CE Class A, C-Tick Class A, FCC Class A, VCCI Class A, BSMI class A
SEGURIDAD	UL/CUL

3.3 Ventajas del sistema

- El sistema combina diseño hardware y software, aprovechando características de ambos desarrollos de diseño.
- Sistema cuenta con entrada de señales analógicas, por lo tanto será posible adaptar señales analógicas a digitalizar y que podrán ser analizadas.
- También se cuenta con salidas analógicas, las cuales pueden ingresar a la planta de trabajo.
- Sistema brinda un nivel de abstracción diferente a otros sistemas convencionales, ya que software y hardware son reconfigurables.

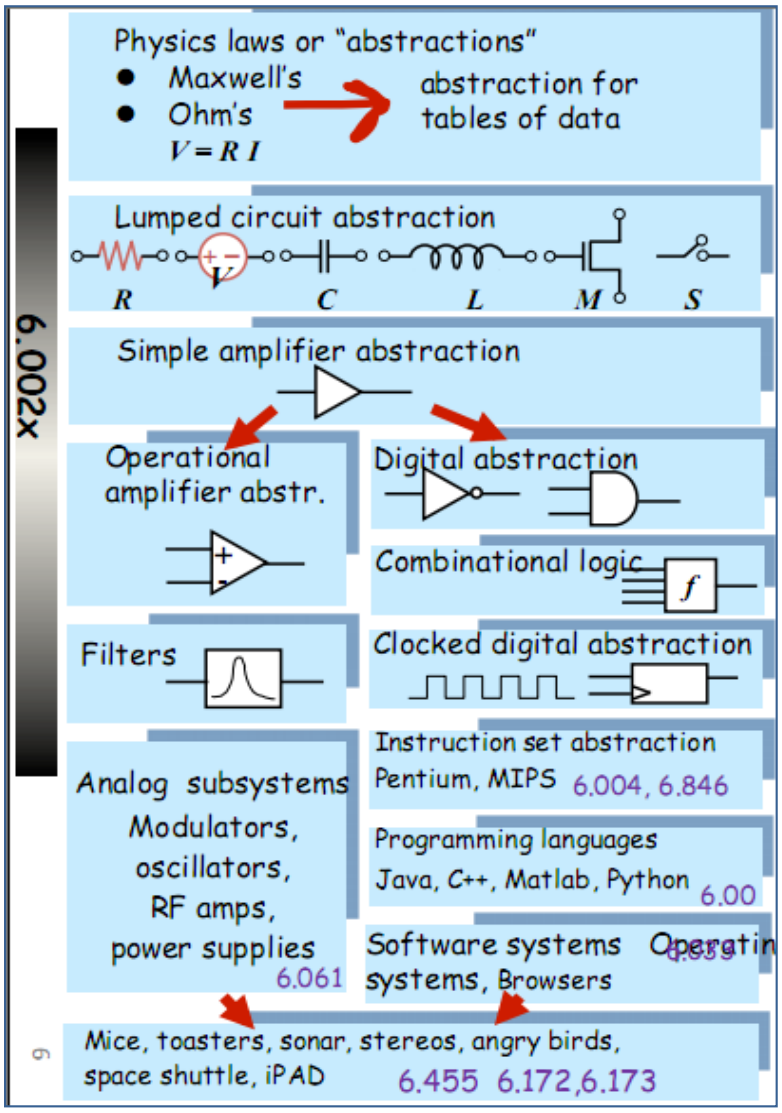

Figura 3. 9 Abstracción y evolución en las tecnologías de diseño. [37]

CAPÍTULO IV

ANÁLISIS Y DISEÑO DEL SISTEMA

4.1 Diseño del Hardware del Sistema

El sistema hardware contará con:

- Un procesador embebido (o incrustado), como se muestra en la Figura 4.1.
- Módulos hardware para la comunicación Ethernet, Serial, Paralelo, etc.

Los módulos hardware requeridos (*IP cores*) para el manejo de las interfaces de comunicación irán conectados al procesador Microblaze mediante el bus PLB (*Processor Local Bus*), como se muestra en la Figura 4.2.

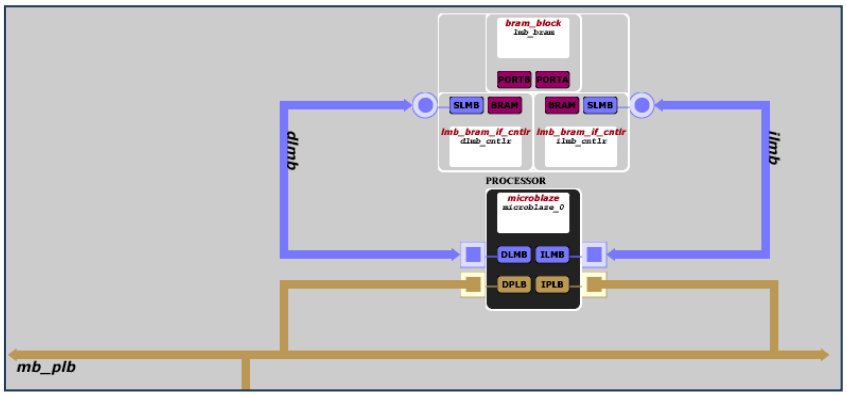

Figura 4. 1 Procesador Microblaze embebido en FPGA.

El sistema utiliza módulos hardware (*IP Cores*) de fabricante y también módulos hardware diseñados por el usuario, tal como se muestra en la Figura 4.2. Para realizar el manejo de los módulos hardware de fabricante, es necesario el estudio de la documentación brindada por éste, para un correcto manejo y así obtener un correcto funcionamiento en el sistema. Los módulos hardware de usuario serán manejados vía procesador según sea descrito en la codificación de estos mismos, el enfoque de diseño de estos está en describir lo requerido para poder configurar los dispositivos (*chips*) requeridos.

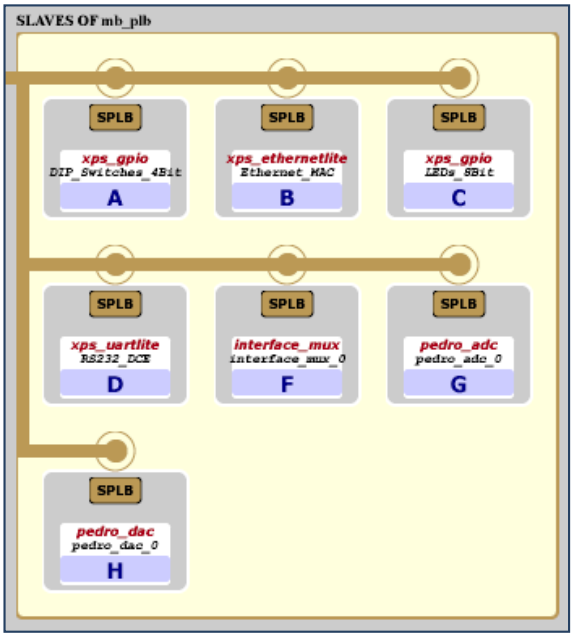

Figura 4. 2 Módulos hardware requeridos.

4.1.1 Diseño del Procesador Embebido Microblaze

Para el diseño del hardware del procesador *Soft-Core* Microblaze usaremos la herramienta XPS (*Xilinx Platform Studio*). [15]

Figura 4. 3 Iniciando la ejecución de la herramienta XPS.

Algunas consideraciones que se deben tener presente para el desarrollo son:

- Se debe trabajar y almacenar todos los archivos de diseño en un directorio específico, cuya dirección no debe contar con espacios en blanco ni tildes, en los caracteres que la especifican.
- Para iniciar el desarrollo en *Xilinx Platform Studio*, se cuenta con 3 opciones de inicio de trabajo. De las 3 opciones elegimos iniciar creando un nuevo diseño.

- Se debe indicar correctamente el código del dispositivo dónde se realizará la implementación del sistema, el cual en el presente proyecto es: **Xilinx Spartan-3E XC3S500E-4FG320**.
- Con la selección del dispositivo se podrá observar el procesador soportado y una pequeña descripción del mismo, tal como muestra la Figura 4.4.

Luego de algunas configuraciones del sistema en general, será necesario indicar configuraciones para el procesador Microblaze. Algunas de estas configuraciones son mostradas en la Figura 4.5.

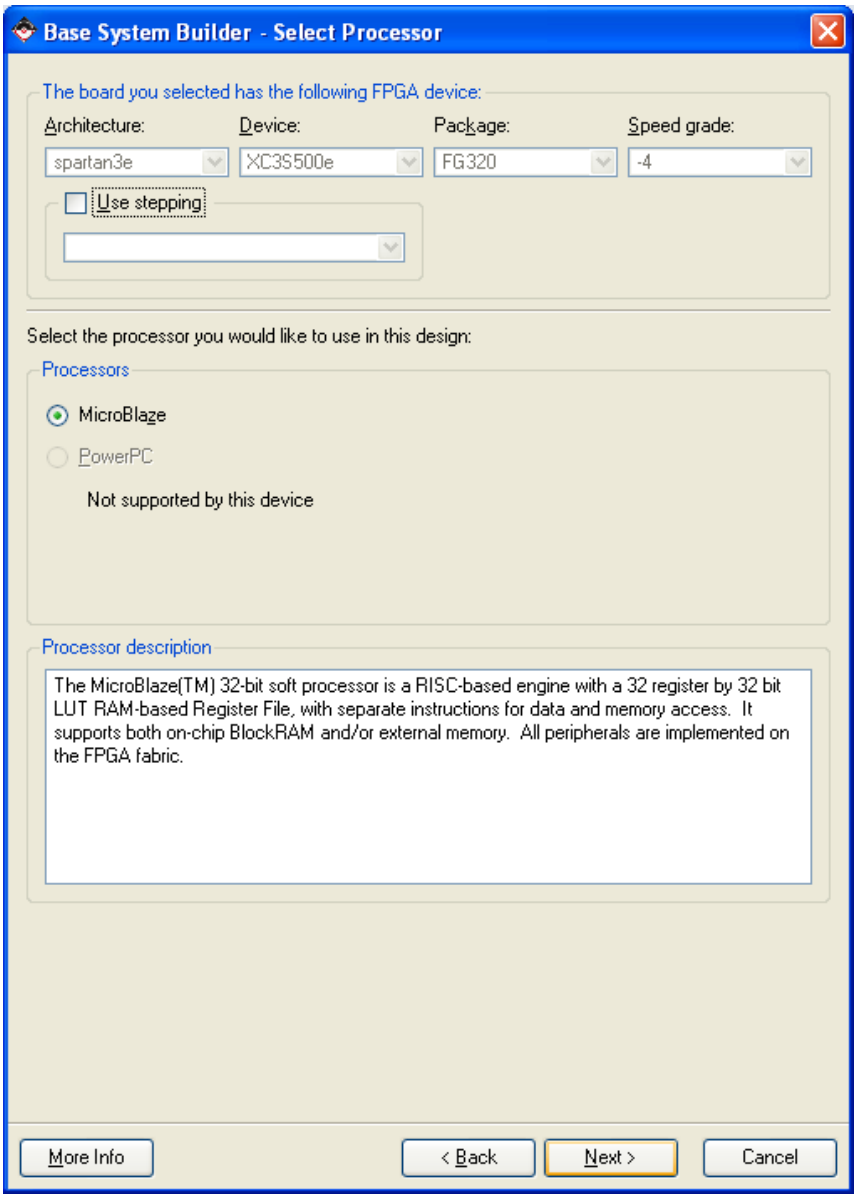

Figura 4. 4 Procesador soportado por tarjeta de desarrollo usada.

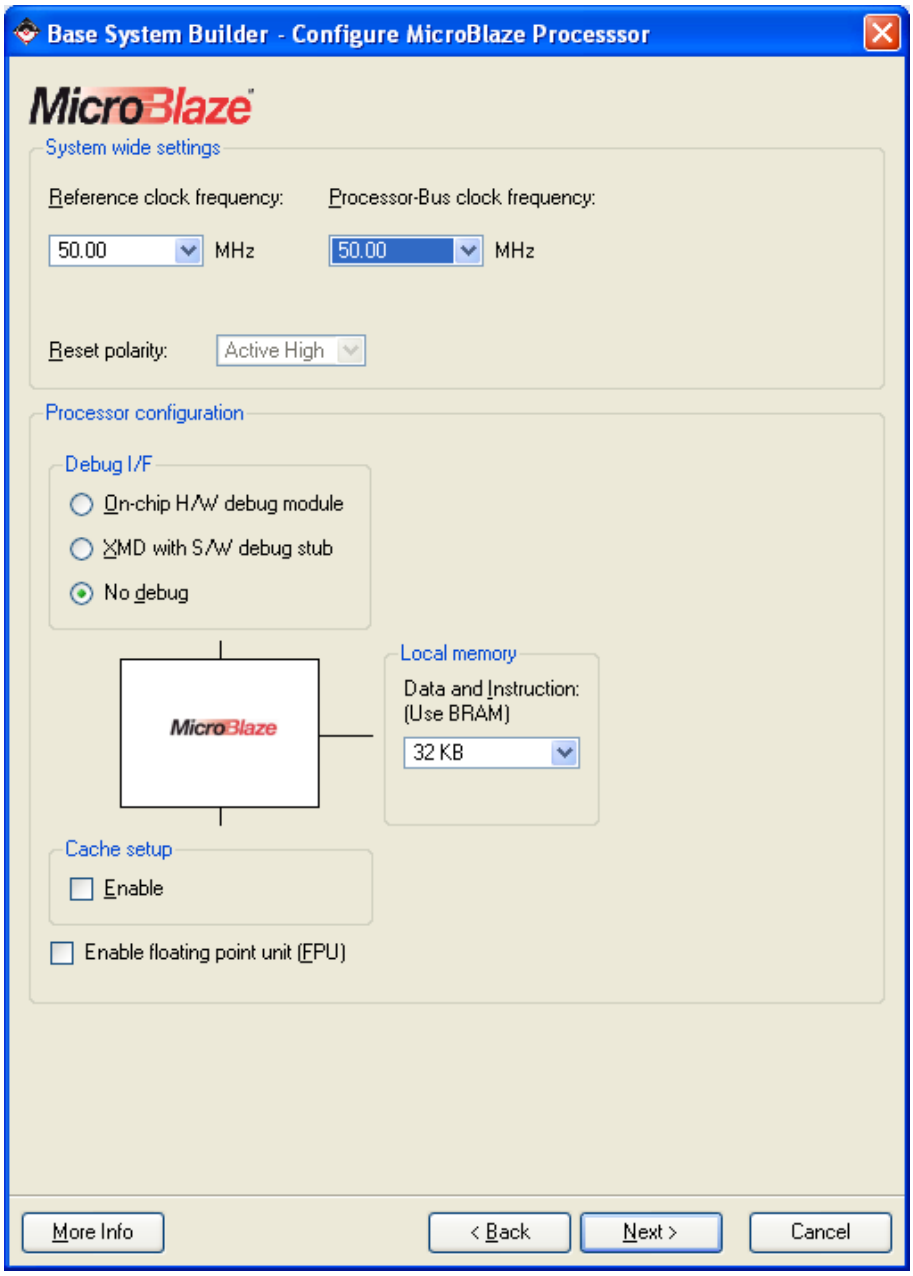

Figura 4. 5 Configuración del Procesador.

Características: [24]

El procesador Soft-Core MicroBlaze es altamente configurable, lo que permite seleccionar un conjunto específico de características requeridas por el diseño.

El conjunto de características fijas del procesador incluye:

- Treinta y dos registros de propósito general de 32bits.
- Instrucción de 32bits con tres operandos y dos modos de direccionamiento.
- Bus de direcciones de 32 bits.

Además de estas características fijas, el procesador MicroBlaze permite la habilitación selectiva de funcionalidades adicionales. La versión del procesador MicroBlaze para la versión 10.1 de EDK es la 7.10.a, como muestra la Figura 4.6.

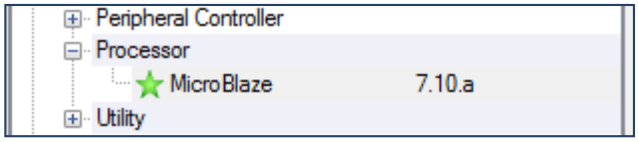

Figura 4. 6 Procesador MicroBlaze en IP Catalog de EDK.

MicroBlaze Configuration Wizard (Asistente de configuración)

El asistente de configuración MicroBlaze ™ provee: [38]

- Plantilla basada en cuadro de diálogo de configuración, permitiendo la configuración de un solo clic.
- Estimaciones de área relativa de MicroBlaze, frecuencia y rendimiento, basada en las opciones que figuran en el cuadro de diálogo.
- Orientación a través del proceso de configuración.
- *Tips* para opciones de configuración para comprender el efecto de cada opción.
- Acceso directo a todas las opciones de interfaz con pestañas a través del botón de Avanzado (*Advanced button*).

El asistente de configuración MicroBlaze tiene las siguientes páginas asistentes:

- **Asistente de configuración**: Primera página que muestra la selección de plantilla y la configuración general.
- **Rendimiento e instrucciones**: Selección de las unidades de ejecución. Siempre se muestra.
- **Excepciones**: Excepciones para habilitar. Se muestra si las excepciones se seleccionaron en la primera página.
- **Depuración**: Número de puntos de interrupción y puntos de observación. Se muestra si la depuración está habilitada en primera página.
- **Cache**: Configuración de caché. Se muestra de haber selección en primera página.
- **MMU**: Ajustes MMU. Se muestra si se selecciona la gestión de memoria en la primera página.
- **PVR y Buses**: Registro de versión del procesador (*Processor Version Register*) y ajustes del bus. Última página, siempre se muestra.

La Tabla 4.1 muestra características que son configurables por versión de Microblaze.

Tabla 4. 1 Características configurables por versión de MicroBlaze. [15]

Feature	***MicroBlaze Versions***				
	v4.00	***v5.00***	***v6.00***	***v7.00***	***v7.10***
Version Status	*deprecated*	*deprecated*	*deprecated*	*deprecated*	*preferred*
Processor pipeline depth	*3*	*5*	*3/5*	*3/5*	*3/5*
On-chip Peripheral Bus (OPB) data side interface	*option*	*option*	*option*	*option*	*option*
On-chip Peripheral Bus (OPB) instruction side interface	*option*	*option*	*option*	*option*	*option*
Local Memory Bus (LMB) data side interface	*option*	*option*	*option*	*option*	*option*
Local Memory Bus (LMB) instruction side interface	*option*	*option*	*option*	*option*	*option*
Hardware barrel shifter	*option*	*option*	*option*	*option*	*option*
Hardware divider	*option*	*option*	*option*	*option*	*option*
Hardware debug logic	*option*	*option*	*option*	*option*	*option*
Fast Simplex Link (FSL) interfaces	*0-7*	*0-7*	*0-7*	*0-15*	*0-15*
Machine status set and clear instructions	*option*	*Yes*	*option*	*option*	*option*
Instruction cache over IOPB interface	*option*	*No*	*No*	*No*	*No*
Data cache over IOPB interface	*option*	*No*	*No*	*No*	*No*
Instruction cache over CacheLink (IXCL) interface	*option*	*option*	*option*	*option*	*option*
Data cache over CacheLink (DXCL) interface	*option*	*option*	*option*	*option*	*option*
4 or 8-word cache line on XCL	*4*	*option*	*option*	*option*	*option*
Hardware exception support	*option*	*option*	*option*	*option*	*option*
Pattern compare instructions	*option*	*Yes*	*option*	*option*	*option*
Floating point unit (FPU)	*option*	*option*	*option*	*option*	*option*
*Disable hardware multiplier**1***	*option*	*option*	*option*	*option*	*option*
Hardware debug readable ESR and EAR	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
Processor Version Register (PVR)	*-*	*option*	*option*	*option*	*option*
Area or speed optimized	*-*	*-*	*option*	*option*	*option*
Hardware multiplier 64-bit result	*-*	*-*	*option*	*option*	*option*
LUT cache memory	*-*	*-*	*option*	*option*	*option*
Processor Local Bus (PLB) data side interface	*-*	*-*	*-*	*option*	*option*
Processor Local Bus (PLB) instruction side interface	*-*	*-*	*-*	*option*	*option*
Floating point conversion and square root instructions	*-*	*-*	*-*	*option*	*option*
Memory Management Unit (MMU)	*-*	*-*	*-*	*option*	*option*
Extended Fast Simplex Link (FSL) instructions	*-*	*-*	*-*	*option*	*option*
Use Xilinx Cache Link for All I-Cache Memory Accesses	*-*	*-*	*-*	*-*	*option*
Use Xilinx Cache Link for All D-Cache Memory Accesses	*-*	*-*	*-*	*-*	*option*

1. Usado en Virtex™-2Pro y familias posteriores, para ahorrar bloques MUL18 y DSP48.

El sistema diseñado no incluye el bloque *debug*, tampoco se ha habilitado la unidad de punto flotante, esto se ha considerado con el propósito de ahorrar bloques lógicos del FPGA usado. El tamaño de memoria seleccionada es igual a 32KB, esta memoria es interna al FPGA por lo que consumirá cierta cantidad de los bloques disponibles para este propósito.

A continuación se muestra la ventana de resumen en la Figura 4.7, esta ventana indica todos los bloques usados en el sistema hasta ahora, ya que en lo posterior es posible añadir y crear más bloques que se conectarán al procesador Microblaze, vía el Bus PLB.

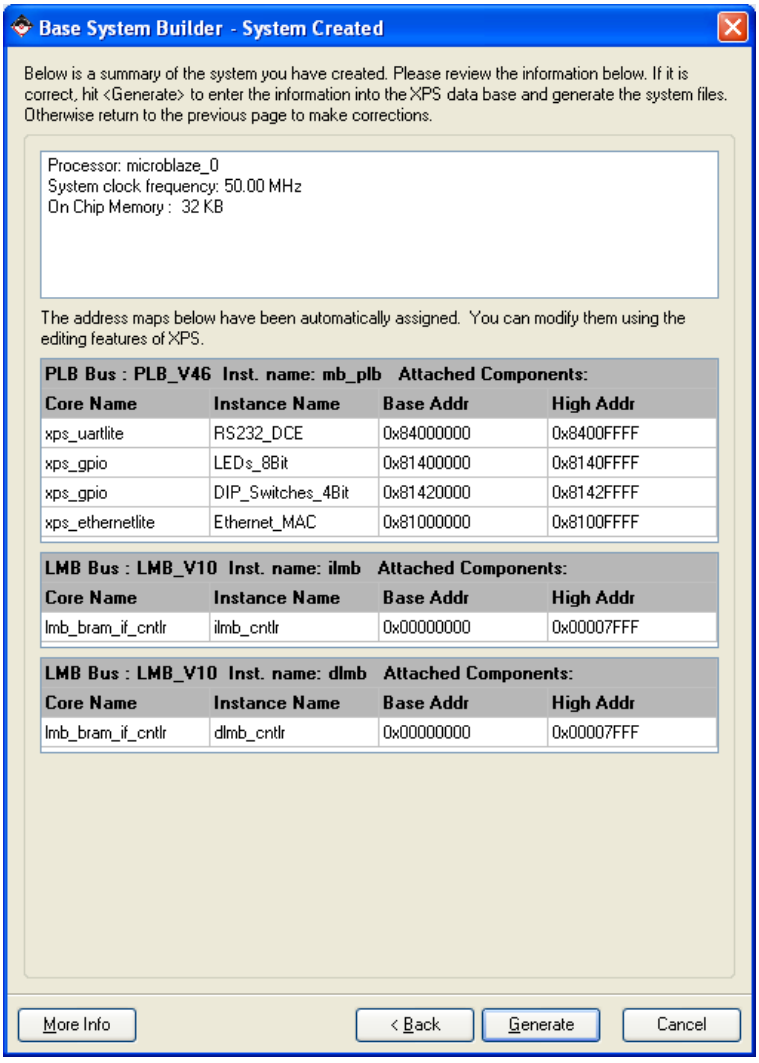

Figura 4. 7 Sistema creado.

Luego de generar nuestro sistema creado, la plataforma XPS tendrá la apariencia que se muestra en la Figura 4.8.

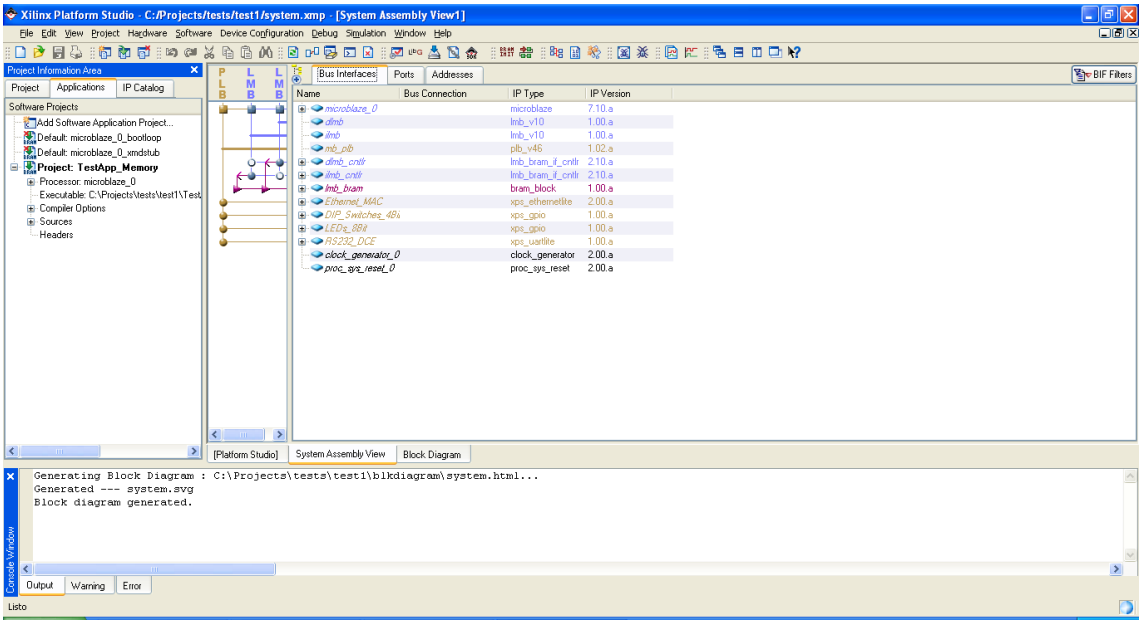

Figura 4. 8 Sistema hardware generado.

4.1.2 Módulo Ethernet

La tarjeta de desarrollo Spartan 3E Starter Kit Board cuenta con una Interface Ethernet, Standard Microsystems LAN83C185 10/100, además de ésta Interfaz Ethernet de la capa física (PHY), también cuenta con un conector RJ-45, como se muestra en la Figura 4.7. Con una Ethernet Media Access Controller (MAC) implementado en la FPGA, la tarjeta de desarrollo se puede conectar opcionalmente a una red Ethernet estándar. Toda temporización se controla desde un oscilador de cristal de 25 MHz que se encuentra en la tarjeta (on-board).

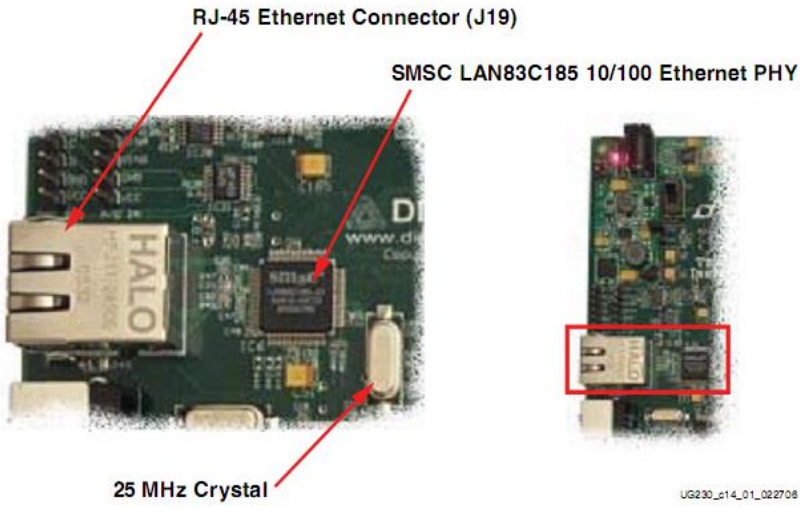

Figura 4. 9 Interface Ethernet PHY 10/100 con Conector RJ-45.

Las conexiones entre FPGA y LAN83C185 Ethernet PHY se realizan vía MII (*Media Independent Interface*), como es mostrado en la Figura 4.8.

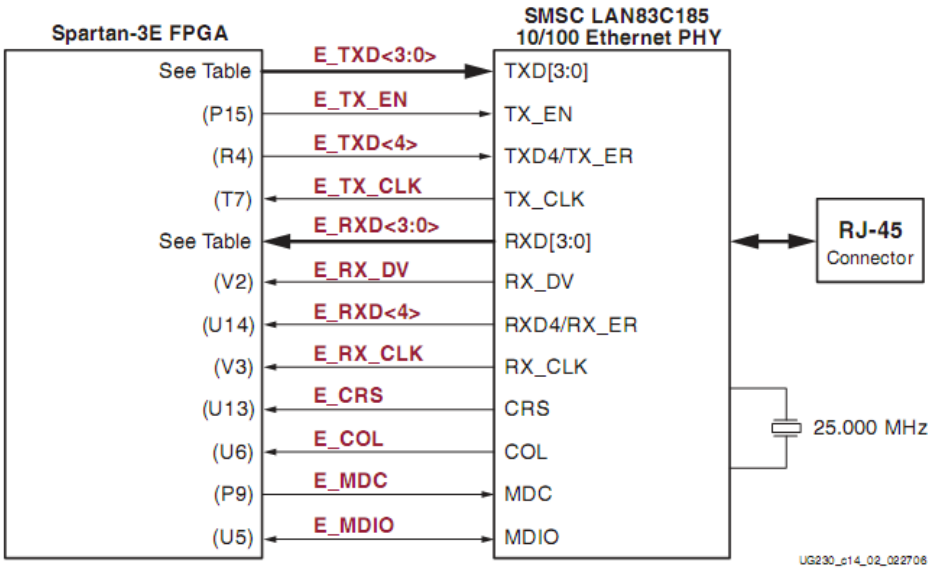

Figura 4. 10 Conexiones del FPGA a la interface Ethernet vía MII.

Una descripción más detallada al respecto de las conexiones realizadas es mostrada en la Tabla 4.2, en ella también se indica la nomenclatura (letra y numeración) de los pines del FPGA.

Tabla 4. 2 Conexiones entre FPGA y LAN83C185 Ethernet PHY vía MII. [24]

Signal Name	***FPGA Pin Number***	***Function***
E_TXD<4>	R6	*Transmit Data to the PHY. E_TXD<4> is also the MII Transmit Error.*
E_TXD<3>	T5	
E_TXD<2>	R5	
E_TXD<1>	T15	
E_TXD<0>	R11	
E_TX_EN	P15	*Transmit Enable.*
E_TX_CLK	T7	*Transmit Clock. 25 MHz in 100Base-TX mode, and 2.5 MHz in 10Base-T mode.*
E_RXD<4>	U14	*Receive Data from PHY.*
E_RXD<3>	V14	
E_RXD<2>	U11	
E_RXD<1>	T11	
E_RXD<0>	V8	
E_RX_DV	V2	*Receive Data Valid.*
E_RX_CLK	V3	*Receive Clock. 25 MHz in 100Base-TX mode, and 2.5 MHz in 10Base-T mode.*
E_CRS	U13	*Carrier Sense.*
E_COL	U6	*MII Collision Detect.*
E_MDC	P9	*Management Clock. Serial management clock.*
E_MDIO	U5	*Management Data Input/Output.*

La Ethernet PHY está principalmente destinada para su uso con aplicaciones MicroBlaze. Es por ello que una Ethernet MAC, en las versiones Full y Lite, como se muestra en la Figura 4.9, es parte de la Plataforma EDK, la cual puede ser insertada al sistema hardware mediante el Base System Builder.

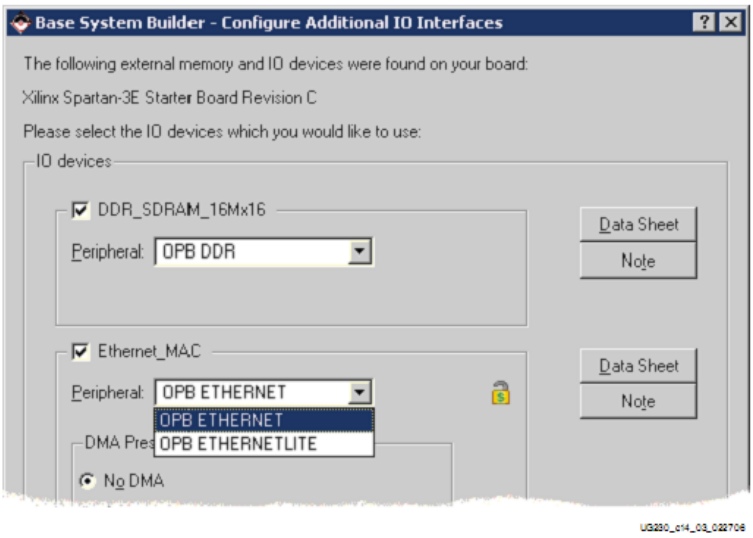

Figura 4. 11 Ethernet MAC IP Core para la Spartan 3E Starter Kit Board

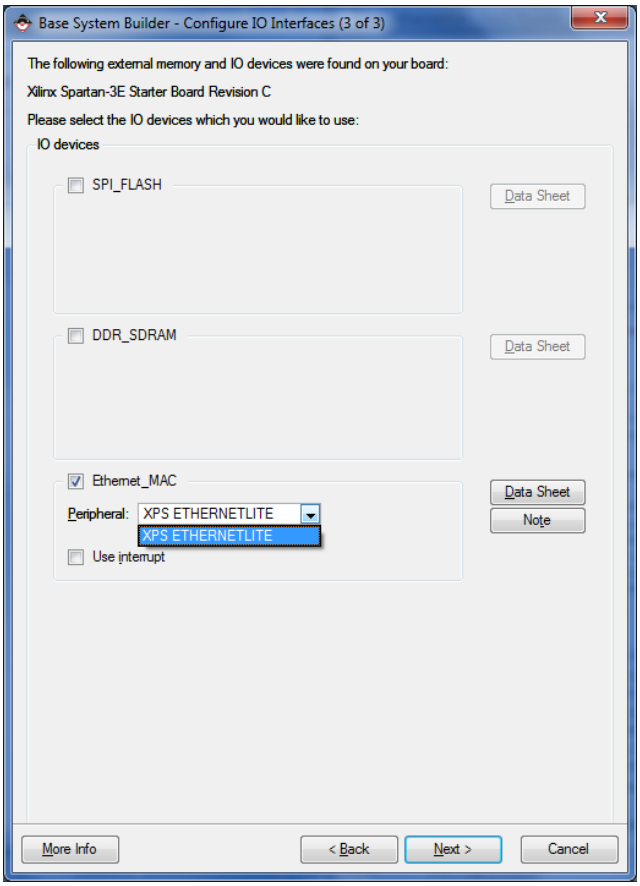

Figura 4. 12 Ethernet MAC IP Core en versión 10.1 de EDK

La ventana que se muestra en la Figura 4.11 es de la versión 8.1i de EDK. La versión Lite usa menos elementos lógicos del FPGA y está diseñado para aplicaciones que no requieren soporte de interrupciones. Sin embargo como se puede apreciar en esta versión 8.1 de EDK el símbolo del candado denota que es una IP con limitaciones, es por ello que trabajamos con la versión 10.1, la cual mostramos en las Figuras 4.12 y 4.13.

Como se puede apreciar en la versión 10.1 sólo se cuenta con la IP Ethernet en versión Lite.

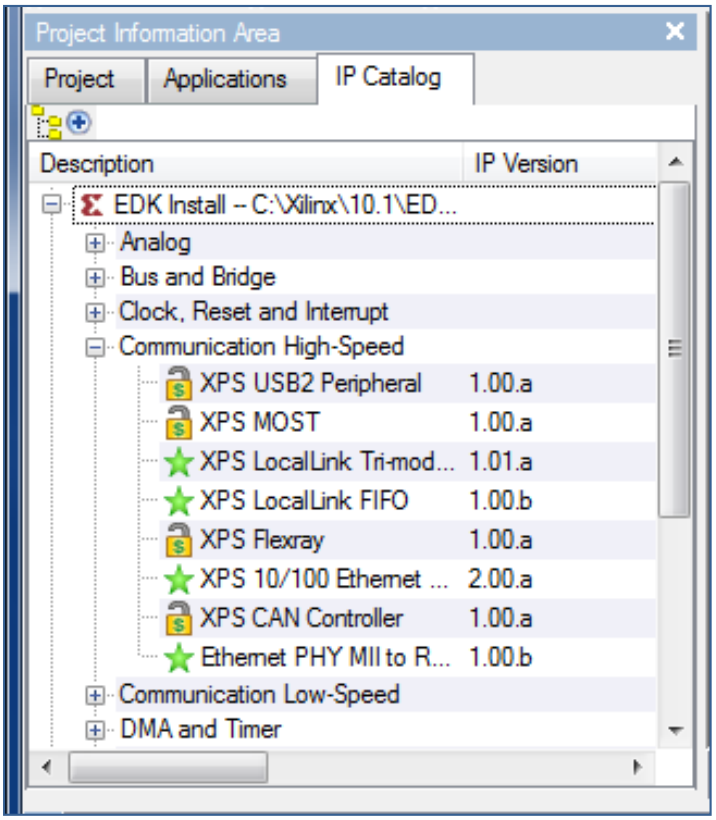

Figura 4. 13 IP Catalog de la versión 10.1 de EDK.

La IP core Ethernet MAC requiere restricciones de diseño para lograr el rendimiento requerido, mayor información al respecto puede encontrarse en el datasheet de la IP, el cual puede ser visto al dar clic en el botón datasheet, como puede observarse en la Figura 4.11 y 4.12.

Para hacer uso de esta interface se hace el uso de una IP core brindada por Xilinx, , la Ethernet Lite MAC (Media Access Controller), la cual se muestra en Figura 4.13 y debe ser manejada vía procesador Microblaze y que es conectada a éste, vía bus PLB, tal como se muestra en Figura 4.14.

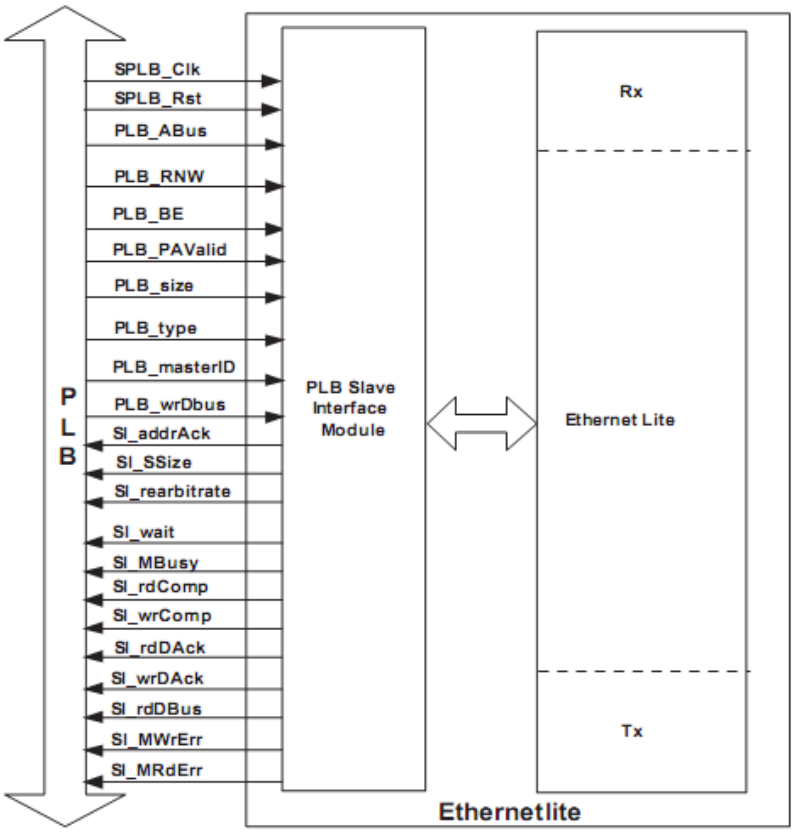

Figura 4. 14 Diagrama de bloques de la XPS Ethernet Lite.

NOTA: Para poder trabajar sin problema con el módulo Ethernet se requiere garantizar una frecuencia de trabajo. El asistente de diseño de EDK nos muestra una advertencia sobre esto, la cual se muestra en la Figura 4.15.

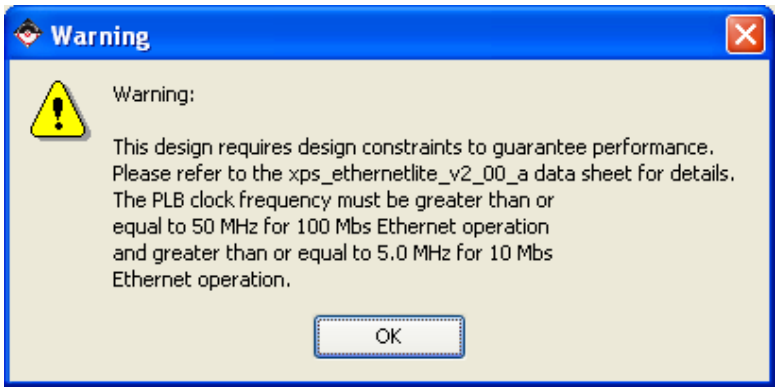

Figura 4. 15 Advertencia respecto a la frecuencia de trabajo de la IP Ethernet.

4.1.3 Módulo ADC [24]

La tarjeta Spartan 3E Starter Board incluye un Circuito de Captura Analógica con 2 canales, el cual consiste de un pre-amplificador de escala programable y un convertidor de analógico a digital (ADC), tal como se muestra en la **Figura 4.16**. Las entradas son suministradas mediante la cabecera J7.

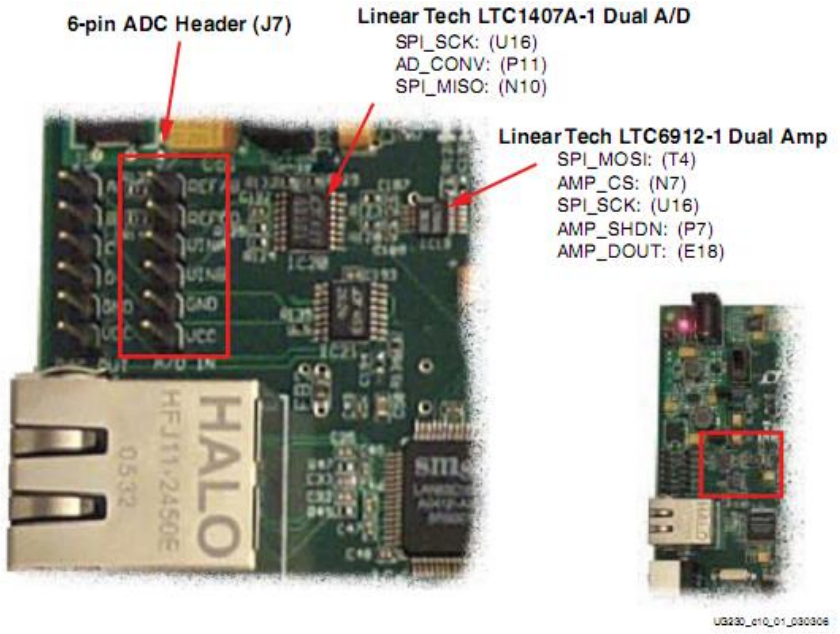

Figura 4. 16 Circuito de Captura Analógica de 2 canales.

El circuito de captura consiste de un pre-amplificador programable, Linear Technology LTC6912-1 Dual Amp [39], el cual escala la señal analógica de entrada en la cabecera J7, como puede observarse en la Figura 4.17.

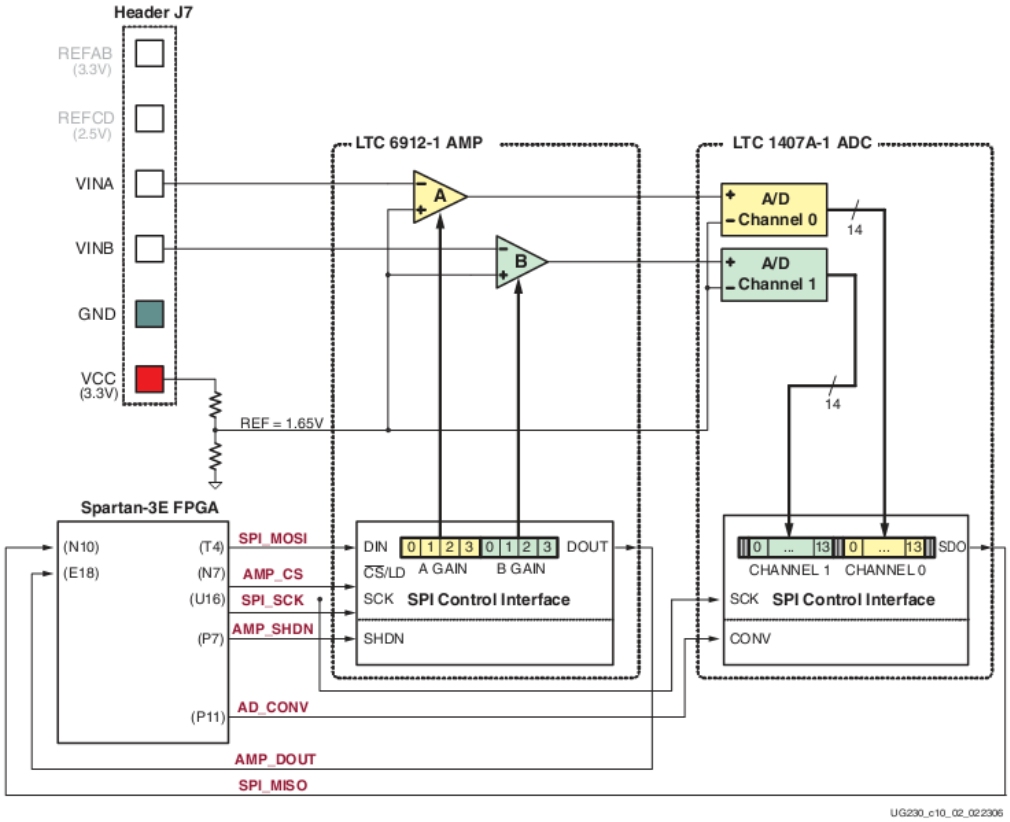

Figura 4. 17 Vista detallada del Circuito de Captura Analógica.

Las salidas de este pre-amplificador son conectadas al convertidor ADC, Linear Technology LTC1407A-1 Dual A/D [40]. Ambos, el pre-amplificador y el convertidor ADC son serialmente programados o controlados por el FPGA.

Salidas Digitales desde las Entradas Analógicas:

El circuito de captura toma los voltajes analógicos en VINA ó VINB, y los convierte a una representación digital de 14bits, D[13:0], como es expresado en la Ecuación (4.1).

Data representada en 14bits del convertidor ADC:

$$\mathrm{D}[13:0] = \mathrm{GAIN} \times \frac{(\mathrm{V_{IN}} - 1.65\mathrm{V})}{1.25\mathrm{V}} \times 8192 \qquad (4.1)$$

El parámetro GAIN es el valor de la configuración actual cargada para la ganancia, dentro del pre-amplificador programable. Los varios valores permitidos para GAIN y voltajes permitidos para las entradas VINA y VINB se muestran en la Tabla 4.4.

El voltaje de referencia para el amplificador y el ADC es 1.65V, el cual es generado vía divisor de voltaje, tal como se muestra en la Figura 4.17. Por consiguiente, 1.65V es sustraído de la entrada de voltaje en VINA ó VINB.

El rango máximo del ADC es ±1.25V, centrado alrededor del voltaje de referencia, 1.65V. Por lo tanto, 1.25V aparece en el denominador para escalar la entrada analógica como corresponde.

Finalmente, el ADC presenta 14bits, salidas digitales complemento a dos. Estos 14bits, en complemento a dos representan valores entre -2^{13} y $(2^{13}-1)$. Por lo tanto, la cantidad es escalada por 8192, ó 2^{13}.

Preamplificador Programable (AMP)

El LTC6912-1 provee 2 amplificadores inversores independientes con ganancia programable. El propósito del amplificador es escalar la entrada de voltaje en VINA o VINB de modo que maximiza el rango de conversión del DAC, es decir, 1.65 ± 1.25 V.

- Interface

La **Tabla 4.3** Enlista las señales de interface entre el FPGA y el amplificador. Las señales SPI_MOSI, SPI_MISO, y el SPI_SCK son compartidas con otros dispositivos en el bus. La señal AMP_CS es la señal de selección de esclavo en bajo (active-Low, activo en bajo) para el amplificador.

Tabla 4. 3 Señales de Interface del AMP

Signal	***FPGA Pin***	***Direction***	***Description***
SPI_MOSI	T4	FPGA ➔ AD	*Data Serial: Master Output, Slave Input. Presents 8-bit programmable gain settings, as defined in* ***Tabla 4.4.***
AMP_CS	N7	FPGA ➔ AMP	*Active-Low chip-select. The amplifier gain is set when signal returns High.*
SPI_SCK	U16	FPGA ➔ AMP	*Clock*
AMP_SHDN	P7	FPGA ➔AMP	*Active-High shutdown, reset.*
AMP_DOUT	E18	FPGA ➔ AMP	*Serial data. Echoes previous amplifier gain settings. Can be ignored in most applications.*

- Ganancia Programable

Cada canal analógico tiene un amplificador de ganancia programable asociado, como se muestra en la Figura 4.17. Las señales analógicas presentadas en las entradas VINA o VINB en la cabecera J7 son amplificadas relativas a 1.65V. La referencia 1.65 es generada usando un divisor de voltaje de la fuente 3.3V.

La ganancia de cada amplificador es programable de -1 a -100, como se muestra en la Tabla 4.4.

Tabla 4. 4 Ajustes de la Ganancia Programable (*Programmable Gain*).

Gain	**A3**	**A2**	**A1**	**A0**	***Input Voltage Range***	
	B3	**B2**	**B1**	**B0**	***Minimum***	***Maximum***
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	17.125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

- Interface de Control SPI

La Figura 4.18 pone de manifiesto la Interfaz de Comunicación basada en SPI (SPI-based). La ganancia de cada amplificador es enviada como una palabra comando de 8bits (8-bit command word), que consiste de dos campos de cuatro bits (4-bit fields). El bit más significativo, B3, es enviado primero.

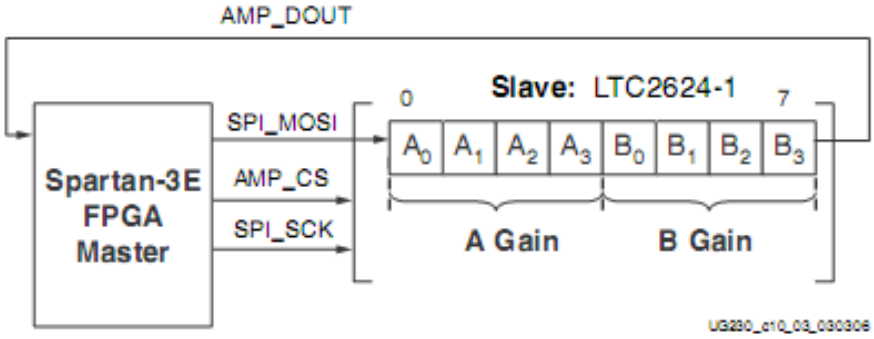

Figura 4. 18 Interface Serial SPI para amplificador.

La salida AMP_DOUT del amplificador, envía eco de los valores de ajuste de ganancia, tal como se muestra en la Figura 4.18. Estos valores pueden ser ignorados para la mayoría de aplicaciones.

La transacción del bus SPI inicia cuando el FPGA envía la señal AMP_CS a nivel bajo, como puede observarse en Figura 4.19. El amplificador captura datos seriales en la señal SPI_MOSI en el flanco de subida de la señal de reloj SPI_SCK. El amplificador presenta datos seriales en AMP_DOUT en el flanco de bajada de SPI_SCK.

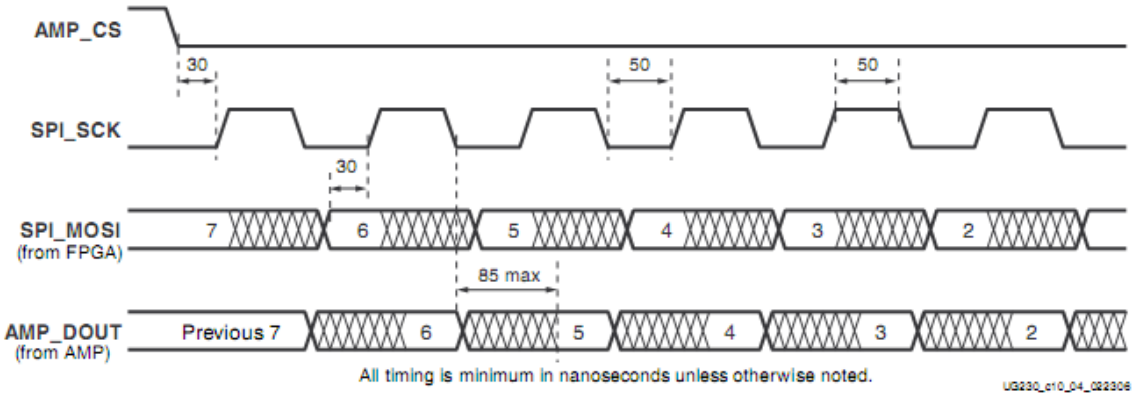

Figura 4. 19 Sincronización SPI cuando se comunica con amplificador

Convertidor de Analógico a Digital (ADC)

El LTC1407A-1 **[40]** provee dos convertidores de analógico a digital (ADC's). Ambas entradas analógicas son muestreadas simultáneamente cuando la señal AD_CONV es aplicada.

- Interface

La Tabla 4.5 enlista las señales de interface entre el FPGA y el ADC. Las señales SPI_MOSI, SPI_MISO, y SPI_SCK son compartidas con otros dispositivos en el bus SPI.

Tabla 4. 5 Señales de Interface ADC

Signal	***FPGA Pin***	***Direction***	***Description***
SPI_SCK	U16	FPGA ➔ ADC	*Clock.*
AD_CONV	P11	FPGA ➔ ADC	*Active-High shutdown and reset.*
SPI_MISO	N10	FPGA ➔ ADC	*Serial data: Master Input, Serial Output. Presents the digital representation of the sample analog values as two 14-bit two's complement binary values.*

- Interface de Control SPI

La Figura 4.20 provee un ejemplo de la transacción del bus SPI hacia el ADC. Cuando la señal AD_CONV toma un nivel alto, el ADC simultáneamente muestrea ambos canales. Los resultados de esta conversión no son presentadas hasta asegurar el siguiente pulso de la señal AD_CONV, una latencia de una muestra. La máxima razón para el muestreo es aproximadamente 1.5 MHz.

El ADC presenta la representación digital de los valores analógicos muestreados en 14 bits, como resultado del valor binario en complemento a dos.

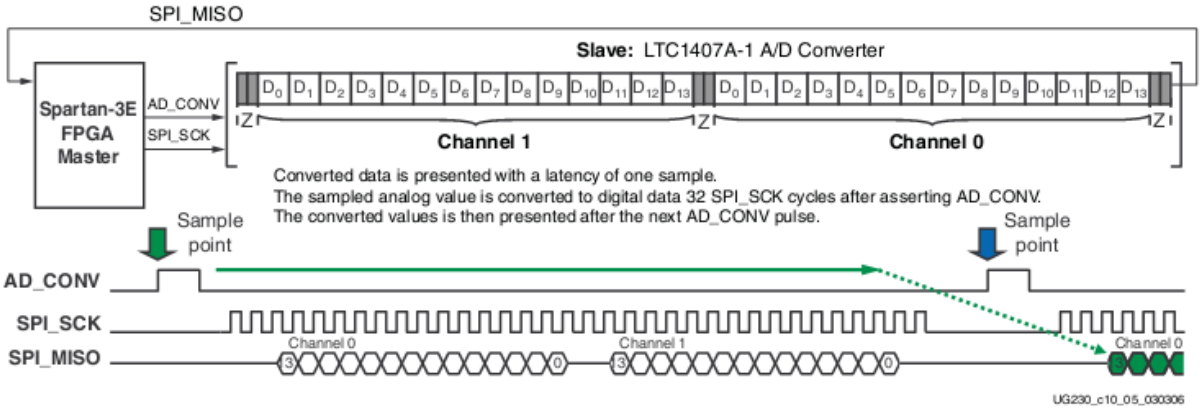

Figura 4. 20 Interface del Convertidor A/D y FPGA

La Figura 4.21 muestra detalladamente la transacción de temporización. La señal AD_CONV no es una señal SPI tradicional de selección del esclavo. Se debe asegurar el proporcionar suficientes ciclos de reloj SPI_SCK para que el ADC deje la señal

SPI_MISO en el estado de alta impedancia. De lo contrario, el ADC bloquearía la comunicación para los otros periféricos SPI. Como se muestra en la Figura 4.20, utilice una secuencia de comunicaciones de 34 ciclos. El ADC afirma en alta impedancia (High-Z) su salida de datos durante dos ciclos de reloj antes y después de cada transferencia de datos de 14 bits.

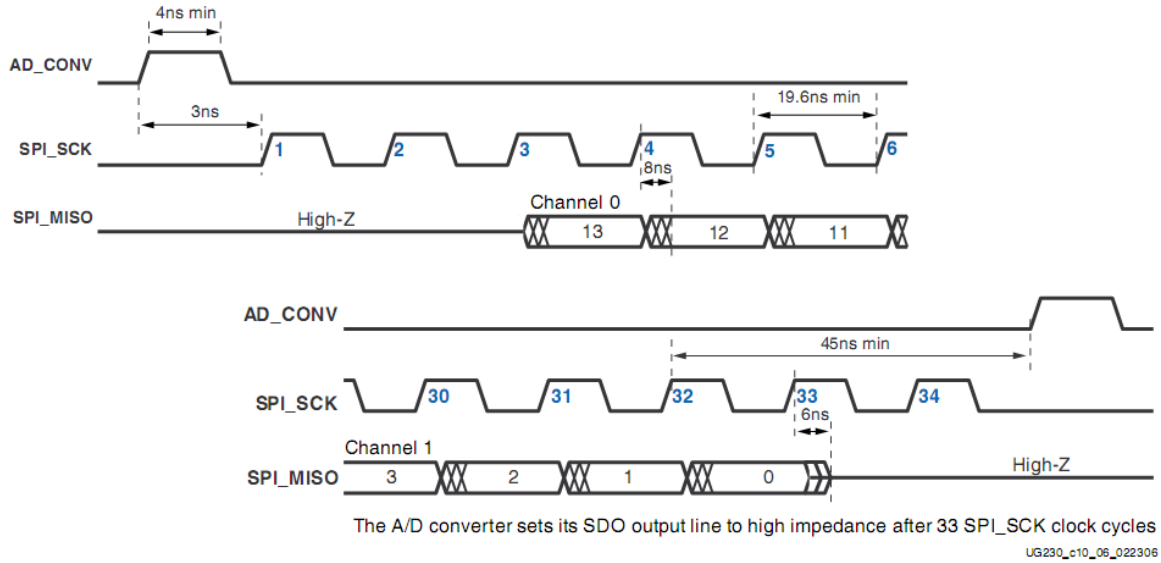

Figura 4. 21 Temporización SPI detallada para el ADC

Deshabilitar otros dispositivos en el bus SPI para evitar contención

Las señales del bus SPI son compartidas por otros dispositivos en la tarjeta. Es vital que otros dispositivos sean deshabilitados cuando el FPGA se comunica con el Amplificador (AMP) o Convertidor (ADC) para evitar contención o bloqueo del bus. La Tabla 4.6 provee las señales y los valores lógicos requeridos para deshabilitar los otros dispositivos. Aunque la memoria, StrataFlash PROM, es un dispositivo paralelo, su bit de datos menos significativo es compartido con la señal SPI_MISO.

Tabla 4. 6 Cuadro que indica los valores para deshabilitar dispositivos.

Signal	***Disabled Device***	***Disable Value***
SPI_SS_B	*SPI Serial Flash*	1
AMP_CS	*Programmable Pre-Amplifier*	1
DAC_CS	*DAC*	1
SF_CE0	*StrataFlash Parallel Flash PROM*	1
FPGA_INIT_B	*Platform Flash PROM*	1

Conectando Entradas Analógicas

Conectar señales AC a VINA o VINB vía un capacitor de bloqueo DC.

4.1.4 Módulo DAC [24]

Este mismo módulo de desarrollo incluye un Circuito Convertidor de Digital a Analógico (DAC), el cual es compatible con el protocolo SPI, y cuenta con 4 canales. Este dispositivo DAC, Linear Technology LTC2624 Quad DAC [41], presenta una resolución de 12bits sin signo. Las 4 salidas del DAC aparecen en la cabecera J5, tal como se muestra en la Figura 4.22.

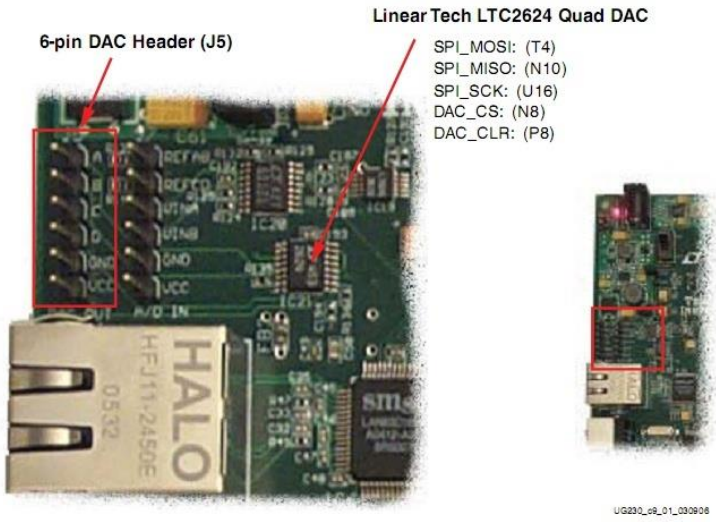

Figura 4. 22 Convertidor de Digital a Analógico y cabecera de conexiones.

Comunicación SPI

Como se muestra en la Figura 4.23, el FPGA usa una Interface Periférica Serial (SPI), para comunicar los valores digitales a cada uno de los cuatro canales del DAC.

El bus SPI es una interface que emplea 4 hilos, con canal orientado a caracteres, *full-duplex* y síncrona. Un maestro de bus, el FPGA en este caso, controla la señal de reloj del bus (SPI_SCK) y transmite data serial (SPI_MOSI) para el esclavo del bus seleccionado, el DAC en este caso. Al mismo tiempo, el esclavo del bus, o dispositivo esclavo más usualmente llamado, provee data serial de regreso al dispositivo maestro (SPI_MISO).

Señales de Interface

La Tabla 4.7 lista las señales de interface entre el FPGA y el DAC. Las señales SPI_MOSI, SPI_MISO, y SPI_SCK son compartidas con otros dispositivos en el bus SPI. La señal DAC_CS es la entrada de selección de esclavo, la cual es activa en baja (active-low), para seleccionar el DAC. La señal DAC_CLR es la entrada de reset asíncrono y es activa en baja para el DAC.

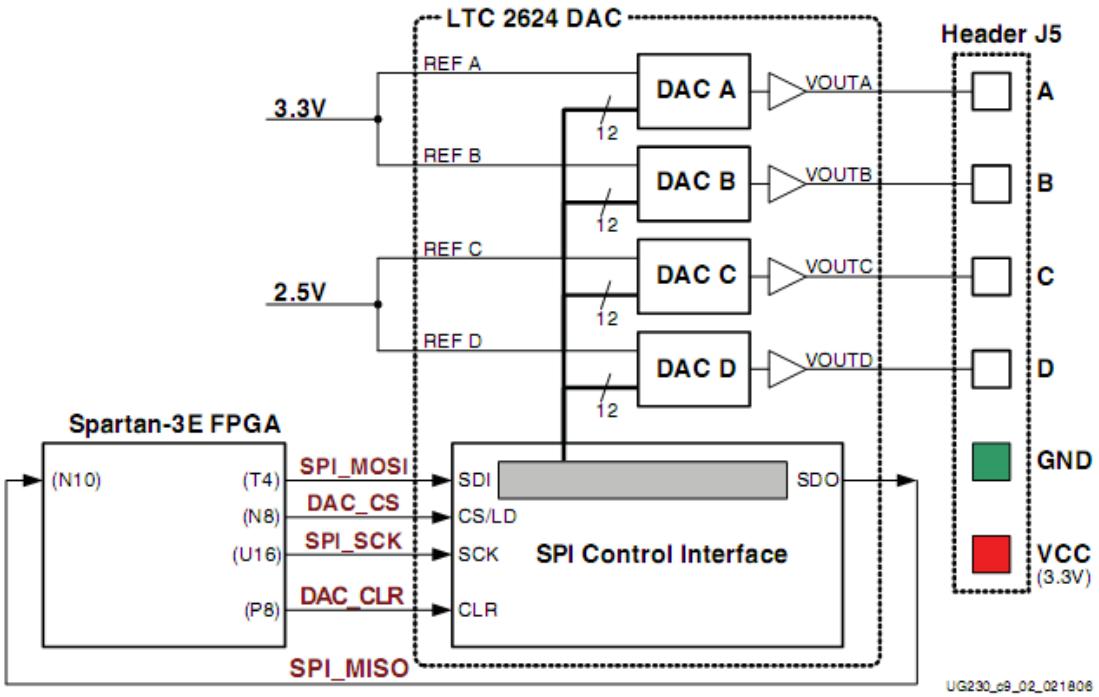

Figura 4. 23 Diagrama esquemático de las conexiones entre FPGA y DAC.

Tabla 4. 7 Señales de Interface DAC

Signal	***FPGA Pin***	***Direction***	***Description***
SPI_MOSI	T4	FPGA ➔ DAC	*Serial data: Master Output, Slave Input*
DAC_CS	N8	FPGA ➔ DAC	*Active-Low chip-select. Digital-to-analog conversion starts when signal returns High.*
SPI_SCK	U16	FPGA ➔ DAC	*Clock*
DAC_CLR	P8	FPGA ➔ DAC	*Asynchronous, active-Low reset input*
SPI_MISO	N10	FPGA ➔ DAC	*Serial data: Master Input, Slave Output*

La salida de datos en serie desde el DAC (SPI_MISO) se utiliza principalmente para conectar en cascada múltiples DAC's. Esta señal puede ser ignorada en muchas aplicaciones aunque ella nos demuestra la característica de comunicación full-duplex sobre el bus SPI.

Deshabilitar otros dispositivos del Bus SPI para evitar Contención

Las señales del bus SPI son compartidas por otros dispositivos en la tarjeta de desarrollo. Por ello es vital el deshabilitar otros dispositivos cuando el FPGA se comunica con el DAC, para así evitar contención del bus. La Tabla 4.8 provee las señales y los

valores lógicos requeridos para deshabilitar otros dispositivos. Aunque la StrataFlash PROM es un dispositivo paralelo, su bit de data menos significativo es compartido con la señal SPI_MISO.

Tabla 4. 8 Dispositivos deshabilitados en el Bus SPI

Signal	***Disabled Device***	***Disable Value***
SPI_SS_B	*SPI serial Flash*	1
AMP_CS	*Programmable pre-amplifier*	1
AD_CONV	*Analog-to-Digital Converter (ADC)*	0
SF_CE0	*StrataFlash Parallel Flash PROM*	1
FPGA_INIT_B	*Platform Flash PROM*	1

Detalles de la Comunicación SPI

La Figura 4.24 muestra un ejemplo detallado de la temporización del Bus SPI. Cada bit es transmitido o recibido con relación a señal de reloj SPI_SCK. El bus es completamente estático y soporta señales de reloj hasta un máximo de 50MHz. Sin embargo, se deben comprobar todos los parámetros de sincronización del datasheet del circuito LTC2624 Quad DAC para ver si se opera a velocidades cercanas a la velocidad máxima.

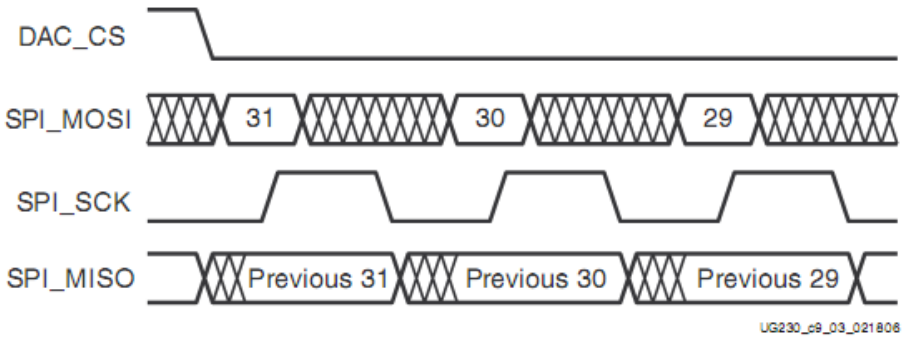

Figura 4. 24 Diagrama de tiempos de la comunicación SPI

Luego de asignarse un valor de nivel lógico bajo a la señal DAC_CS, la cual es active-low, el FPGA trasmite datos mediante la señal SPI_MOSI, primero el MSB (Bit más significativo). El LTC2624 captura la entrada de datos (SPI_MOSI) en el flanco de subida del reloj SPI_SCK; los datos deben ser validos durante al menos 4ns con respecto al flanco de subida del reloj.

El LTC2624 DAC trasmite sus datos mediante la señal SPI_MISO en el flanco de bajada del reloj SPI_SCK. El FPGA captura estos datos en el siguiente flanco de subida de la señal SPI_SCK. El FPGA debe leer el primer valor de SPI_MISO en el primer flanco de subida de la señal reloj SPI_SCK después que la señal DAC_CS va al nivel lógico bajo. De otra forma, el bit 32 sería perdido.

Luego de transmitir todos los 32 bits, el FPGA completa la transacción del Bus SPI retornando la señal seleccionadora de esclavo DAC_CS a un nivel lógico alto. El llevado de la señal a nivel alto inicia el proceso real de conversión de digital a analógico dentro del DAC.

Protocolo de Comunicación

La Figura 4.25 muestra el protocolo de comunicación requerido para la interface con el LTC2624 DAC. El DAC soporta los protocolos a 24bits y 32bits. El protocolo de 32bits es el mostrado.

Internamente al convertidor D/A, la interface SPI es formada por un registro de desplazamiento de 32bits (32-bit shift register). Cada palabra de comando de 32bits consiste de un comando, una dirección, seguido por el valor de los datos (data value). Cuando un nuevo comando entra en la DAC, la palabra de comando anterior de 32bits se repite de nuevo al maestro. La respuesta desde el DAC puede ser ignorada aunque esto es útil para confirmar correcta comunicación.

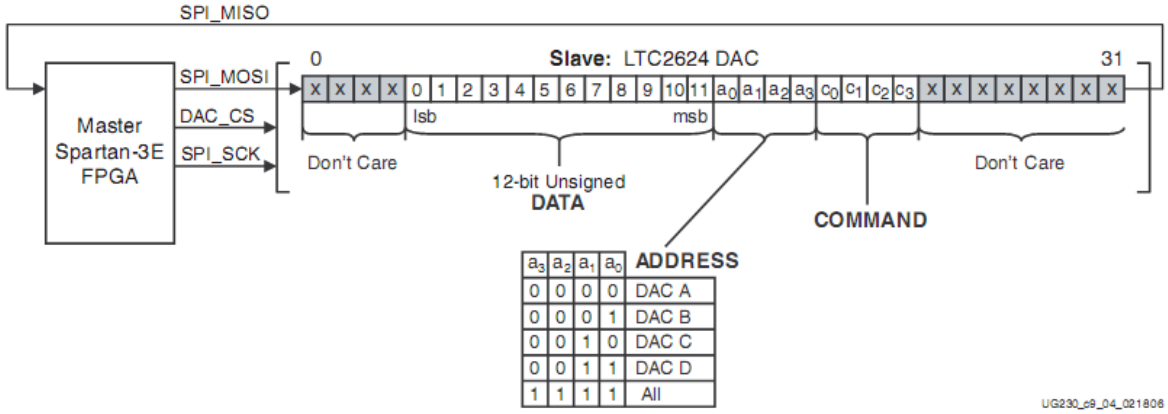

Figura 4. 25 Protocolo de Comunicación SPI para el LTC2624 DAC.

El FPGA primero envía 8bits "*dummy*" o "*don't care*" (bits de condiciones de no importa), seguido por un comando de 4bits. El comando que es más comúnmente usado es el COMMAND[3:0] = "0011", que inmediatamente actualiza la salida de DAC seleccionada con su específico valor de datos. Luego del comando, el FPGA selecciona uno o todos los canales de salida del DAC vía un campo de dirección (ADDRESS) de 4bits.

Posteriormente a este campo de dirección, el FPGA envía 12bits de datos, de valor sin signo, los cuales el DAC convertirá a un valor analógico en la salida o salidas seleccionadas. Finalmente, 4bits "dummy" o "don't care" (bits de condiciones de no importa) para completar la palabra de comando de 32bits.

Especificación del voltaje de salida del DAC

Como se muestra en la Figura 4.23, cada nivel de la salida DAC es el equivalente analógico de un valor digital sin signo de 12bits, D[11:0], escrito por el FPGA para el DAC vía interface SPI.

El voltaje en una salida específica es generalmente descrito en la Ecuación (4.2). El voltaje de referencia, $V_{REFERENCE}$, es diferente entre las cuatro salidas DAC. Los canales A y B usan 3.3V de voltaje de referencia, y canales C y D usan 2.5V. Los voltajes de referencia cuentan con un ±5% de tolerancia, así que podrá haber una correspondiente pequeña variación en la salida de voltaje, V_{OUT}.

$$\mathrm{V_{OUT}} = \frac{\mathrm{D[11:0]}}{4096} \times \mathrm{V_{REFERENCE}} \tag{4.2}$$

Salidas A y B:

$$\mathrm{V_{OUT}} = \frac{\mathrm{D[11:0]}}{4096} \times (3.3\mathrm{V} \pm 5\%) \tag{4.3}$$

Salidas C y D:

$$\mathrm{V_{OUT}} = \frac{\mathrm{D[11:0]}}{4096} \times (2.5\mathrm{V} \pm 5\%) \tag{4.4}$$

4.1.5 Módulo de Expansión de Conexiones

La tarjeta de desarrollo provee una variedad de expansión de conectores para fácil flexibilidad de interface con otros componentes externos a ella. La tarjeta incluye las siguientes I/O (Entrada/Salida) cabeceras de expansión (ver Figura 4.26):

- Un conector, Hirose 100-pin edge connector, con 43 pines de entrada y salida de usuario (user-I/O) asociados, incluyendo hasta 15 pares diferenciales LVDS I/O y dos pares de sólo entrada pares.

- Tres módulos de conexiones de periféricos de 6 pines (*6-pin Accesory Header*).

- Plataforma para sondas de analizadores lógicos (sin conector) Agilent o Tektronix.

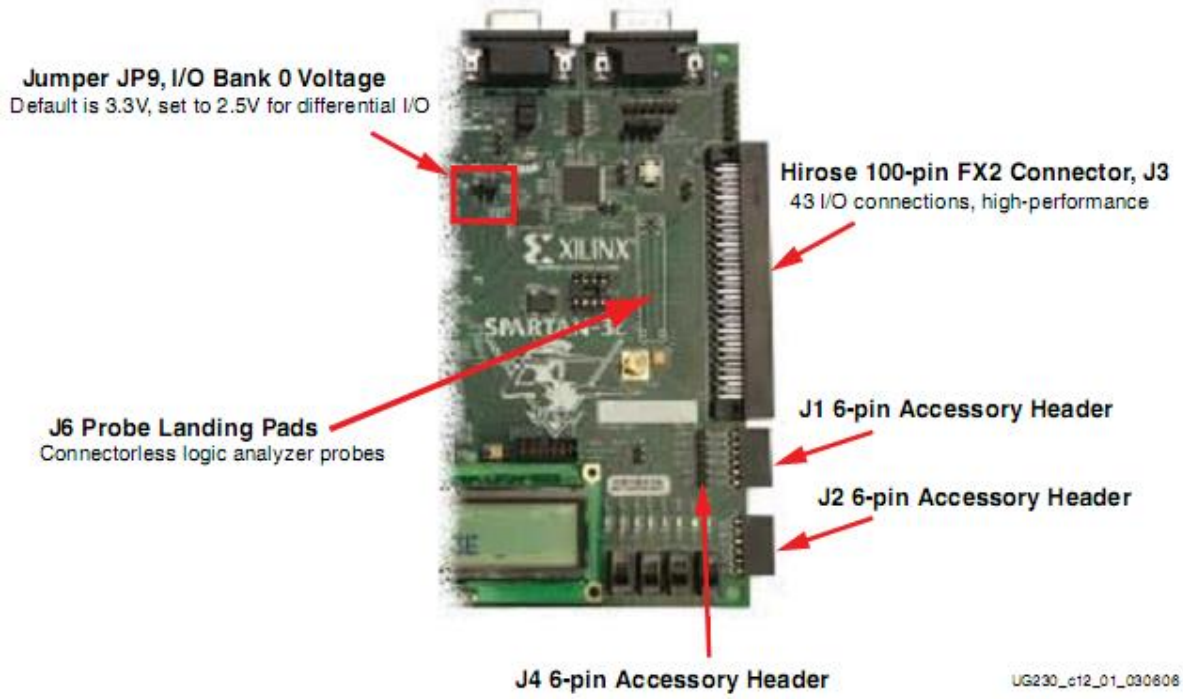

Figura 4. 26 Cabeceras de Expansión

Hirose 100-pin FX2 Edge Connector (J3)

Un conector de 100 pines es localizado a lo largo del borde derecho de la tarjeta de desarrollo (ver Figura 4.26). Este conector es un Hirose FX2-100P-1.27DS *header*, con 1.27mm de paso. Será llamado conector FX2 para mayor facilidad de aquí en adelante.

Como se muestra en la Figura 4.27, 43 pines I/O del FPGA interactúan con el conector FX2. Todos, excepto cinco de estos pines, son pines verdaderos, bidireccionales de I/O, capaces de conducción y recepción de señales. Cinco pines, FX2_IP<38:35> y FX2_IP<40> son pines de sólo entrada en el FPGA. Estos pines son resaltados en color verde en la Tabla 4.9 y no pueden conducir señales al conector FX2, pero pueden recibir señales.

Tres señales son principalmente reservadas como señales de reloj entre la tarjeta de desarrollo y el conector FX2, aunque los tres se conectan al *full I/O pins*.

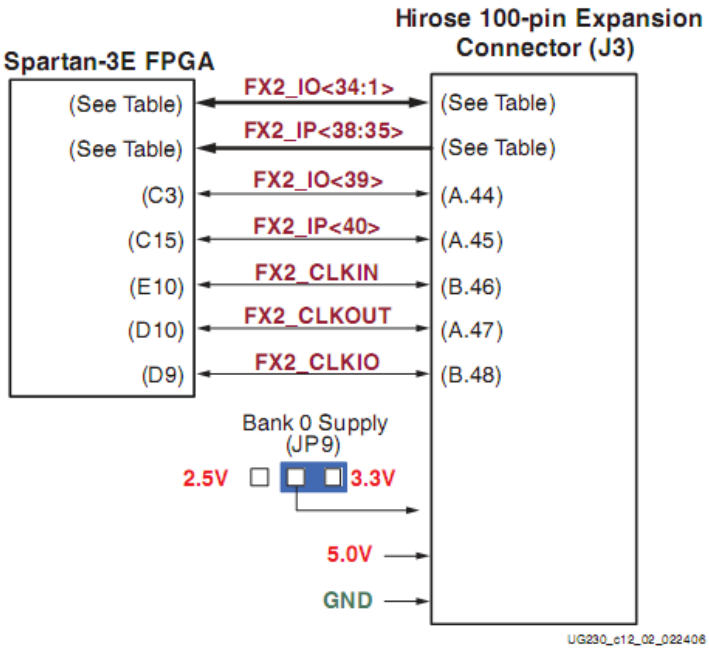

Figura 4. 27 Conexiones entre FPGA e Hirose 100-pin Edge Connector

Fuentes de Voltaje al Conector

La tarjeta de desarrollo provee potencia al conector Hirose 100-pin FX y a cualquier tarjeta adjunta vía dos fuentes (Ver Figura 4.27). La fuente de 5.0V provee una fuente de voltaje para cualquier lógica de 5V en la tarjeta adjunta o para alternadamente proveer potencia a cualquier regulador de voltaje en la tarjeta adjunta.

Una fuente separada provee el mismo voltaje que el aplicado en las E/S (Entradas/Salidas) del Banco 0 del FPGA (*FPGA's I/O Bank 0*). Todas las E/S del FPGA que interactúan con el conector Hirose están en el Banco 0. Las E/S del Banco 0 proveen 3.3V por defecto. Sin embargo, el nivel de voltaje puede ser cambiado a 2.5V usando un *jumper JP9.* Algunos estándares de E/S del FPGA, especialmente los estándares diferenciales, tales como RSDS y LVDS, requieren una salida de fuente de voltaje de 2.5V. Para soportar altas velocidades a través del conector, una mayoría de los pines en el lado B (B-side) del conector FX2 están vinculados a GND.

Conector Pinout y Conexiones FPGA

La Tabla 4.9 muestra el pinout (*a diagram showing the arrangement of pins on an integrated circuit and their functions*) del conector FX2 y las conexiones de los pines del FPGA asociados. El conector FX2 tiene dos filas de conectores, cada una con 50 conexiones, mostradas en la tabla usando resaltado en color amarillo. También resalta las conexiones que son compartidas con los 8 LED's discretos, los tres 6-pin accesorios de cabecera (*Accesory Headers*, J1, J2 y J4), y la cabecera de depuración sin conectores (J6).

Tabla 4. 9 *Hirose 100-pin FX2 Connector Pinout and FPGA Connections (J3)*

Signal Name	*FPGA Pin*	*Shared Header Connections*					*FX2 Connector*		*FPGA Pin*	*Signal Name*
		LED	*J1*	*J2*	*JP4*	*J6*	*A (top)*	*B (bottom)*		
	VCCO_0						1	1		SHIELD
	VCCO_0						2	2	GND	GND
TMS_B							3	3		TDO_XC2C
JTSEL							4	4		TCK_B
TDO_FX2							5	5	GND	GND
FX2_IO1	B4		♦			♦	6	6	GND	GND
FX2_IO2	A4		♦			♦	7	7	GND	GND
FX2_IO3	D5		♦			♦	8	8	GND	GND
FX2_IO4	C5		♦			♦	9	9	GND	GND
FX2_IO5	A6			♦		♦	10	10	GND	GND
FX2_IO6	B6			♦		♦	11	11	GND	GND
FX2_IO7	E7			♦		♦	12	12	GND	GND
FX2_IO8	F7			♦		♦	13	13	GND	GND
FX2_IO9	D7				♦	♦	14	14	GND	GND
FX2_IO10	C7				♦	♦	15	15	GND	GND
FX2_IO11	F8				♦	♦	16	16	GND	GND
FX2_IO12	E8				♦	♦	17	17	GND	GND
FX2_IO13	F9	LD7				♦	18	18	GND	GND
FX2_IO14	E9	LD6				♦	19	19	GND	GND
FX2_IO15	D11	LD5				♦	20	20	GND	GND
FX2_IO16	C11	LD4				♦	21	21	GND	GND
FX2_IO17	F11	LD3				♦	22	22	GND	GND
FX2_IO18	E11	LD2				♦	23	23	GND	GND
FX2_IO19	E12	LD1					24	24	GND	GND
FX2_IO20	F12	LD0					25	25	GND	GND
FX2_IO21	A13						26	26	GND	GND
FX2_IO22	B13						27	27	GND	GND
FX2_IO23	A14						28	28	GND	GND
FX2_IO24	B14						29	29	GND	GND
FX2_IO25	C14						30	30	GND	GND
FX2_IO26	D14						31	31	GND	GND
FX2_IO27	A16						32	32	GND	GND
FX2_IO28	B16						33	33	GND	GND
FX2_IO29	E13						34	34	GND	GND
FX2_IO30	C4						35	35	GND	GND
FX2_IO31	B11						36	36	GND	GND
FX2_IO32	A11						37	37	GND	GND
FX2_IO33	A8						38	38	GND	GND
FX2_IO34	G9						39	39	GND	GND
FX2_IO35	D12						40	40	GND	GND
FX2_IO36	C12						41	41	GND	GND
FX2_IO37	A15						42	42	GND	GND
FX2_IO38	B15						43	43	GND	GND
FX2_IO39	C3						44	44	GND	GND
FX2_IO40	C15						45	45	GND	GND
GND	GND						46	46	E10	FX2_CLKIN
FX2_CLKOUT	D10						47	47	GND	GND
GND	GND						48	48	D9	FX2_CLKIO
5.0V							49	49		5.0V
5.0V							50	50		SHIELD

E/S Diferenciales

El conector FX2, cabecer J3, suporta hasta 15 pares Entradas/Salidas (E/S) diferenciales (*Differential I/O pairs*) y dos pares solo entrada (*input-only*), ya sea usando la normas LVDS o RSDS de E/S, como se indica en la Tabla 4.10. Todos los pares de E/S soportan terminación de entrada diferencial (DIFF-TERM) como es descrito en la ficha de datos (*Datasheet)* de la Spartan 3E.

Tabla 4. 10 Differential I/O Pairs

Differential Pair	*Signal Name*	*FPGA Pins*	*FPGA Pin Name*	*Direction*	*DIFF_TERM*	*External Resistor Designator*
1	FX2_IO1	B4	IO_L24N_0	I/O	Yes	
	FX2_IO2	A4	IO_L24P_0	I/O	Yes	
2	FX2_IO3	D5	IO_L23N_0	I/O	Yes	
	FX2_IO4	C5	IO_L23P_0	I/O	Yes	
3	FX2_IO5	A6	IO_L20N_0	I/O	Yes	
	FX2_IO6	B6	IO_L20P_0	I/O	Yes	
4	FX2_IO7	E7	IO_L19N_0	I/O	Yes	
	FX2_IO8	F7	IO_L19P_0	I/O	Yes	
5	FX2_IO9	D7	IO_L18N_0	I/O	Yes	
	FX2_IO10	C7	IO_L18P_0	I/O	Yes	
6	FX2_IO11	F8	IO_L17N_0	I/O	Yes	
	FX2_IO12	E8	IO_L17P_0	I/O	Yes	
7	FX2_IO13	F9	IP_L15N_0	I/O	Yes	
	FX2_IO14	E9	IP_L15P_0	I/O	Yes	
8	FX2_IO15	D11	IP_L09N_0	I/O	Yes	
	FX2_IO16	C11	IP_L09P_0	I/O	Yes	
9	FX2_IO17	F11	IO_L08N_0	I/O	Yes	R202
	FX2_IO18	E11	IO_L08P_0	I/O	Yes	
10	FX2_IO19	E12	IO_L06N_0	I/O	Yes	R203
	FX2_IO20	F12	IO_L06P_0	I/O	Yes	
11	FX2_IO21	A13	IO_L05P_0	I/O	Yes	R204
	FX2_IO22	B13	IO_L05N_0	I/O	Yes	
12	FX2_IO23	A14	IO_L04N_0	I/O	Yes	R205
	FX2_IO24	B14	IO_L04P_0	I/O	Yes	
13	FX2_IO25	C14	IO_L03N_0	I/O	Yes	R206
	FX2_IO26	D14	IO_L03P_0	I/O	Yes	
14	FX2_IO27	A16	IO_L01N_0	I/O	Yes	R207
	FX2_IO28	B16	IO_L01P_0	I/O	Yes	
15	FX2_IO35	D12	IP_L07N_0	Input		R208
	FX2_IO36	C12	IP_L07P_0	Input		
16	FX2_IO37	A15	IP_L02N_0	Input		R209
	FX2_IP38	B15	IP_L02P_0	Input		
17	FX2_CLKIN	E10	IO_L11N_0/ GCLK5	I/O	Yes	R210
	FX2_CLKOUT	D10	IO_L11P_0/ GCLK4	I/O	Yes	

4.2 Diseño del Software del Sistema

El diseño Software está enfocado al programa que será ejecutado por el procesador Microblaze y a la Interface de Usuario requerida para el usuario a distancia.

Software del Procesador Microblaze

El diseño concierne al manejo de los módulos diseñados en la parte hardware vía instrucciones de procesador. Cada módulo cuenta con Software para ser manejado.

Características para manejo de la IP core Ethernet Lite MAC:

- Requiere programación con una dirección MAC.
- Proporciona buffers de transmisión y recepción para tramas Ethernet.
- Se conecta al Ethernet PHY mediante MII (pines reales del FPGA).
- Agrega el preámbulo y FCS (Frame Check Sequence) de la trama Ethernet(ver Figura 4.20)
- Gestiona el acceso a la red y proporciona detección de colisiones.

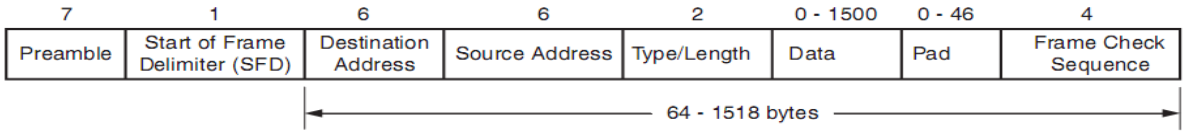

Figura 4. 28 Trama Ethernet (tamaño de campos en bytes)

Interface de Usuario

Se usará plataforma Java para el diseño y generación de ejecutable *.jar de la Interface de usuario prototipo (Figura 4.29), la cual debe cumplir las siguientes características:

- Contar con interface de ingreso de datos.
- Contar con interface de visualización de datos.
- Contar con interface de comunicación Ethernet que inicie envío y recepción luego de ingreso a sistema.

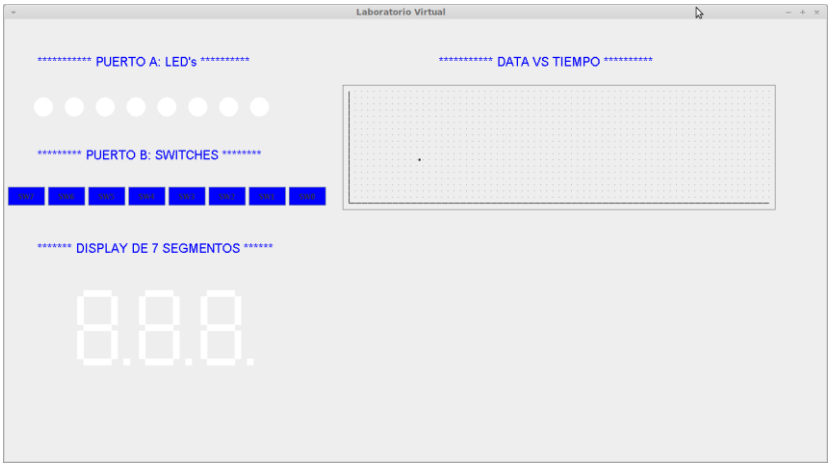

Figura 4. 29 Interface de Usuario.

- El desarrollo del software del procesador Microblaze es desarrollado en el SDK (Software Development Kit) aunque también es posible trabajar el software del procesador en XPS, tal vez en el futuro sea quitada esta posibilidad ya que así es mencionada en cierta documentación de Xilinx, con el fin de enfocar a SDK para ese objetivo.
- Adicionalmente también podemos mencionar que Xilinx cuenta con DSP Tools en donde tenemos a la herramienta Xilinx Systems Generator, ésta herramienta trabaja conjuntamente con Simulink de Matlab, lo cual permite trabajar bloques en Simulink que generarán código en HDL.

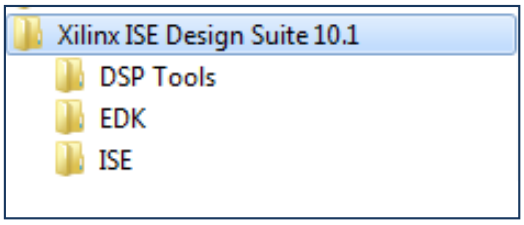

Figura 4. 30 Organización de herramientas CAD y EDA de Xilinx.

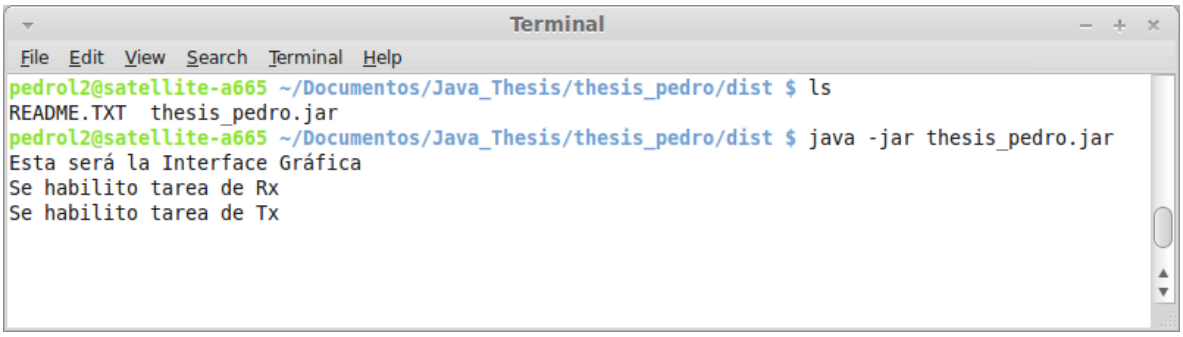

Figura 4. 31 Ejecución de archivo *.jar

4.2.1 Arquitectura General

A continuación se muestra un diagrama de la arquitectura general del Software del sistema:

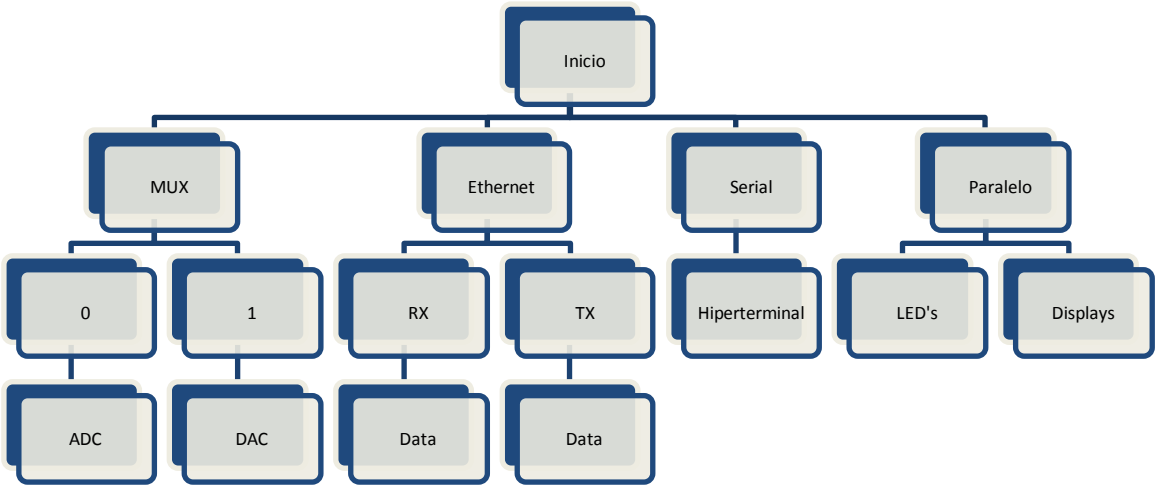

Figura 4. 32 Organización del sistema.

De manera muy simple se puede decir que el software debe encargarse de primeramente habilitar los módulos hardware diseñados, tales como: MUX, DAC y ADC, así como también los módulos de fabricante usados, siendo en este caso el del módulo Ethernet el más importante. Es por ello que contamos con un inicio general para estos módulos. Al inicio también deben ser creadas todas las variables requeridas para almacenar y enviar valores, según corresponda por módulo de comunicación.

4.3 Análisis del Sistema a verificar

El sistema a verificar deberá ser un dispositivo del cual se conozca los resultados a ciertos estímulos, ya que el test será realizado vía el sistema diseñado, es así que nuestro sistema se convierte en un Laboratorio Virtual, el cual puede ser evaluado a distancia.

4.3.1 Modelo en dispositivo de bajo costo

A continuación se mostrará el modelo de verificación del sistema más simple que puede implementarse:

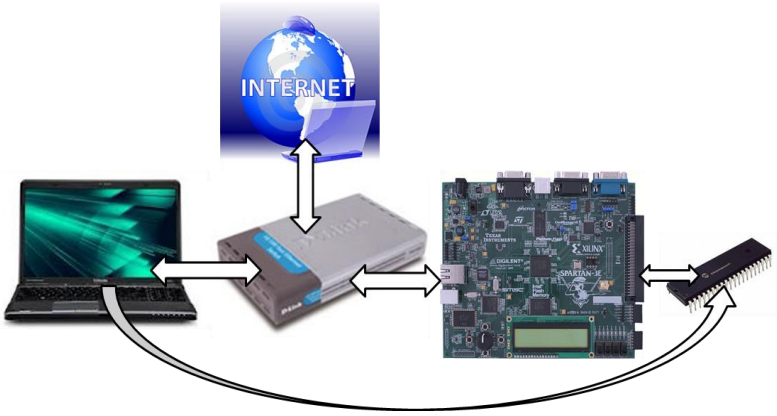

Figura 4. 33 Representación de la plataforma para una aplicación sencilla.

Este sistema está conformado por los siguientes elementos:

- FPGA: El cual se comunica con el dispositivo a ser testeado.
- Switch: Se encarga de realizar la comunicación vía Ethernet entre los elementos del sistema.
- PC: Computador que se comunica vía Ethernet al FPGA.

CAPÍTULO V

IMPLEMENTACIÓN DEL SISTEMA DE DEPURACIÓN

5.1 Implementación del Hardware en la tarjeta de desarrollo

En la **Figura 5.1** se muestra de manera simplificada la implementación requerida para el Hardware del Sistema. El bloque uB representa al procesador Soft-Core Microblaze, los bloques IP_ADC e IP_DAC representan a los módulos para el manejo de los convertidores ADC y DAC respectivamente, posteriormente se detallará mayor información sobre su funcionamiento y uso dado en el sistema, así como también de otros de los módulos o IP's requeridos.

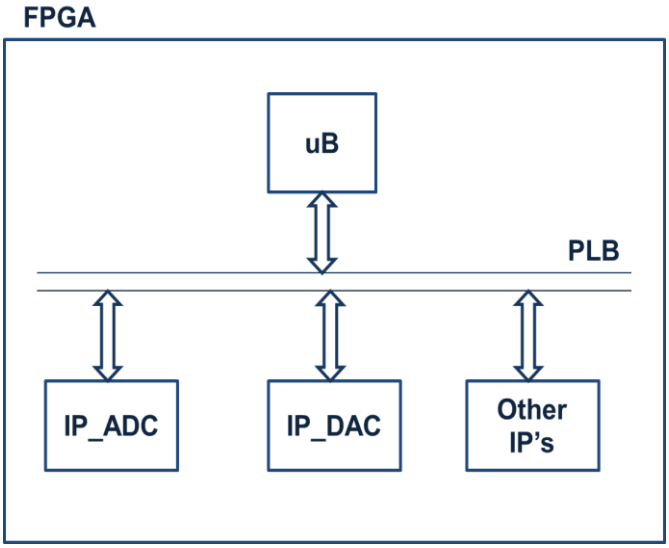

Figura 5. 1 Representación simplificada del Sistema Hardware.

Para realizar la conexión entre módulos (IP cores) y el procesador Microblaze debe seguirse la metodología implementada por las herramientas EDA de Xilinx, es de mucha ayuda revisar los archivos creados automáticamente por estas herramientas en el tiempo de diseño de cada uno de los archivos.

El archivo **README**, se encuentra ubicado en la raíz que tiene la forma:

"CARPETA DE PROYECTO"\pcores\"NOMBRE DE IP CORE"\devl

Por ejemplo: D:\AllIneedtowrite\thesis_pedro\pcores\interface_mux_v1_00_a\devl

Este documento presenta el siguiente contenido:

1) Peripheral Summary: Resumen general de la IP core en diseño.
2) Description of Generated Files: Descripción de cada uno de los archivos de diferente extensión creados.
3) Description of Used IPIC Signal: Descripción de cada una de las señales involucradas en la IP core en diseño.
4) Description of Top Level Generics: Descripción de datos genéricos, los cuales vienen a ser como etiquetas dentro de la codificación de la IP core en diseño.

Es recomendable trabajar y verificar funcionamiento de cada uno de los elementos hardware de manera independiente antes de ser añadidos al sistema.

5.1.1 Configuración del Procesador

La configuración del procesador Microblaze para el presente sistema incluye los siguientes puntos:

- Indicar frecuencia de trabajo (frecuencia que será distribuida por el bus).

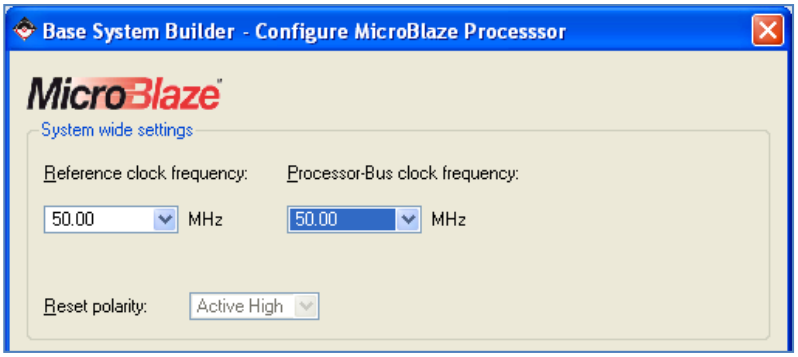

Figura 5. 2 Configuración de frecuencia del bus.

- Tamaño de memoria, indicar que es memoria interna, es decir *BlockRAM.*
- Indicar que no se está usando el FPU (hardware para trabajar con punto flotante).
- No se incluye bloque para depuración, el procesador Microblaze es un procesador Soft-Core del propietario Xilinx.

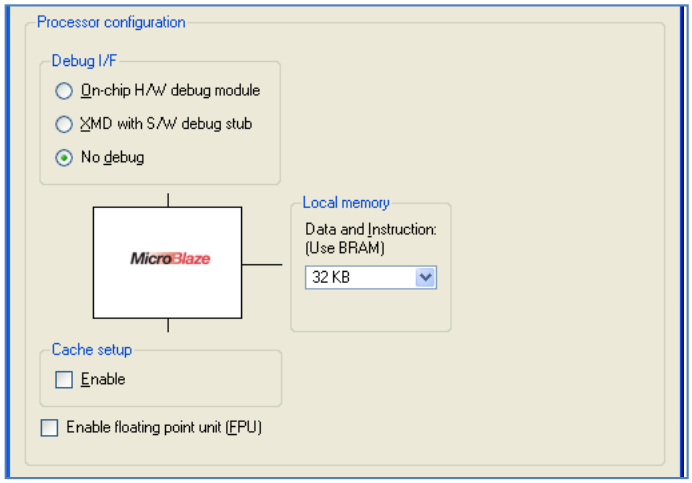

Figura 5. 3 Otras configuraciones para el Procesador Microblaze.

En el proceso de configuración del procesador Microblaze son añadidas las siguientes IP de Fabricante:

- Módulos de comunicación: Serial y Ethernet.
- Módulo para entradas y salidas: LED's, Switches.

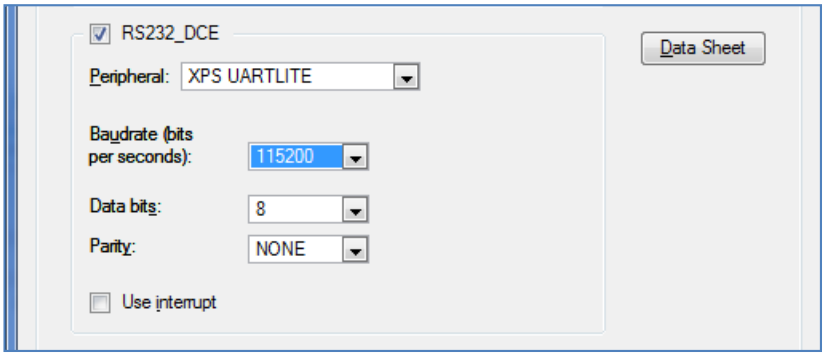

Figura 5. 4 Módulo para comunicación Serial (conector tipo hembra en tarjeta)

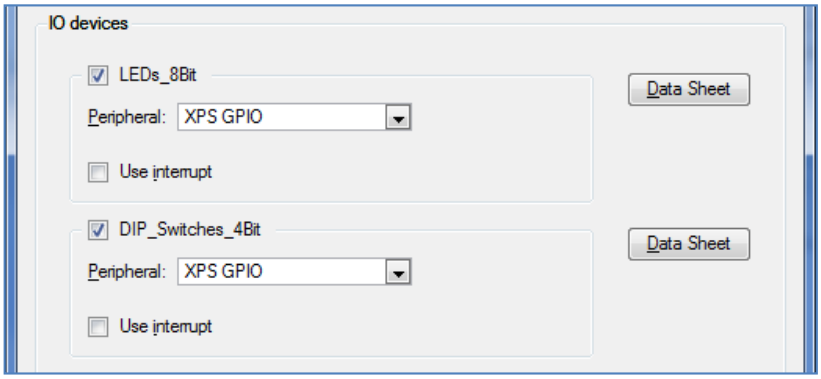

Figura 5. 5 Módulos para entradas y salidas.

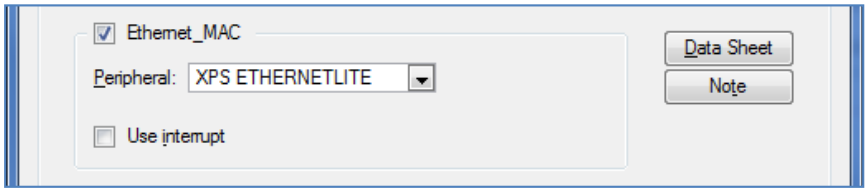

Figura 5. 6 Módulo para comunicación Ethernet.

Luego de haber realizado estas configuraciones el sistema requiere tomar un medio de comunicación para el trabajo de entrada y salida estándar (STDIN, STDOUT), así como también una memoria de inicio (o arranque).

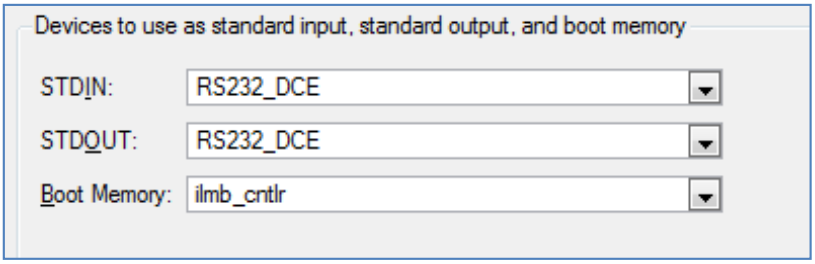

Figura 5. 7 Configuraciones finales (generalmente por defecto).

Luego de todas las configuraciones se cuenta con el siguiente resumen, para poder realizar verificación del sistema antes de la confirmación para su generación.

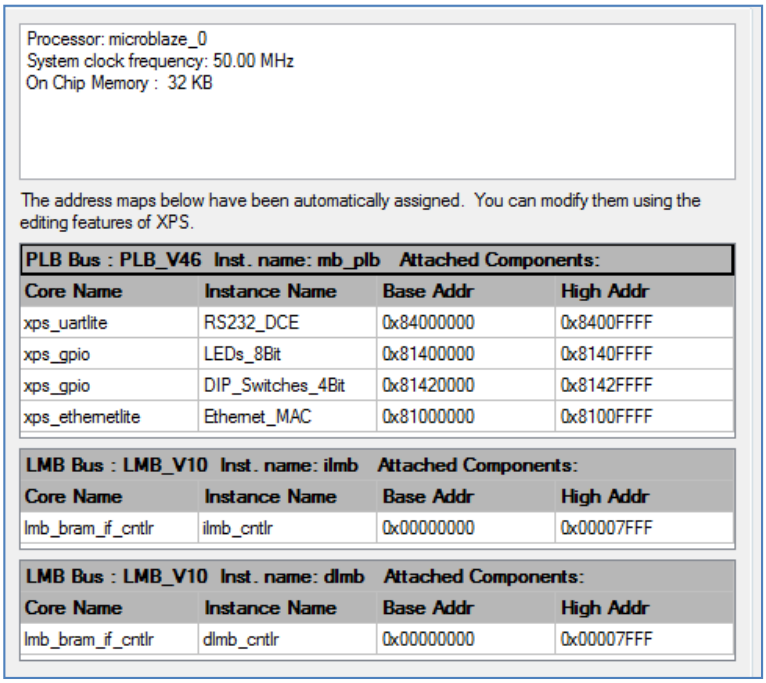

Figura 5. 8 Resumen del sistema.

Luego de la generación del sistema se obtendrá un primer sistema, tal como se muestra en la Figura 5.9, el cual cuenta con el procesador Microblaze, y solamente con los IP Cores de Fabricante.

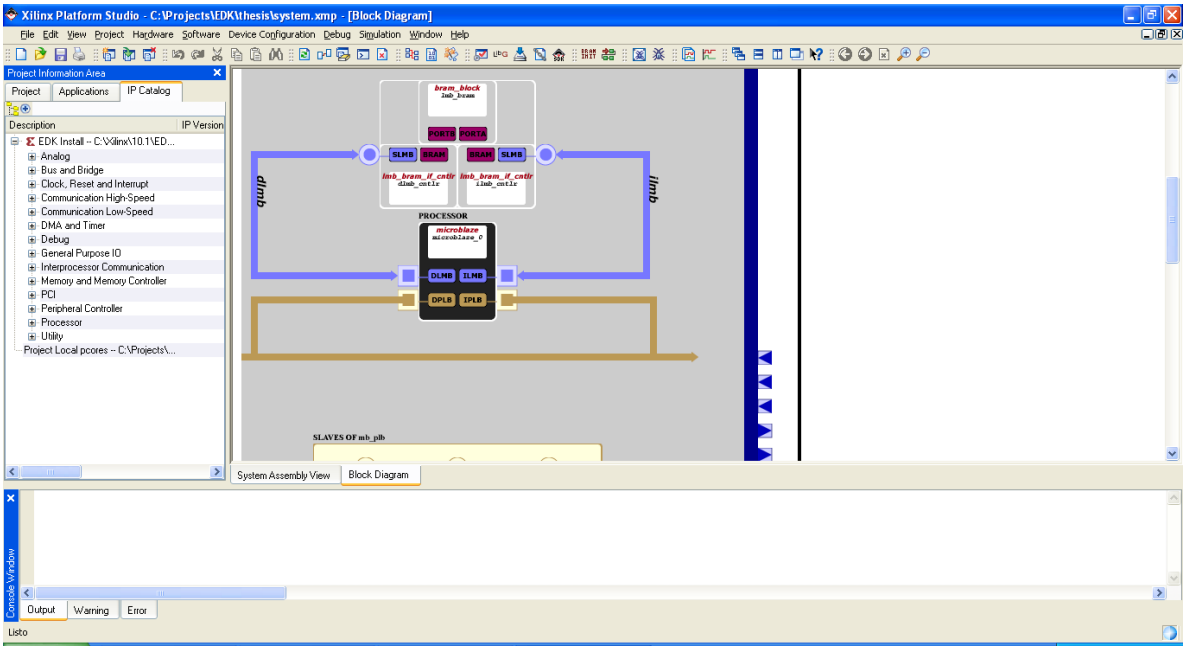

Figura 5. 9 Entorno EDK (Kit de Desarrollo Embebido).

A partir del sistema creado deben añadirse las IP Cores de usuario en diseño, lo cual involucra una conexión de cada uno de ellos al bus del sistema, así como una actualización de direcciones en el sistema, esto no fue necesario anteriormente ya que fue realizado automáticamente, pero en lo posterior debe realizarse manualmente.

5.1.2 Adición de IP Cores

A continuación se procede con la adición de las IP Cores de usuario, las cuales son creadas para realizar el manejo de los periféricos requeridos en el sistema. Para el desarrollo de cada nuevo hardware a usar de nuestra tarjeta, se hace uso de la herramienta ISE de Xilinx, con el propósito de verificar funcionamiento de manera independiente de cada módulo, antes de su adición al sistema.

Los módulos para el manejo de los convertidores ADC y DAC requieren el manejo del protocolo de comunicación SPI, el bloque (o módulo) MUX realiza la gestión de las señales de comunicación de estos dos módulos, ya que requieren ser administradas para evitar errores.

IP DEL ADC: pedro_adc

Para el manejo del convertidor A/D se diseña una máquina de estados, debe permitir seguir el diagrama de tiempos que se muestra en la Figura 4.20, aquí como podemos ver se tiene la señal AD_CONV, la cual nos sirve para marcar el inicio de la conversión, la cual tiene una duración de por lo menos 34 ciclos de reloj del sistema.

Las otras 2 señales (SPI_SCK y SPI_MISO), son el *clock* del sistema (ésta se creará a partir del *clock* de 50MHz con el que cuenta la tarjeta) y la otra señal es la entrada maestro y salida esclavo, la cual escribe la data convertida de los canales A y B del circuito convertidor.

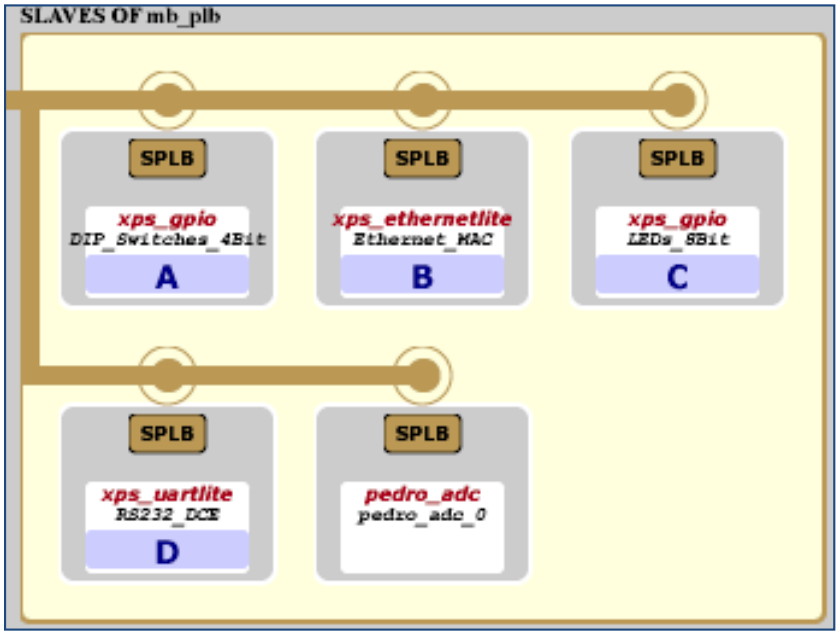

Figura 5. 10 Sistema luego de añadir bloque ADC (pedro_adc).

Debido a que la interface SPI es usada por otros dispositivos en el sistema, los cuales el FPGA también controla como maestro, entonces para poder usar el convertidor y el pre-amplificador se tendrá que deshabilitar los dispositivos mediante sus pines de *chip-select*, tal como se muestra en la Tabla 4.6.

En la codificación VHDL esto se traduce a lo siguiente:

```
ADC_SPI_SS_B      <= '1';
ADC_DAC_CS        <= '1';
ADC_SF_CE0        <= '1';
ADC_FPGA_INIT_B   <= '1';
```

La señal de reloj para este módulo se obtiene a partir de la señal del bus principal, haciendo uso de un divisor de frecuencia, con lo cual se obtiene una señal de aproximadamente 1.5MHz.

```
-- 1.5MHz clock
                clock_divider : process (Bus2IP_Clk)
                begin
                  if(rising_edge(Bus2IP_Clk)) then
                        if(counter = 33) then
                                risingedge <= risingedge xor '1';
                                clk_sample <= clk_sample xor '1';
                                counter <= 0;
                        else
                                counter <= counter + 1;
                        end if;
                end if;
                end process;
```

La máquina de estados cuenta con tres estados, los cuales son descritos de manera breve en la Figura 5.11.

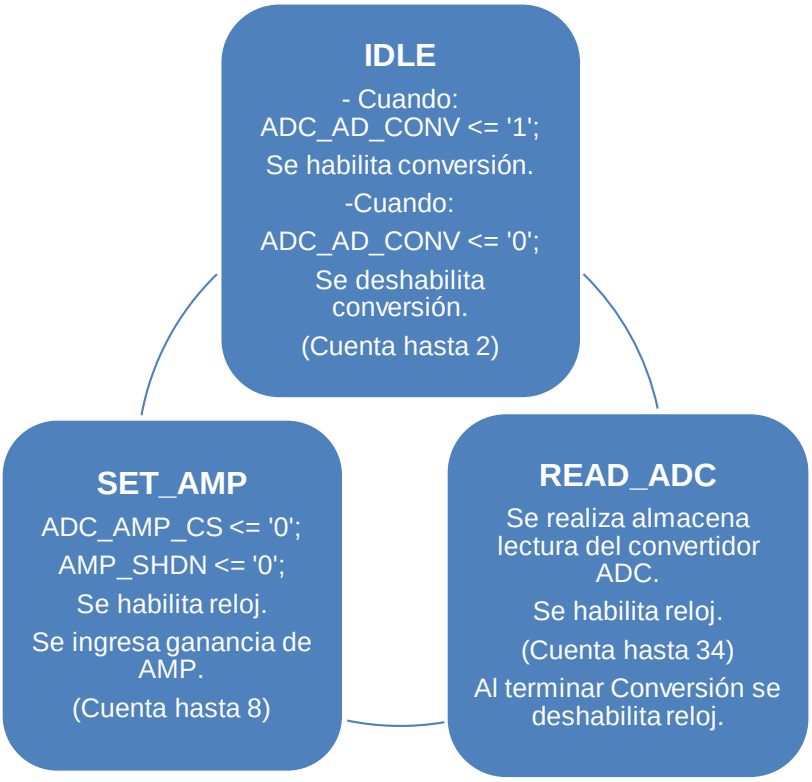

Figura 5. 11 Estados de la Máquina.

IP DEL DAC: pedro_dac

El manejo de este módulo D/A se realiza con una máquina de estados, la cual cuenta con 2 estados, uno para lectura de la *data* digital y el otro para la conversión de esta *data* digital a analógica. Este módulo usa interface SPI teniendo como maestro en este caso al FPGA mediante la máquina de estados. El circuito DAC cuenta con 4 canales (A, B, C y D), cada uno de ellos puede ser seleccionado en la trama de *data* enviada, mediante 4 bits de dirección que seleccionan el canal o los canales a usar.

En la Figura 4.25 puede observarse que la *data* a enviar es de 32 bits y que está divida en 5 campos, siendo ellos *Don't care, Command, Address, Data* y nuevamente *Don't care*. Las conexiones de las salidas del DAC son hechas a la cabecera de pines de la tarjeta de desarrollo, tal como se muestra en Figura 4.22.

Al igual que en el convertidor ADC, se deben deshabilitar los otros dispositivos que comparten la interface SPI; se deben usar los valores lógicos mostrados en la Tabla 4.8. En la codificación VHDL será como sigue:

```
SPI_SS_B        <= '1';
AMP_CS          <= '1';
AD_CONV         <= '0';
SF_CE0          <= '1';
FPGA_INIT_B     <= '1';
```

En la Figura 5.12 se describe brevemente los estados de la máquina de estados.

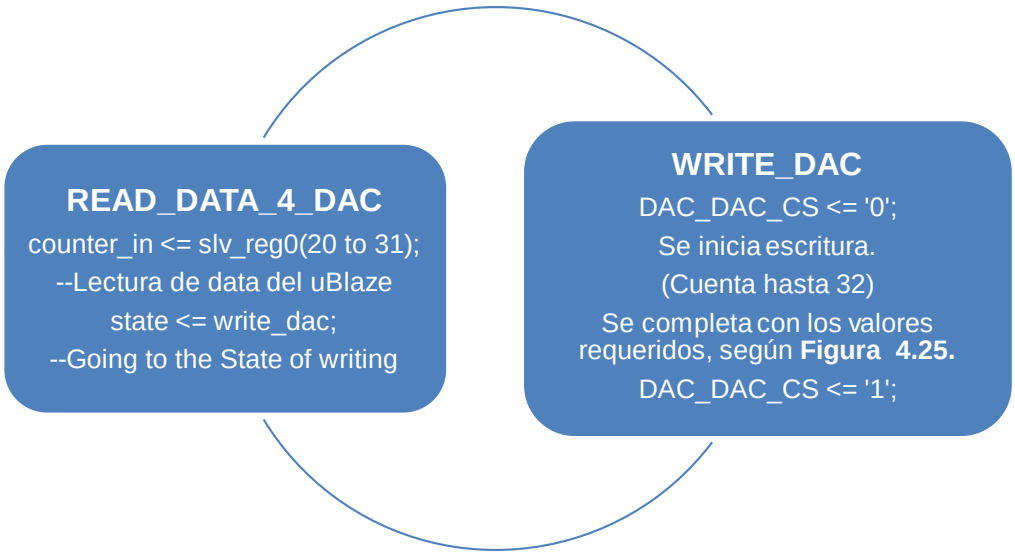

Figura 5. 12 Estados de la Máquina para el manejo de módulo DAC.

ENTORNO ISE DE XILINX

Como es indicado al inicio del capítulo, es recomendable realizar pruebas a los módulos de manera independiente, es por ello que primeramente se trabaja en el entorno ISE de Xilinx, para luego realizar la adición de cada IP core diseñada.

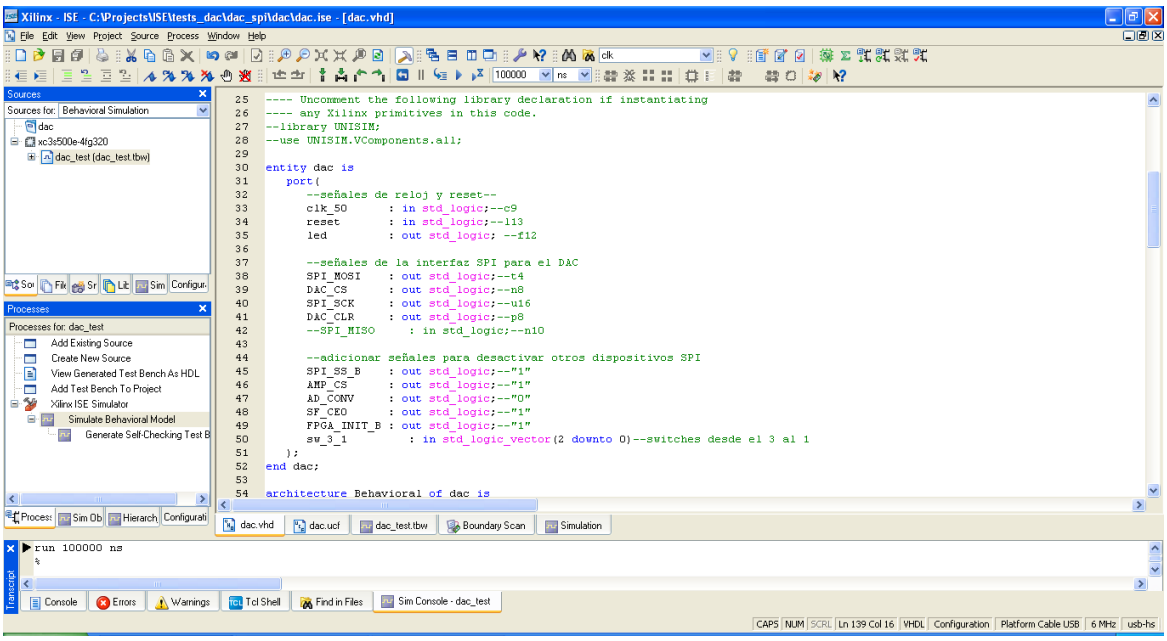

Figura 5. 13 Entorno de desarrollo ISE de Xilinx.

Se verifica el código en este entorno y se realiza la implementación en la tarjeta de desarrollo después de hacerse la simulación respectiva sobre ésta misma herramienta de desarrollo.

SIMULACIÓN

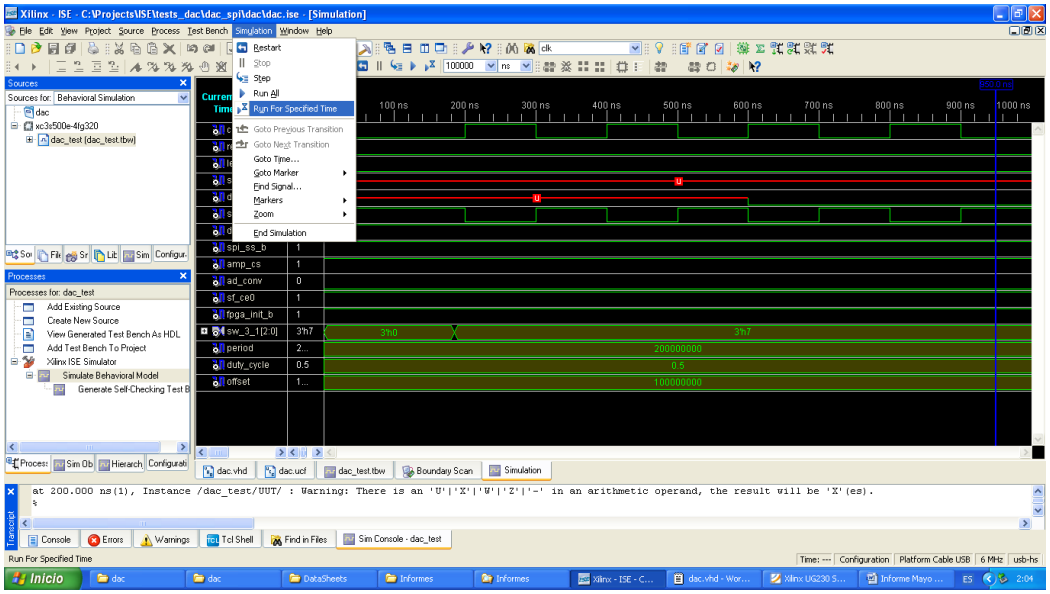

Figura 5. 14 Simulación con tiempo de simulación por defecto.

Para poder visualizar la simulación con mayor claridad, se puede cambiar el tiempo de simulación en la pestaña Simulation como es que se muestra en la Figura 5.14.

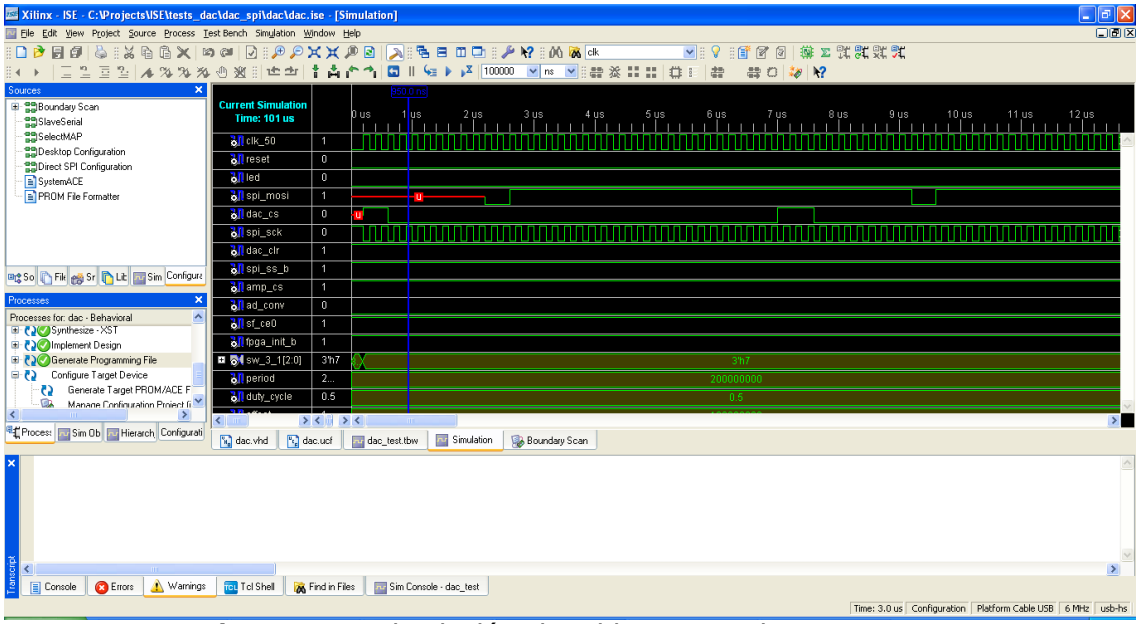

Figura 5. 15 Simulación obtenida para un tiempo mayor.

Luego de realizarse la correcta implementación de este módulo en hardware, se puede realizar la conección de ésta IP core al sistema basado en procesador Microblaze y así poder enviar la data a convertir desde el procesador vía instrucciones en código C.

A continuación se muestra la Figura 5.16, la cual muestra una ampliación de las conexiones para visualizar de mejor manera los esclavos conectados al bus PLB.

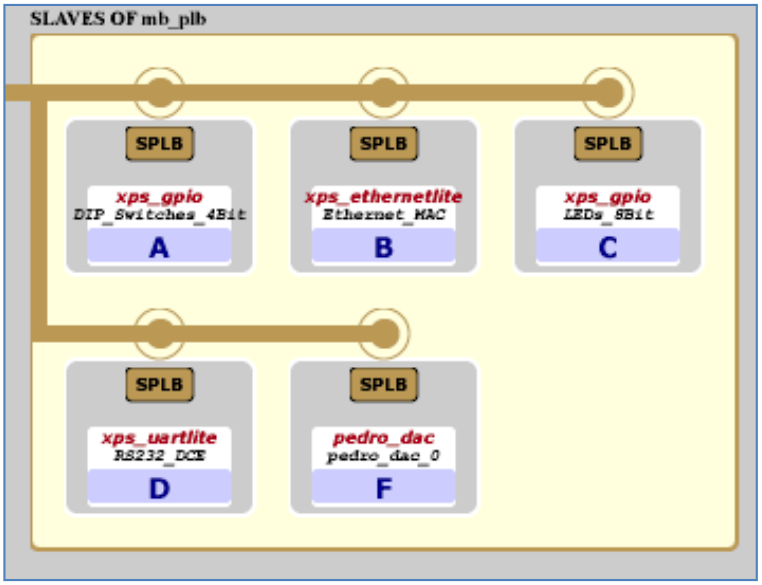

Figura 5. 16 Sistema luego de añadir bloque DAC (pedro_dac).

Para poder manejar ésta IP core se deberán usar instrucciones adicionales, tales como se muestran a continuación:

```
/***************CODIGO PARA PRUEBA DE PEDRO_DAC IP CORE*************************/

                if (data == 4096) {
                data_dac--;
                }
                else {
                data = data_dac;
                data_dac++;
                }
        PEDRO_DAC_mWriteReg(XPAR_PEDRO_DAC_0_BASEADDR, 0, data_dac);
/*******************************************************************************************************/
```

IP DEL MUX: pedro_mux

Debido a que algunas de las señales del bus SPI son compartidas por varios dispositivos, será necesario deshabilitar dispositivos que no son requeridos en el sistema, para lograr una correcta comunicación.

Para dar solución al problema de contar con señales compartidas del Bus SPI entre el módulo ADC y DAC se presentan diferentes alternativas, tales como crear una IP con un control de las señales, con lo cual ya no serían compartidas al ser una sola IP, otra solución es diseñar un multiplexor (MUX) que pueda ser controlado para enviar estas compartidas y las demás señales necesarias, tal como se muestra en la Figura 5.17.

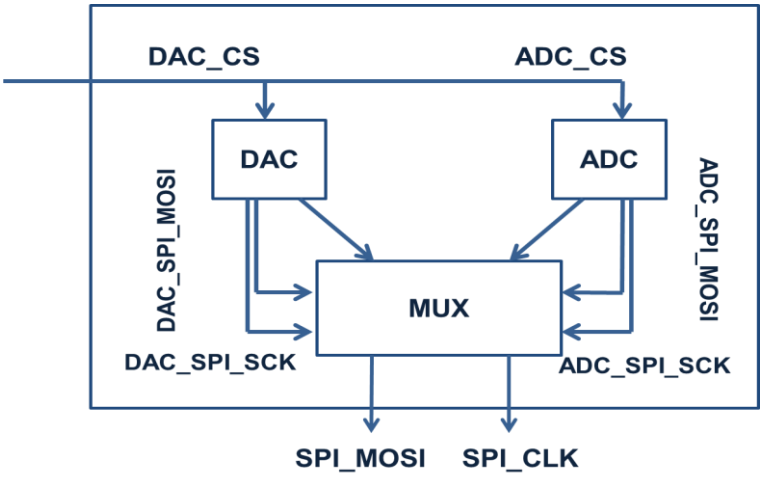

Figura 5. 17 Multiplexor diseñado.

Se toma la solución haciendo uso de un multiplexor (MUX), el cual habilita al módulo DAC o ADC, según sea indicado, la cual en este caso es vía software, realizada por el procesador Microblaze, haciendo uso de la instrucción:

```
INTERFACE_MUX_mWriteReg(XPAR_INTERFACE_MUX_0_BASEADDR, 0, dac_on);
```

El valor la variable dac_on selecciona al módulo DAC o ADC. La arquitectura de este multiplexor es como sigue:

```
--USER logic implementation added here

        adc_n_dac <= slv_reg0(31);
        -----*****************************************************************-----
        with adc_n_dac select
                SPI_MOSI <= ADC_SPI_MOSI when '0',
                                        DAC_SPI_MOSI when others;
        with adc_n_dac select
                SPI_SCK         <=      ADC_SPI_SCK when '0',
                                        DAC_SPI_SCK when others;
        with adc_n_dac select
                AD_CONV         <= ADC_AD_CONV when '0',
                                        DAC_AD_CONV when others;
        with adc_n_dac select
                DAC_CS          <=      ADC_DAC_CS when '0',
                                        DAC_DAC_CS when others;
        with adc_n_dac select
                SPI_SS_B <= ADC_SPI_SS_B when '0',
                                        DAC_SPI_SS_B when others;
        with adc_n_dac select
                FPGA_INIT_B <= ADC_FPGA_INIT_B when '0',
                                        DAC_FPGA_INIT_B when others;
        with adc_n_dac select
                AMP_CS          <= ADC_AMP_CS when '0',
                                        DAC_AMP_CS when others;

        with adc_n_dac select
                SF_CE0          <= ADC_SF_CE0 when '0',
                                        DAC_SF_CE0 when others;
```

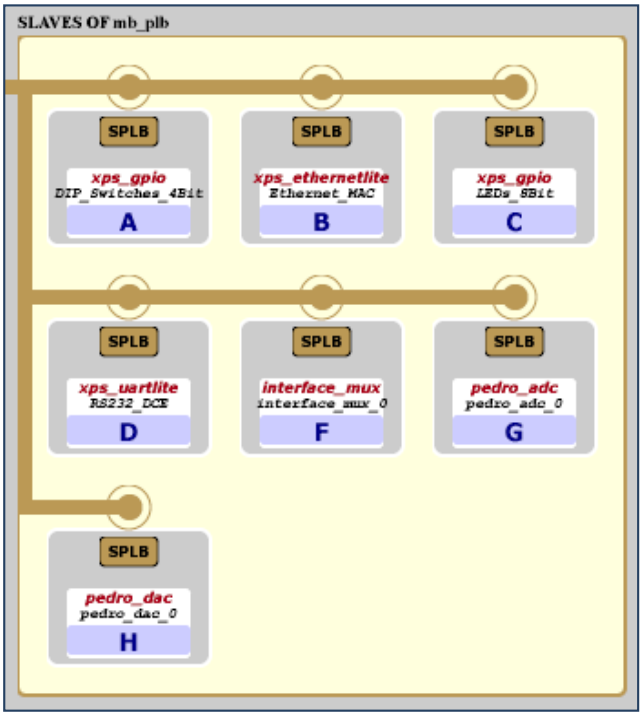

Figura 5. 18 Sistema con todos los IP Cores requeridos.

5.2. Software del procesador embebido

- Las siguientes funciones son usadas para configurar, leer y escribir data para el núcleo Ethertnetilte MAC (core):

Variables requeridas:

// Create an Instance of the EmacLite driver and a pointer to it.
static XEmacLite EmacLiteInstance
XEmacLite *EmacLiteInstancePtr = &EmacLiteInstance

Los buffers:

// Create Frame Buffers for Tx and Rx - on 32 bit word boundary
static Xuint32 frame_Rx_buffer[MAX_FRAME_SIZE_IN_WORDS]
static Xuint32 frame_Tx_buffer[MAX_FRAME_SIZE_IN_WORDS]

Configuración del EmacLiteInstance:

XEmacLite_Initialize(EmacLiteInstancePtr, XPAR_XPS_ETHERNETLITE_0_DEVICE_ID)
XEmacLite_SetMacAddress(EmacLiteInstancePtr, mac_addr)
XEmacLite_FlushReceive(EmacLiteInstancePtr)

Read Rx data:

XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8 *)frame_Rx_buffer)

Write Tx data:

XEmacLite_Send(EmacLiteInstancePtr, (Xuint8 *)frame_Tx_buffer,
(unsigned)buffer_length)

- Las siguientes funciones son usadas para configurar y escribir un valor para el convertidor DAC:

Variable requerida:

Xuint8 data_dac=0;

Habilitar módulo DAC:

INTERFACE_MUX_mWriteReg(XPAR_INTERFACE_MUX_0_BASEADDR, 0, 1);

Escribir data para conversión:

PEDRO_DAC_mWriteReg(XPAR_PEDRO_DAC_0_BASEADDR, 0, data_dac);

- Las siguientes funciones son usadas para configurar y escribir un valor para el convertidor ADC:

Variable requerida:

Xuint8 data_adc=0;

Habilitar módulo ADC:

INTERFACE_MUX_mWriteReg(XPAR_INTERFACE_MUX_0_BASEADDR, 0, 0);

Leer data convertida:

data_adc=PEDRO_ADC_mReadReg(XPAR_PEDRO_ADC_0_BASEADDR, 0);

El flujo del programa se describe de la siguiente manera:

• Inclusión de archivos.
Definiciones.
Asignación de variables globales.

• Rutina de inicialización del driver “Xilinx Ethernetlite MAC”

• Rutina principal.
Declaración de buffers de transmisión/recepción.
Chequear si ha llegado la trama Ethernet.
Chequear para una entrada de usuario.
Enviar una trama Ethernet.

• Funciones de proceso.
Enviar frame UDP.
Procesa una trama Ethernet recibida.

• Funciones de ayuda.
Rutina compara byte buffer.
Rutina copia byte buffer.
Rutina de checksum.

Nota: Los módulos DAC y ADC también son inicializados, pero los tiempos para la selección de uno u otro dependerán de la aplicación en particular.

5.2.1 Configuración de la adquisición y distribución de datos

La distribución de datos a la actualidad se viene trabajando con la siguiente configuración:

Data for LED’s	Data for Display	Data for ADC

```
Xuint8 d[12]={   0x30, 0x30, 0x30, 0x30,   //leds
                 0x30, 0x30, 0x30, 0x30,   //display
                 0x30, 0x30, 0x30, 0x30};  //adc
//El valor 0x30 es equivalente a 30 en base 16, el cual como número ASCII es 0.
```

```
/*********************************************************************/
Xuint8 d[12]={   0x30, 0x30, 0x30, 0x30,   //leds
                 0x30, 0x30, 0x30, 0x30,   //display
                 0x30, 0x30, 0x30, 0x30};   //adc
        //El valor 0x30 es equivalente a 30 en base 16, el cual como número ASCII es 0.
/*********************************************************************/
/*********************************************************************/
//Xuint8 mesg_buf[] = {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30};

Xuint8 mesg_buf[] = {d[11],d[10],d[9],d[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]};

long_msje = strlen( mesg_buf );//uso del String.h para medir el tamaño del mensaje
//        xil_printf("Longitud_Mensaje: %d\n\r",long_msje);
        long_trama = construccion_trama( mesg_buf, long_msje, TxBuf_p );
//        xil_printf("Longitud_Trama: %d\n\r",long_trama);

/*********************Envia Instrucción**********************/
XEmacLite_Send(EmacLiteInstancePtr, (Xuint8 *)trama_Tx,(unsigned)long_trama);
/*******************Recibe instrucción*********************************/
if(instruccion=XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8 *)trama_Rx)!=0x00){

//if(XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8 *)trama_Rx)!=0x00){
//xil_printf("Instruccion: %x\n\r",instruccion);
//instruccion=XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8 *)trama_Rx);
instruccion = analisis_trama( RxBuf_p );
//        xil_printf("Instruccion: %x\n\r",instruccion);
                                    }
/*********************************************************************/
```

5.3 Interface Gráfica de Usuario

La interface gráfica de usuario consta de los siguientes archivos de diseño (archivos de extensión *.java):

- TareaRx.java: Es el archivo encargado de la recepción de datos, en este caso recepción de los datos enviados por el FPGA.

- TareaTx.java: Es el archivo encargado de la transmisión de datos, es decir datos enviados desde la PC host al FPGA.
- Ventana.java: Es el archivo encargado de crear los componentes gráficos de la interface de usuario.
- VirtuaLab.java: Es el archivo principal, encargado de iniciar las diferentes tareas, y de ese modo administrar el control de envío y recepción de datos en el sistema.

5.3.1 Configuración de la adquisición y distribución de datos

Los datos en la Interface de usuario son representados mediante los siguientes componentes:

- Botones
- Switches
- Displays

SWITCHES

Se crean 8 botones, nombrados de la siguiente manera: SW0, SW1,…, SW7

```
        //Creando los botones a usar
        JButton b1 = new JButton("SW7");
        JButton b2 = new JButton("SW6");
        JButton b3 = new JButton("SW5");
        JButton b4 = new JButton("SW4");
        JButton b5 = new JButton("SW3");
        JButton b6 = new JButton("SW2");
        JButton b7 = new JButton("SW1");
        JButton b8 = new JButton("SW0");
```

Se sitúan adecuadamente dentro de la ventana diseñada para la Interface de Usuario.

```
        //Cambiando la posición de los botones
//        this.setLayout(null);
//        b1.setBounds(50, 50, 100, 100);
        JPanel bottomPanel = new JPanel();
        JPanel holdAll = new JPanel();
```

Se añaden las características necesarias a los botones usados para ser switches de la interface de usuario.

```
/*****************************************/
        //Cambiando el color por defecto de los botones
        b1.setBackground(Color.blue);
```

```
b1.setLayout(null);
b1.setBounds(5, 270, 60, 30);
//botones.setBounds(-30, 270, 600, 60);

b2.setBackground(Color.blue);
b2.setLayout(null);
b2.setBounds(70, 270, 60, 30);

b3.setBackground(Color.blue);
b3.setLayout(null);
b3.setBounds(135, 270, 60, 30);

b4.setBackground(Color.blue);
b4.setLayout(null);
b4.setBounds(200, 270, 60, 30);

b5.setBackground(Color.blue);
b5.setLayout(null);
b5.setBounds(265, 270, 60, 30);

b6.setBackground(Color.blue);
b6.setLayout(null);
b6.setBounds(330, 270, 60, 30);

b7.setBackground(Color.blue);
b7.setLayout(null);
b7.setBounds(395, 270, 60, 30);

b8.setBackground(Color.blue);
b8.setLayout(null);
b8.setBounds(460, 270, 60, 30);
```

Se añade la característica de Listener para que puedan “escuchar” al ser presionados:

```
//Listen for actions on buttons 1 and 8.
b1.addActionListener(this);
b2.addActionListener(this);
```

```
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
b6.addActionListener(this);
b7.addActionListener(this);
b8.addActionListener(this);
```

Se selecciona una acción a realizar para el botón al ser presionado:

```
//b1.doClick();
b1.setActionCommand("disable01");
b2.setActionCommand("disable02");
b3.setActionCommand("disable03");
b4.setActionCommand("disable04");
b5.setActionCommand("disable05");
b6.setActionCommand("disable06");
b7.setActionCommand("disable07");
b8.setActionCommand("disable08");

bottomPanel.setLayout(null);
//bottomPanel.setDoubleBuffered(true);

bottomPanel.add(b1);bottomPanel.add(b2);bottomPanel.add(b3);bottomPanel.add(b4);
bottomPanel.add(b5);bottomPanel.add(b6);bottomPanel.add(b7); bottomPanel.add(b8);
```

LEDS

La distribución de datos a la actualidad se viene trabajando con la siguiente configuración:

buffer = new byte [12]; // one more (from 3 to 6) to send data to display

Data for LED's	Data for Display	Data for ADC

String cadena = new String(data,8,data.length-8); // Data for LED's

String cadena1 = new String(data,4,data.length-8); //Data for Display

String cadena2 = new String(data,0,data.length-8); //Data for ADC

CAPÍTULO VI

EVALUACIÓN Y RESULTADOS

6.1 Evaluación de Confiabilidad del Sistema

La confiabilidad del Sistema está directamente ligada a la confiabilidad de la comunicación Ethernet realizada, así como de la implementación de todos los protocolos de comunicación involucrados en el desarrollo. Es posible aumentar esta confiabilidad al realizar algún tipo de codificación digital a la data que es enviada al FPGA.

6.2 Análisis del sistema de depuración diseñado

En la siguiente figura se muestran todos los dispositivos y equipos usados.

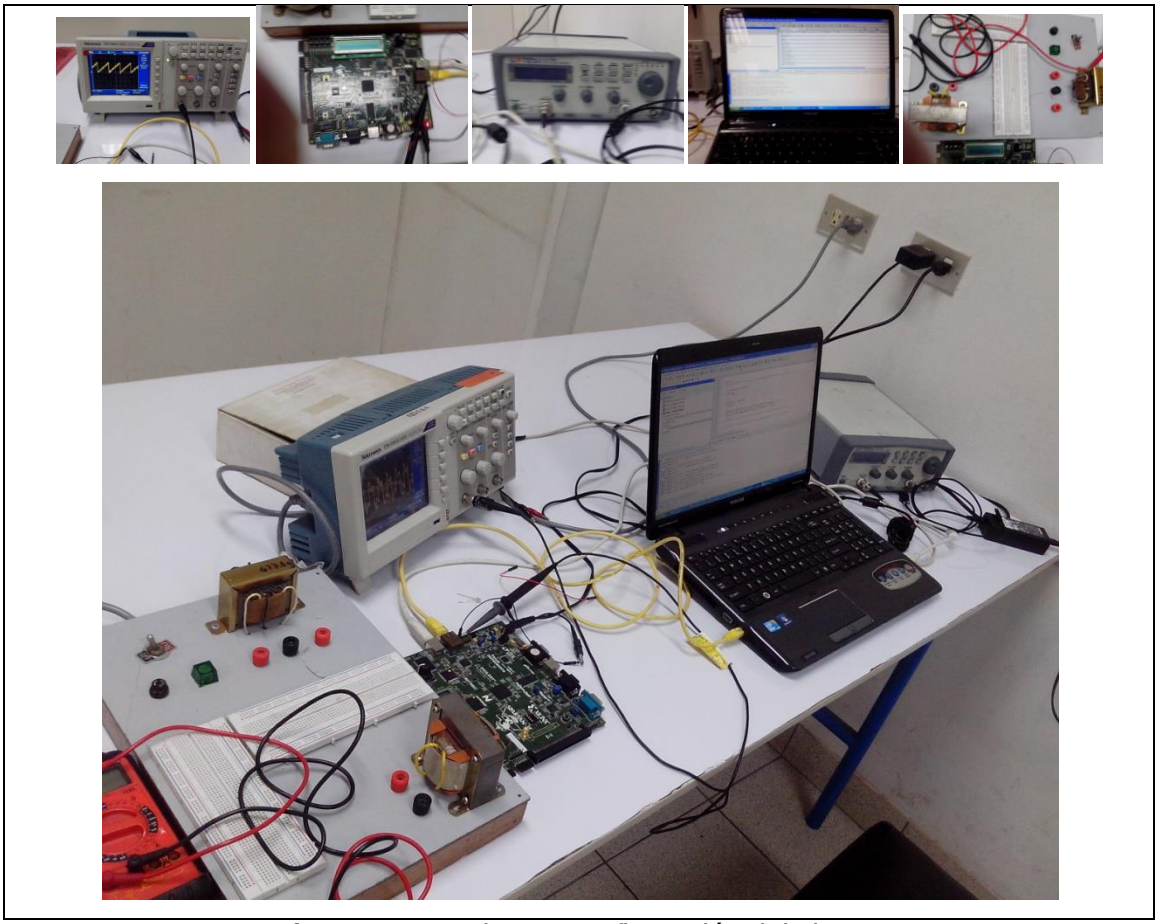

Figura 6. 1 Equipos y configuración del sistema.

En la Figura 6.1 se muestra todo el equipamiento y hardware usado, los cuales podemos enlistar de la siguiente manera:

- Tarjeta de desarrollo Spartan 3E Starter Board.
- Osciloscopio.
- Generador de Señales.
- Panel de conexiones.

Figura 6. 2 Configuración de la Plataforma de Desarrollo para FPGA's

Como puede verse en la Figura 6.2, las conexiones para configuración que se realizan en el FPGA son:

- Fuente de alimentación.
- Conexión USB-JTAG.

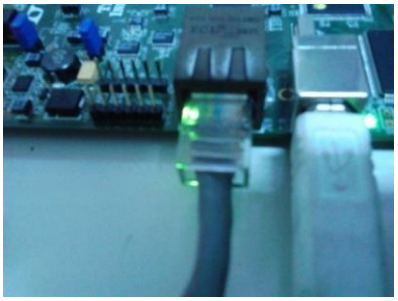

Figura 6. 3 Verificación de Puertos de Comunicación.

Los puertos de comunicación a verificar son:

- Puerto USB: JTAG-USB, el cual es usado para la configuración de la plataforma de desarrollo.
- Puerto Ethernet: Es usado para la comunicación, para envío y recepción de data de la plataforma diseñada y la PC host.

También para las pruebas del sistema de requiere de software de acceso remoto, contamos con las siguientes alternativas:

JRDESKTOP (Códido libre): http://jrdesktop.sourceforge.net/

Figura 6. 4 Enlace de Java Remote Desktop-jrDesktop

Figura 6. 5 Ejecución del software jrDesktop.

Con este software se realizaron pruebas de manera satisfactoria. También se trabajó con una herramienta de acceso libre que cuenta con la ventaja de poder ser ejecutada en dispositivos móviles como celulares o tablets.

TEAMVIEWER: http://www.teamviewer.com/

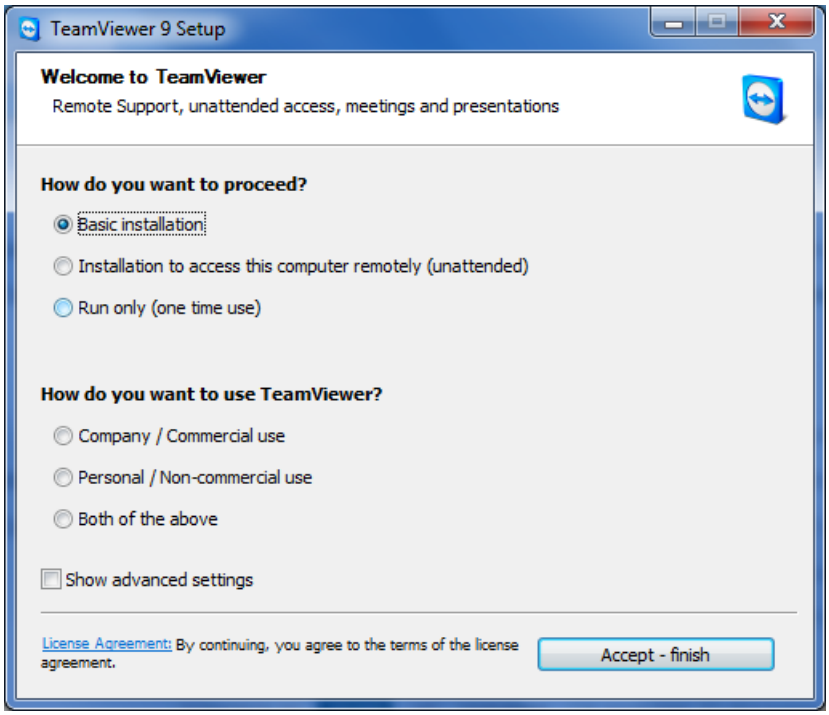

Figura 6. 6 Proceso de Instalación del TeamViewer en PC.

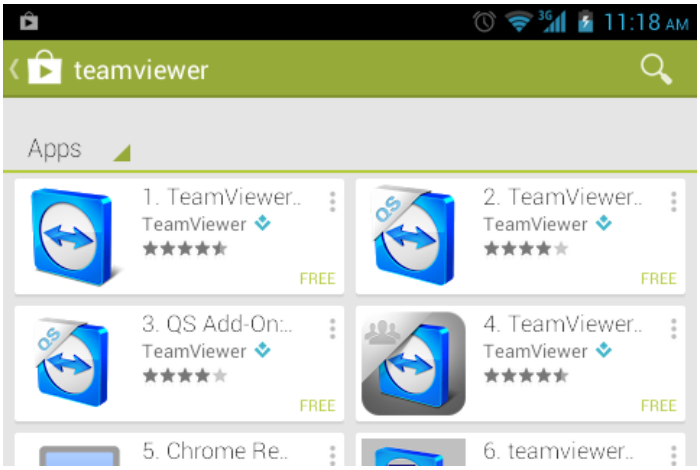

Figura 6. 7 Teamviewer de Android

Figura 6. 8 Proceso de Instalación de TeamViewer en celular con Android.

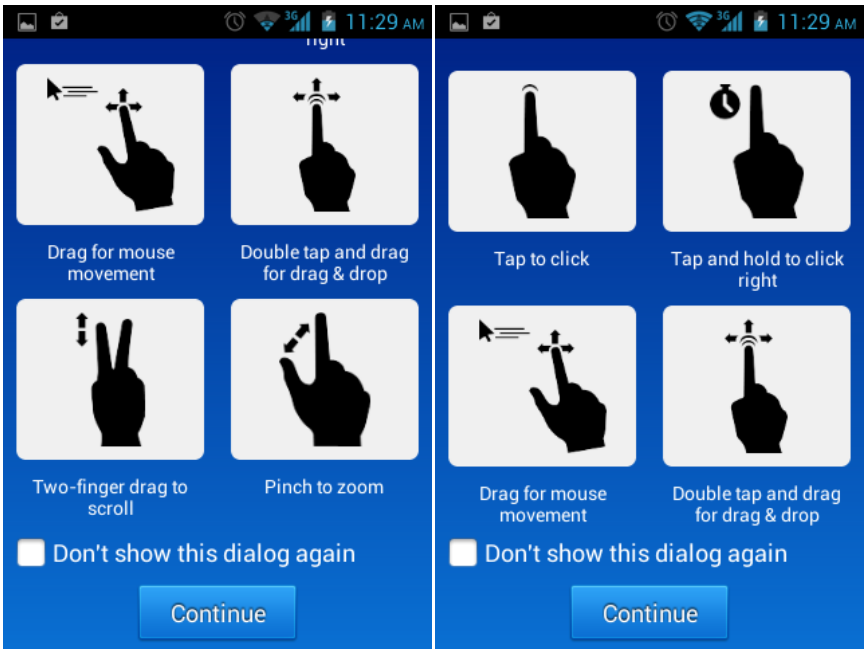

Figura 6. 9 Indicaciones de uso indicadas por TeamViewer para móviles.

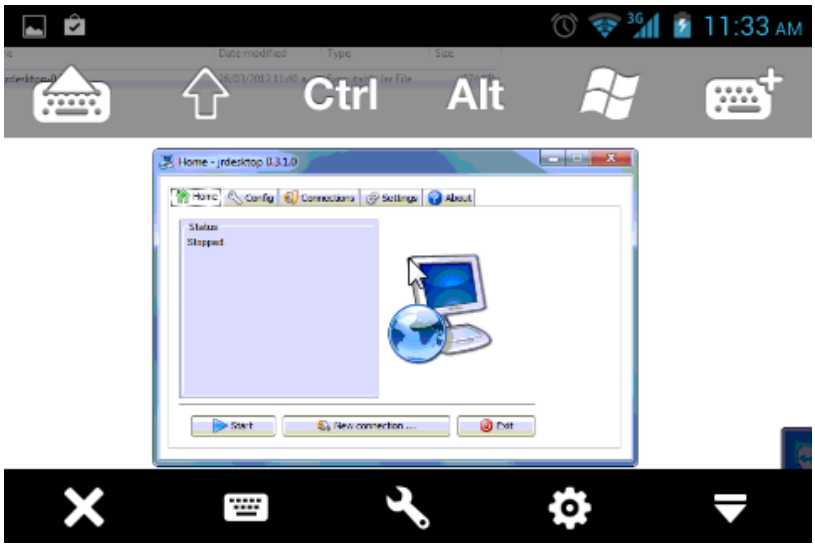

Figura 6. 10 Conexión realizada entre celular y PC host.

6.3 Resultados

El sistema fue en primera instancia probado con el *Switch DES-1008D*. Luego se tuvo la oportunidad de probar el sistema con otro switch de más entradas, el funcionamiento fue similar, con la diferencia que en esta segunda prueba se contaba con la posibilidad de un número mayor de computadores con facilidad para comunicarse vía este *switch* a la plataforma diseñada.

Figura 6. 11 *Switch* usado para la comunicación Ethernet.

En la Figura 6.12 podemos ver que este switch cuenta con muchos más puertos Ethernet disponibles

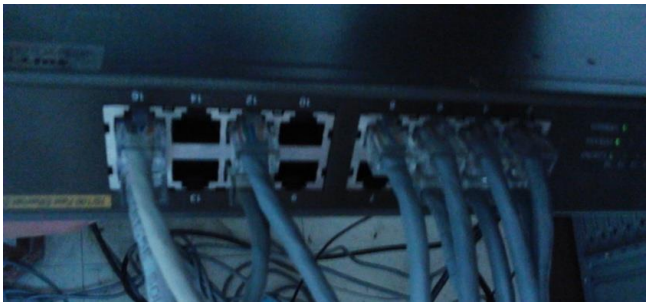

Figura 6. 12 Puertos Ethernet usados en el Laboratorio de prueba.

La primera verificación que se hace es la ejecución de la interface de usuario en dos computadores, la cual también permite verificar la comunicación Ethernet.

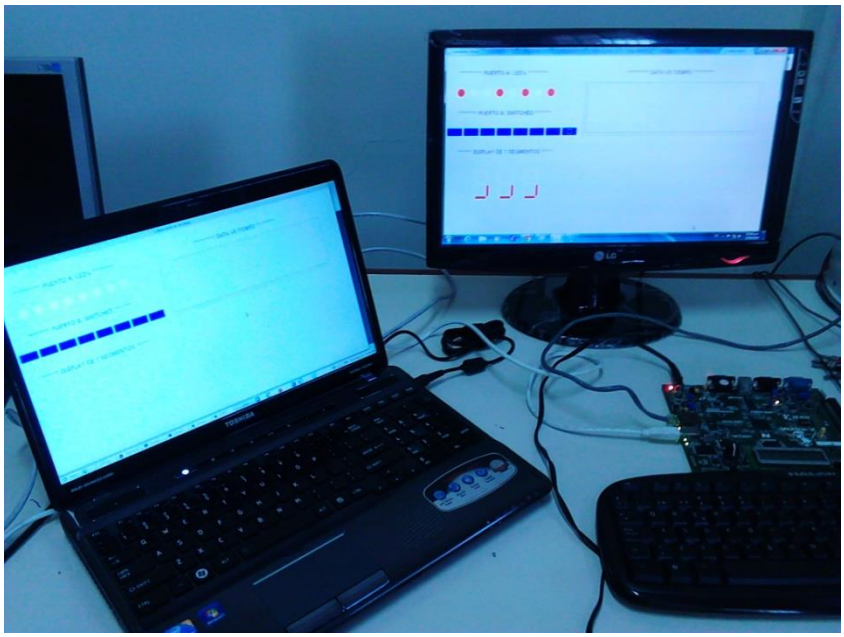

Figura 6. 13 Ejecución de la Interface de Usuario en 2 computadores.

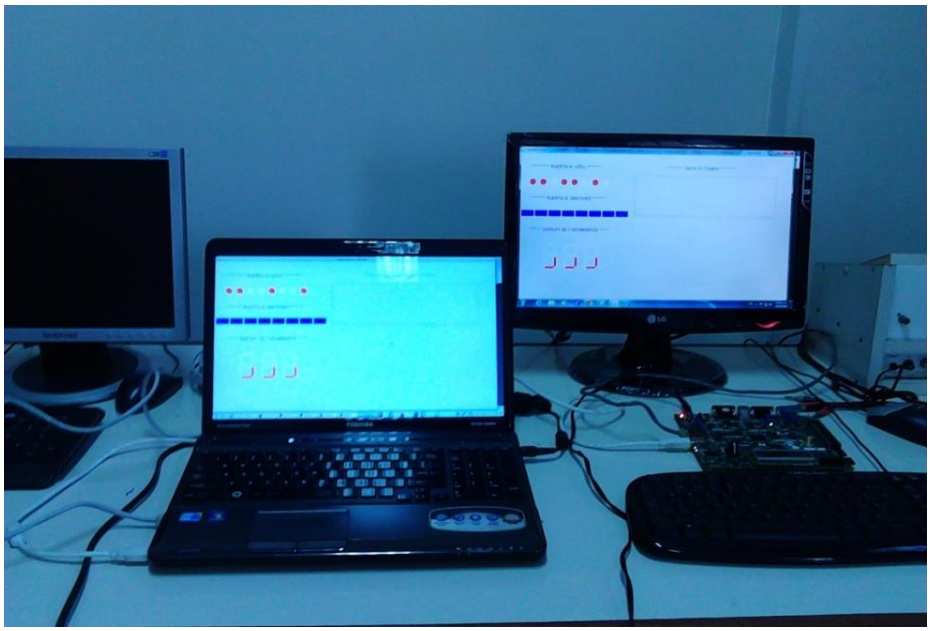

Figura 6. 14 Sistema corriendo en dos máquinas de prueba.

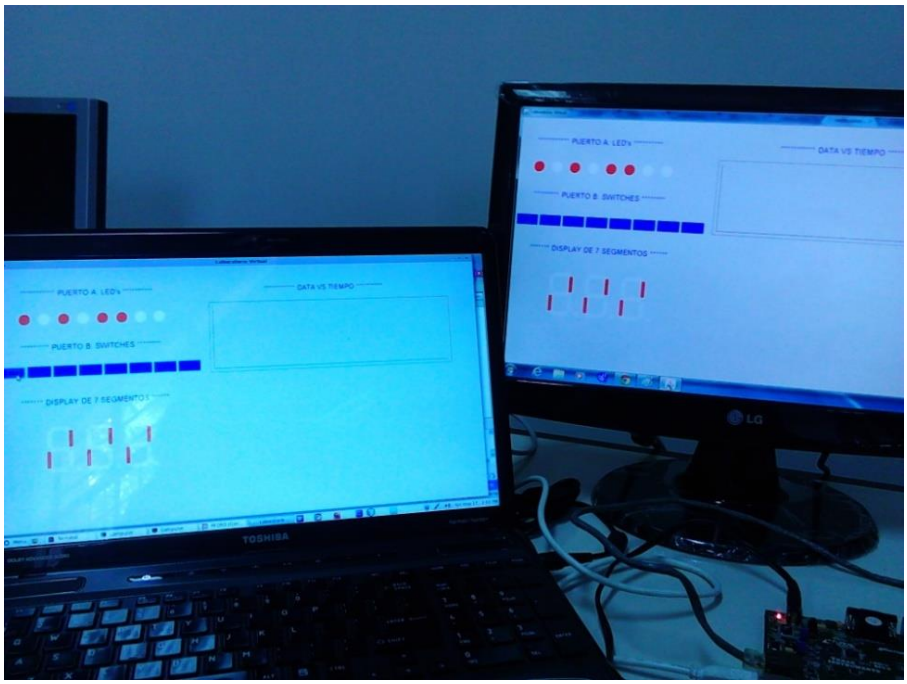

Figura 6. 15 Verificación de Sistema sincronizado.

En la Figura 6.15 se observa a 2 computadores haciendo uso de la Interface de Usuario del sistema, con la característica de encontrarse sincronizados. Se cuenta con un resultado satisfactorio de esta prueba.

A continuación se muestran algunas pruebas de los datos recibidos en el FPGA, mediante envío de datos de la Interface de Usuario.

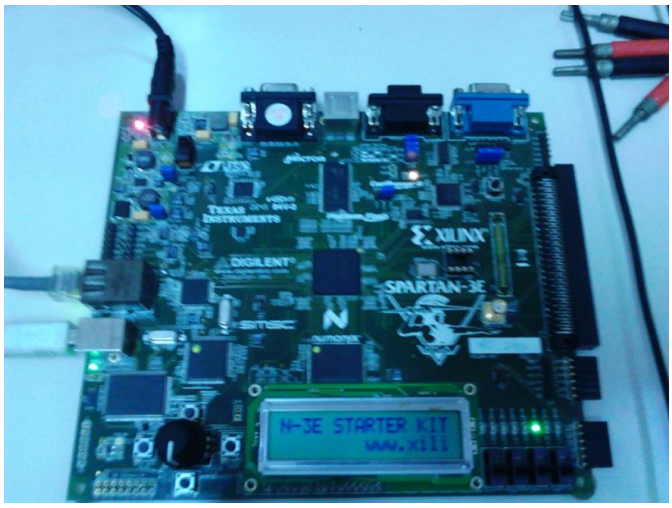

Figura 6. 16 Tarjeta de desarrollo recibiendo datos desde Interface de Usuario.

PRUEBA DEL MÓDULO DAC

Se hacen pruebas de la intensidad de la luz emitida por un led para diferentes valores obtenidos mediante el convertidor DAC. Las instrucciones del procesador requeridas para su evaluación son las siguientes:

```
INTERFACE_MUX_mWriteReg(XPAR_INTERFACE_MUX_0_BASEADDR, 0, 1);
PEDRO_DAC_mWriteReg(XPAR_PEDRO_DAC_0_BASEADDR, 0, data_dac);
```

Se hacen pruebas para los siguientes valores de data_dac: 3200, 3300, 3400, 3500 y 4095.

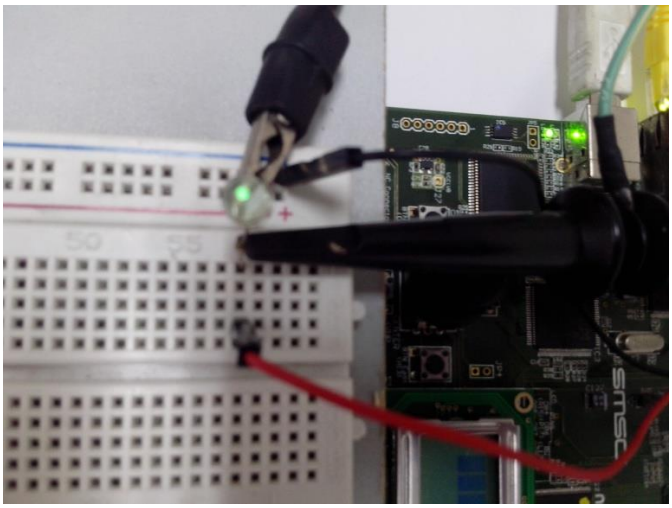

Figura 6. 17 Intensidad de LED para data_dac=3300.

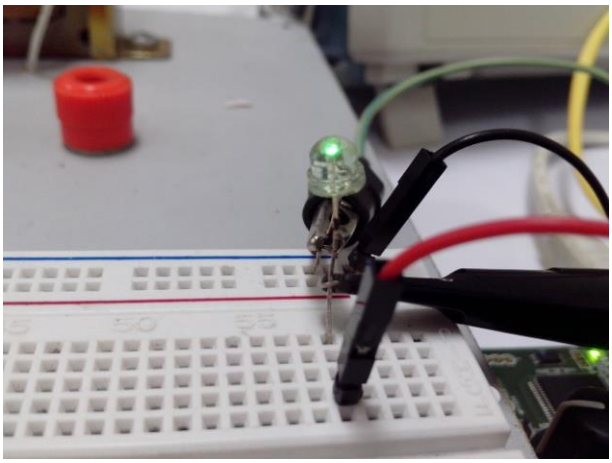

Figura 6. 18 Intensidad de LED para data_dac=3400.

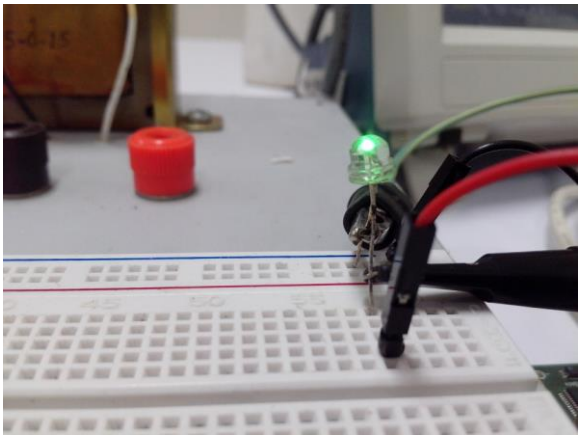

Figura 6. 19 Intensidad de LED para data_dac=3500.

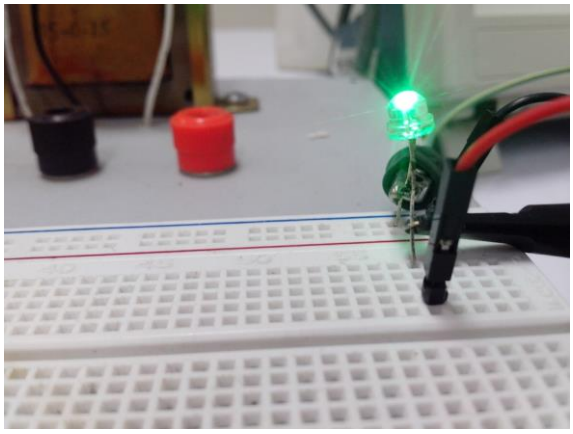

Figura 6. 20 Intensidad de LED para data_dac=4095.

Esta prueba se realizo satisfactoriamente. Otra de las pruebas a realizar es generar una señal triangular, tal como se muestra a continuación:

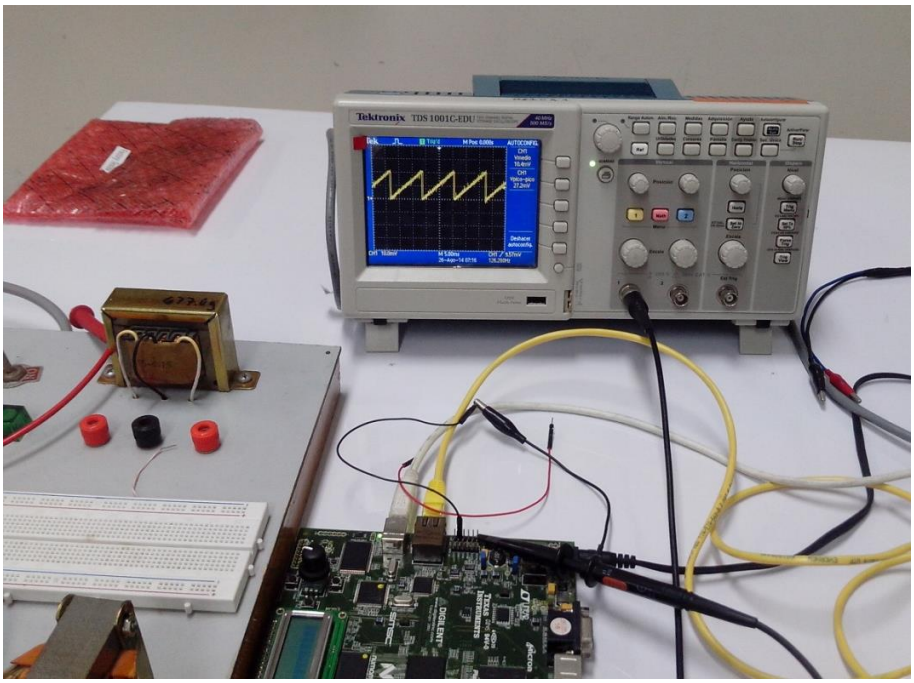

Figura 6. 21 Generación de señal triangular.

PRUEBA DEL MÓDULO ADC

La prueba de este módulo también se realizo de manera exitosa.

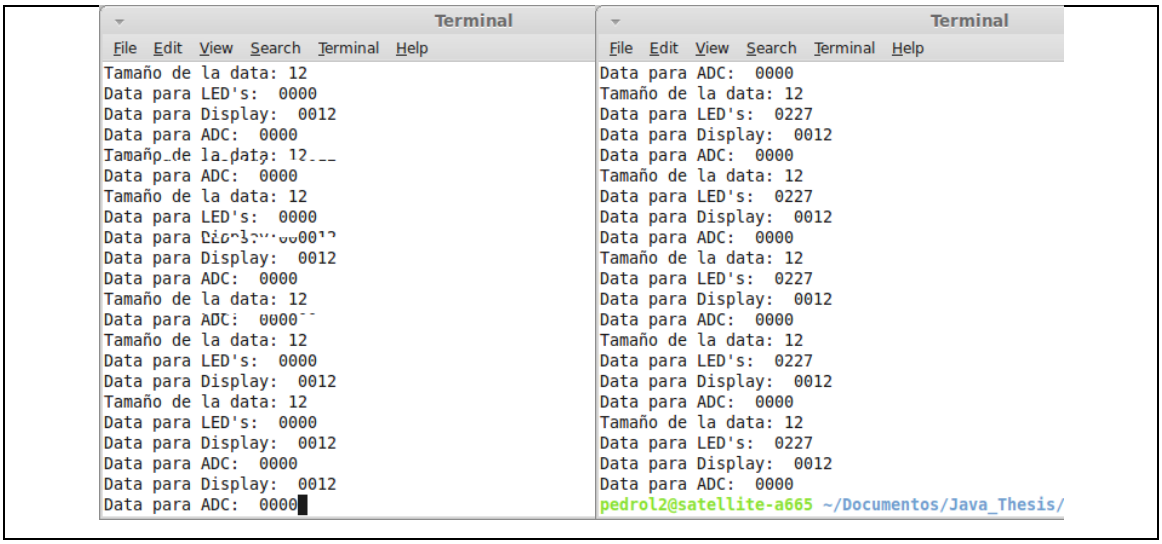

Figura 6. 22 Izquierda, prueba en marcha, derecha, fin de prueba.

Como puede notarse en la parte izquierda de la figura, al encontrarse en marcha el programa algunos números y letras aparecen imcompletos.

EJECUCIÓN DESDE MÓVIL

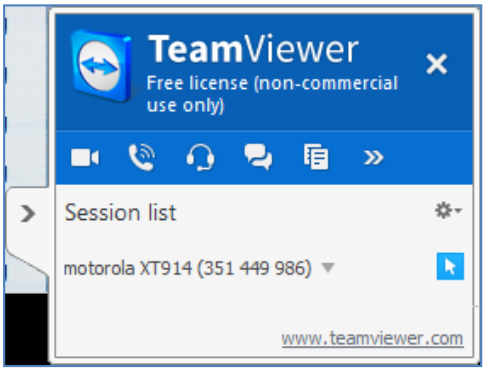

Figura 6. 23 Activación de comunicación con dispositivo celular.

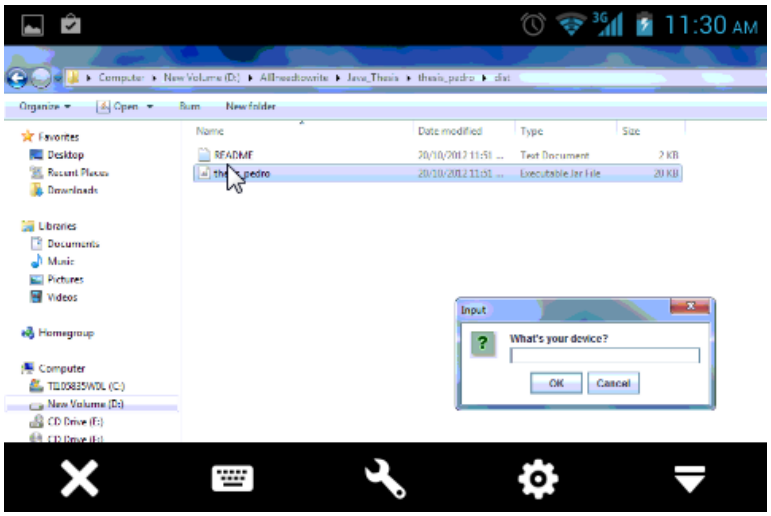

Figura 6. 24 Ubicación de ejecutable *.jar.

Nota: La ejecución del ejecutable *.jar en Windows se realiza con un simple clic, la manera de ejecución en linux ya fue detallada en el capítulo IV (ver Figura 4.31).

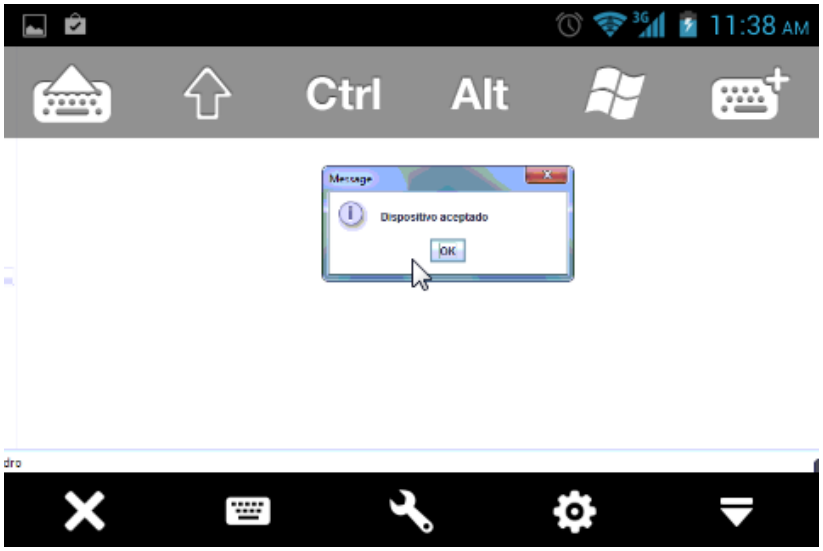

Figura 6. 25 Mensaje luego de ingresar número"1" como código de dispositivo.

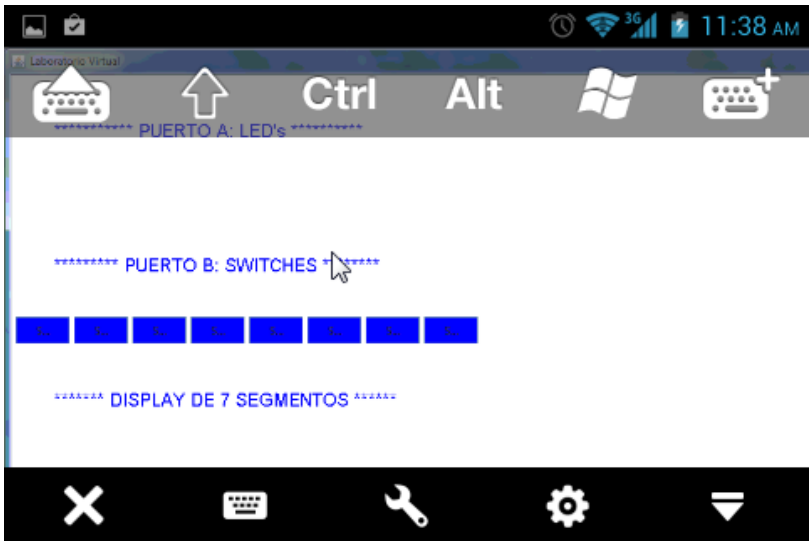

Figura 6. 26 Ejecución satisfactoria desde móvil.

Nota: Figuras 6.23, 6.24 y 6.25 son tomadas de impresión de pantalla en celular.

La prueba del sistema se realizo según el modelo de bajo costo, como se muestra en la Figura 4.32, para este fin se uso la tarjeta de desarrollo para FPGA's DE1 de Altera, como dispositivo de prueba, la comunicación entre módulos se realizo vía

Costos y Presupuestos

Los costos provienen de:

- Tarjeta de desarrollo para FPGA, Spartan 3E Starter Board
- PC host
- Switch

Mejoras Futuras

El sistema se podría adaptar con un servidor y más componentes que nos permitan administrar de una manera más completa.

Se puede añadir una gama de dispositivos por defecto a ser evaluados, en esta primera etapa se hicieron pruebas para un solo dispositivo, pero se tuvo en cuenta esa alternativa, es por ello que siempre en la ejecución de una nueva verificación se debe ingresar un número o código de dispositivo.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1) El sistema desarrollado es transportable (o portable) para FPGA's Xilinx de diferentes densidades que puedan implementar el procesador Microblaze y los módulos periféricos requeridos. Por lo tanto se logra cumplir el objetivo de portabilidad del sistema, para no quedar obsoleto en el tiempo.
2) Trabajar con un procesador embebido permite alcanzar un nivel de abstracción más alto, por lo tanto, para cualquier sistema en diseño, se debe determinar qué parte del sistema se desarrolla en hardware, mediante HDL (lenguaje de descripción de hardware) y qué parte se desarrolla en software.
3) El procesador embebido se puede personzalizar, haciendo uso de los periféricos que requiere el sistema, por lo tanto el sistema permite ahorrar energía y Elementos Lógicos. Esto es debido a que el procesador del tipo Soft-Core usa mayor o menor cantidad de LE's (Elementos Lógicos del FPGA) dependiendo el tipo de diseño.
4) Debido a que el hardware del FPGA es definido por el usuario, podemos obtener la cantidad deseada módulos de cierto tipo (por ejemplo PWM's diseñados en HDL); dónde la única limitante es la cantidad de LE's disponibles en el FPGA.
5) Se logra la comunicación y verificación del sistema. Actualmente la trama de datos enviados y recibidos es pequeña pero debido a las ventajas de reconfigurabilidad del sistema, es posible su escalabilidad para enviar una mayor cantidad de datos, consumiendo un ancho de banda reducido comparado a otros sistemas actuales.
6) Finalmente aunque el sistema implementado no posee todos los elementos que posee un sistema complejo, el cuál contaría con un servidor e interfaces más complejas, éste sistema puede aumentar su capacidad para monitorear otros dispositivos, haciendo uso de puertos libres del módulo FX2.

Recomendaciones

1) Acondicionar el sistema para su uso con un mayor número de dispositivos de prueba. En éste trabajo se realizó la prueba para un solo dispositivo, pero es posible la expansión.
2) Verificar el funcionamiento de cada módulo de comunicación de manera separada, haciendo uso de las instrucciones de procesador generadas por los IP Cores, para pruebas de módulos aislados.
3) Acondicionar adecuadamente las señales y pines de comunicación, ya que el sistema maneja diferentes protocolos de comunicación, y un mal acondicionamiento puede afectar uno u otro módulo de comunicación.
4) Debido aque lo FPGA's, hacen uso de un lenguaje de descripción de hardware estandarizado (VHDL o Verilog). Se recomienda hacer el uso de las IP Core de usuario que pueden migrar a FPGA's de cualquier fabricante.
5) Se recomienda continuar el estudio para acceder de manera remota desde dispositivos móviles con una mayor rápidez, y sin necesidad de internet.

ANEXO A: IP's DE USUARIO

CODIFICACIÓN VHDL - MÓDULO ADC (DISEÑO HARDWARE)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE ----------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                -- Slave interface data bus width
--   C_NUM_REG                   -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                  -- Bus to IP clock
--   Bus2IP_Reset                -- Bus to IP reset
--   Bus2IP_Data                 -- Bus to IP data bus
--   Bus2IP_BE                   -- Bus to IP byte enables
--   Bus2IP_RdCE                 -- Bus to IP read chip enable
--   Bus2IP_WrCE                 -- Bus to IP write chip enable
--   IP2Bus_Data                 -- IP to Bus data bus
--   IP2Bus_RdAck                -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                -- IP to Bus error response
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                : integer              := 32;
    C_NUM_REG                   : integer              := 1
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here

              ADC_SPI_SS_B        : out std_logic;
                    ADC_DAC_CS : out std_logic;
```

```
                ADC_SF_CE0  : out std_logic;
                ADC_FPGA_INIT_B     : out std_logic;
        ------------------------------------------------
                        ADC_SPI_SCK : out  STD_LOGIC;
        ADC_AD_CONV : out  STD_LOGIC;

        SPI_MISO : in  STD_LOGIC;

        ADC_SPI_MOSI : out  STD_LOGIC;
        ADC_AMP_CS : out  STD_LOGIC;

        AMP_SHDN : out  STD_LOGIC;
        AMP_DOUT : in  STD_LOGIC;
                        --mclk: in STD_LOGIC; --Bus2IP_Clk instead of mclk
--                      amplitude1: out STD_LOGIC_VECTOR(13 downto 0);
                        amplitude2: out STD_LOGIC_VECTOR(13 downto 0);
--                      LED : out std_logic_vector(7 downto 0);
                        trigger : out std_logic;


   -- ADD USER PORTS ABOVE THIS LINE ------------------

   -- DO NOT EDIT BELOW THIS LINE ----------------------
   -- Bus protocol ports, do not add to or delete
   Bus2IP_Clk                   : in  std_logic;
   Bus2IP_Reset                 : in  std_logic;
   Bus2IP_Data                  : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
   Bus2IP_BE                    : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
   Bus2IP_RdCE                  : in  std_logic_vector(0 to C_NUM_REG-1);
   Bus2IP_WrCE                  : in  std_logic_vector(0 to C_NUM_REG-1);
   IP2Bus_Data                  : out std_logic_vector(0 to C_SLV_DWIDTH-1);
   IP2Bus_RdAck                 : out std_logic;
   IP2Bus_WrAck                 : out std_logic;
   IP2Bus_Error                 : out std_logic
   -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic

          type state_type is (idle, set_amp, read_adc);
          signal state : state_type := set_amp;
          type state_type_clock is (clock_on, clock_off);
          signal state_clock : state_type_clock := clock_off;
          signal cnt : integer range 0 to 40 := 0;
          signal clk_sample : STD_LOGIC := '0';
          signal gain_set : STD_LOGIC := '0';
          signal amplitude1_buffer : STD_LOGIC_VECTOR(13 downto 0);
```

```
        signal amplitude1_buffer_n : STD_LOGIC_VECTOR(13 downto 0);
        signal amplitude2_buffer : STD_LOGIC_VECTOR(13 downto 0);
        signal gain1 : STD_LOGIC_VECTOR(3 downto 0) := "0001";
        signal gain2 : STD_LOGIC_VECTOR(3 downto 0) := "0001";
        signal risingedge : std_logic := '1';
        signal counter : integer range 0 to 34;
        ---------------------------------------------
        signal LED : std_logic_vector(7 downto 0);
        signal amplitude1: STD_LOGIC_VECTOR(13 downto 0);


  ----------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ----------------------------------------
  signal slv_reg0                 : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel          : std_logic_vector(0 to 0);
  signal slv_reg_read_sel           : std_logic_vector(0 to 0);
  signal slv_ip2bus_data            : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack               : std_logic;
  signal slv_write_ack              : std_logic;

begin

  --USER logic implementation added here

  ----------------------(9/8/12)-----------------------------
  slv_reg0(24 to 31) <= amplitude1_buffer(11 downto 4);
        --slv_reg0(18 to 31) <= amplitude1_buffer(13 downto 0);
        -------------------------------It was before the last change(9/8/12)-------------
                --slv_reg0(20 to 31) <= amplitude1_buffer(11 downto 0);
                ------------------------------------
                        --slv_reg0(19 to 31) <= not amplitude1_buffer(12 downto 0);
                ------------------------------------
        ---------------------------------------------------------------


-----------------------------------------------------------------
  ---------------------------------------------------------------------------------------

                ADC_SPI_SS_B          <= '1';
                ADC_DAC_CS <= '1';
                ADC_SF_CE0 <= '1';
                ADC_FPGA_INIT_B     <= '1';
-- 1.5MHz clock
                clock_divider : process (Bus2IP_Clk)
                begin
                  if(rising_edge(Bus2IP_Clk)) then
                        if(counter = 33) then
                                risingedge <= risingedge xor '1';
                                clk_sample <= clk_sample xor '1';
                                counter <= 0;
                        else
                                counter <= counter + 1;
                        end if;
                end if;
                end process;

                sclk_clock : process(Bus2IP_Clk)
                begin
                        if(rising_edge(Bus2IP_Clk)) then
```

```
                        case state_clock is
                                when clock_on =>
                                ADC_SPI_SCK <= clk_sample;
                                when clock_off =>
                                ADC_SPI_SCK <= '0';
                                end case;

                        end if;
                end process;

        main : process (Bus2IP_Clk)
        begin
        if(rising_edge(Bus2IP_Clk)) then
                if(counter = 33 and risingedge = '1') then
                --if(risingedge = '1') then
        -- Set gain at -1
        case state is
        when set_amp =>
        ADC_AMP_CS <= '0';
        AMP_SHDN <= '0';
                if (cnt < 4) then
                ADC_SPI_MOSI <= gain2(3 - cnt);
                cnt <= cnt + 1;
                state <= set_amp;
                state_clock <= clock_on;
                elsif (cnt > 3 and cnt < 8) then
                ADC_SPI_MOSI <= gain1(7 - cnt);
                cnt <= cnt + 1;
                state <= set_amp;
                elsif (cnt = 8) then
                cnt <= 0;
                ADC_AMP_CS <= '1';
                state <= idle;
                end if;

        when idle =>
        -- 13ns delay
                if (cnt < 2) then
                ADC_AD_CONV <= '1';
                cnt <= cnt + 1;
                state <= idle;
                elsif (cnt = 2) then
                ADC_AD_CONV <= '0';
                cnt <= 0;
                state <= read_adc;
                trigger <= '0';
                end if;

        when read_adc =>
                if (cnt < 2) then
                cnt <= cnt + 1;
                state <= read_adc;
                state_clock <= clock_on;
                elsif (cnt > 1 and cnt < 16) then
                amplitude1_buffer(15 - cnt) <= SPI_MISO;
                cnt <= cnt + 1;
                state <= read_adc;
                elsif (cnt > 15 and cnt < 18) then
                cnt <= cnt + 1;
```

```
                    state <= read_adc;
                    elsif (cnt > 17 and cnt < 32) then
                    amplitude2_buffer(31 - cnt) <= SPI_MISO;
                    cnt <= cnt + 1;
                    state <= read_adc;
                    elsif (cnt > 31 and cnt < 34) then
                    cnt <= cnt + 1;
                    state <= read_adc;
                    elsif (cnt = 34) then
                    cnt <= 0;

                    amplitude1<=amplitude1_buffer;
                    amplitude2<=amplitude2_buffer;

                    state_clock <= clock_off;
                    state <= idle;
                    trigger <= '1';
                    end if;
                    end case;
                    end if;
                    end if;
          end process;

  ------------------------------------------
  -- Example code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- The example code presented here is to show you one way of reading/writing
  -- software accessible registers implemented in the user logic slave model.
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
  --
  --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                "1000"   C_BASEADDR + 0x0
  --                "0100"   C_BASEADDR + 0x4
  --                "0010"   C_BASEADDR + 0x8
  --                "0001"   C_BASEADDR + 0xC
  --
  ------------------------------------------
  slv_reg_write_sel <= Bus2IP_WrCE(0 to 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(0 to 0);
  slv_write_ack     <= Bus2IP_WrCE(0);
  slv_read_ack      <= Bus2IP_RdCE(0);

  -- implement slave model software accessible register(s)
--  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
--  begin
--
--    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
--      if Bus2IP_Reset = '1' then
--        slv_reg0 <= (others => '0');
--      else
--        case slv_reg_write_sel is
--          when "1" =>
--            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
--              if ( Bus2IP_BE(byte_index) = '1' ) then
--                slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
```

```
byte_index*8+7);
--              end if;
--            end loop;
--          when others => null;
--        end case;
--      end if;
--    end if;
--
--  end process SLAVE_REG_WRITE_PROC;

  -- implement slave model software accessible register(s) read mux
  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0 ) is
  begin

    case slv_reg_read_sel is
      when "1" => slv_ip2bus_data <= slv_reg0;
      when others => slv_ip2bus_data <= (others => '0');
    end case;

  end process SLAVE_REG_READ_PROC;

  ---------------------------------------
  -- Example code to drive IP to Bus signals
  ---------------------------------------
  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                  (others => '0');

  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
  IP2Bus_Error <= '0';

end IMP;
```

CODIFICACIÓN VHDL - MÓDULO DAC (DISEÑO HARDWARE)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                -- Slave interface data bus width
--   C_NUM_REG                   -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                  -- Bus to IP clock
```

```
--   Bus2IP_Reset            -- Bus to IP reset
--   Bus2IP_Data             -- Bus to IP data bus
--   Bus2IP_BE               -- Bus to IP byte enables
--   Bus2IP_RdCE             -- Bus to IP read chip enable
--   Bus2IP_WrCE             -- Bus to IP write chip enable
--   IP2Bus_Data             -- IP to Bus data bus
--   IP2Bus_RdAck            -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck            -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error            -- IP to Bus error response
------------------------------------------------------------------------------
entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                : integer             := 32;
    C_NUM_REG                   : integer             := 1
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
                    --señales de reloj y reset--
                    --clk_50                   : in std_logic;--c9, ahora este será Bus2IP_Clk
                    reset                     : in std_logic;--l13
                    led                       : out std_logic; --f12
                    ------------------------------------------------------------
                    --Aquí van las señales que se tienen que reemplazar para la simulación

                    --señales de la interfaz SPI para el DAC
                    DAC_SPI_MOSI        : out std_logic;--t4
                    DAC_DAC_CS          : out std_logic;--n8
                    DAC_SPI_SCK         : out std_logic;--u16

                    DAC_CLR                     : out std_logic;--p8
                    --SPI_MISO              : in std_logic;--n10

                    --adicionar señales para desactivar otros dispositivos SPI
                    DAC_SPI_SS_B        : out std_logic;--"1"
                    DAC_AMP_CS          : out std_logic;--"1"
                    DAC_AD_CONV                 : out std_logic;--"0"
                    DAC_SF_CE0          : out std_logic;--"1"
                    DAC_FPGA_INIT_B     : out std_logic;--"1"
                    -----Data a ingresar al sistema ----------
                    --sw_3_1                   : in std_logic_vector(2 downto 0)--switches desde el 3 al 1

    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                  : in  std_logic;--ahora vamos a usar este clock en lugar de clk_50
    Bus2IP_Reset                : in  std_logic;
    Bus2IP_Data                 : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
                                   --ahora vamos a usar esta data en lugar de sw_3_1
```

```
    Bus2IP_BE                   : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                 : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                 : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                : out std_logic;
    IP2Bus_WrAck                : out std_logic;
    IP2Bus_Error                : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
type state_type is (read_dac, write_dac);-- the third state was not necessary(set_data)
signal state : state_type := read_dac;
signal sclk        : std_logic:='0'; --señal para generar el clock del sistema SPI
--signal command         : std_logic_vector(3 downto 0);
--señal para mandar el comando "0011" vía SPI, el cual hará funcionar nuestro sistema
--signal address           : std_logic_vector(11 downto 0);
--señal para mandar la dirección que selecciona el canal a usar para el ADC

--DAC A: "0000"           --DAC B: "0001" --DAC C: "0010" --DAC D: "0011" -- ALL: "1111"
signal count      : integer range 0 to 32 := 0;
signal s: std_logic_vector(23 downto 0); --Señal para generar el sclk a partir del clk_50
--signal counter_out: std_logic_vector(11 downto 0);
signal counter_in          : std_logic_vector(11 downto 0):= X"000";

  ---------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ---------------------------------------------
  signal slv_reg0                   : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel            : std_logic_vector(0 to 0);
  signal slv_reg_read_sel             : std_logic_vector(0 to 0);
  signal slv_ip2bus_data              : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack                 : std_logic;
  signal slv_write_ack                : std_logic;

begin

  --USER logic implementation added here

          --Primero deshabilitamos los otros dispositivos que comparten con el DAC la interface SPI
          DAC_SPI_SS_B <= '1';
          DAC_AMP_CS  <= '1';
          DAC_AD_CONV          <= '0';
          DAC_SF_CE0   <= '1';
          DAC_FPGA_INIT_B <= '1';

clock:    process(Bus2IP_Clk, reset)
```

```
        begin
        if(reset='1') then
                sclk <= '0';
        elsif(Bus2IP_Clk'event and Bus2IP_Clk='1') then
                s <= s+1;
                if(s=x"FFFFFF") then --20 = X"14"
                        s <= x"000000";
                        sclk <= not sclk;
                end if;
        end if;
        end process clock;
        ----------Signals----------
        led <= sclk;
        DAC_SPI_SCK <= not Bus2IP_Clk;
        DAC_CLR <= not reset;
        --------------------------------
dac:    process(reset, Bus2IP_Clk)
        begin
        if(reset='1') then
                counter_in <= X"000";
        elsif(Bus2IP_Clk'event and Bus2IP_Clk='1') then
                count <= count+1;
        case state is
                ------------------------------State for Reading---------------------------------
                when read_dac =>
                        --DAC_CS <= '1';
                        --counter_in <= sw_3_1&sw_3_1&sw_3_1&sw_3_1;
                         --Con esta linea cargamos los datos;
                        --Comentamos y modificamos línea superior--
                        counter_in <= slv_reg0(20 to 31);
                        --Cambiamos sw_3_1 por la data del uBlaze
                        state <= write_dac;   --Going to the State of writing

                ------------------------------State for Writing---------------------------------
                when write_dac =>
                        DAC_DAC_CS <= '0';
                        if(count <= 7) then
                -- 8 don't care(SPI_MOSI)
                                state <= write_dac;
                                elsif (8<=count and count<= 9) then
                                -- Sending Command '0011', this command is the most known
                                        DAC_SPI_MOSI <= '0';
                                -- because it refreshes the data of the output when it is received
                                        state <= write_dac;
                                elsif (10 <= count and count <= 11) then
                                -- The Command '0011' is still sent
                                        DAC_SPI_MOSI <= '1';
                                        state <= write_dac;
                                elsif (12<=count and count<=15) then
                                -- Send Address to select the output channels
                                        DAC_SPI_MOSI <= '1';
                                --'0' solo canal A, '1' todos los canales
                                        state <= write_dac;
                                elsif(15< count and count < 28) then
                                -- output: 12 Databits (MSB first)
                                        DAC_SPI_MOSI <= counter_in(27-count);
                                        state <= write_dac;
                                elsif(count > 27 and count < 32) then
                                -- output: 4 don't care(SPI_MOSI)
```

```
                                        state <= write_dac;
                                    else
                                        count <= 0;
                                    -- return to write mode
                                        DAC_DAC_CS <= '1';
                                        state <= read_dac;
                                     -- Going to state of reading
                                    end if;
                                    ------------------------------------------------------------------------------
          end case;
          end if;
          end process dac;
  ------------------------------------------
  -- Example code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- The example code presented here is to show you one way of reading/writing
  -- software accessible registers implemented in the user logic slave model.
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
  --
  --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                "1000"   C_BASEADDR + 0x0
  --                "0100"   C_BASEADDR + 0x4
  --                "0010"   C_BASEADDR + 0x8
  --                "0001"   C_BASEADDR + 0xC
  --
  ------------------------------------------
  slv_reg_write_sel <= Bus2IP_WrCE(0 to 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(0 to 0);
  slv_write_ack     <= Bus2IP_WrCE(0);
  slv_read_ack      <= Bus2IP_RdCE(0);

  -- implement slave model software accessible register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        slv_reg0 <= (others => '0');
      else
        case slv_reg_write_sel is
          when "1" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              end if;
            end loop;
          when others => null;
        end case;
      end if;
    end if;

  end process SLAVE_REG_WRITE_PROC;

  -- implement slave model software accessible register(s) read mux
```

```
--  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0 ) is
--  begin
--
--    case slv_reg_read_sel is
--      when "1" => slv_ip2bus_data <= slv_reg0;
--      when others => slv_ip2bus_data <= (others => '0');
--    end case;
--
--  end process SLAVE_REG_READ_PROC;
  ---------------------------------------
  -- Example code to drive IP to Bus signals
  ---------------------------------------
--  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
--                  (others => '0');

  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
  IP2Bus_Error <= '0';

end IMP;
```

CODIFICACIÓN VHDL – MÓDULO MUX (DISEÑO HARDWARE)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                -- Slave interface data bus width
--   C_NUM_REG                   -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                  -- Bus to IP clock
--   Bus2IP_Reset                -- Bus to IP reset
--   Bus2IP_Data                 -- Bus to IP data bus
--   Bus2IP_BE                   -- Bus to IP byte enables
--   Bus2IP_RdCE                 -- Bus to IP read chip enable
--   Bus2IP_WrCE                 -- Bus to IP write chip enable
--   IP2Bus_Data                 -- IP to Bus data bus
--   IP2Bus_RdAck                -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                -- IP to Bus error response
------------------------------------------------------------------------------

entity user_logic is
```

```
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                : integer              := 32;
    C_NUM_REG                   : integer              := 1
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
          ----------------------------------
          SPI_MOSI      : out std_logic;
          SPI_SCK               : out std_logic;
          ----------------------------------
    AD_CONV    : out std_logic;
          DAC_CS                : out std_logic;
          SPI_SS_B      : out std_logic;
    FPGA_INIT_B: out std_logic;
    AMP_CS                : out std_logic;
          SF_CE0                : out std_logic;
          ----------------------------------------------------
          ADC_SPI_MOSI          : in std_logic;
          ADC_SPI_SCK           : in std_logic;
          ----------------------------------
    ADC_AD_CONV       : in std_logic;
          ADC_DAC_CS            : in std_logic;
          ADC_SPI_SS_B          : in std_logic;
    ADC_FPGA_INIT_B : in std_logic;
    ADC_AMP_CS        : in std_logic;
          ADC_SF_CE0            : in std_logic;
          ----------------------------------------------------
          DAC_SPI_MOSI          : in std_logic;
          DAC_SPI_SCK           : in std_logic;
          ----------------------------------
    DAC_AD_CONV       : in std_logic;
          DAC_DAC_CS            : in std_logic;
          DAC_SPI_SS_B          : in std_logic;
    DAC_FPGA_INIT_B : in std_logic;
    DAC_AMP_CS        : in std_logic;
          DAC_SF_CE0            : in std_logic;

          ------------------------------------------------------------
          --adc_n_dac             : in std_logic; --Now it must be as a signal
          ------------------------------------------------------------
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ----------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                  : in  std_logic;
    Bus2IP_Reset                  : in  std_logic;
    Bus2IP_Data                 : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                   : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                   : in  std_logic_vector(0 to C_NUM_REG-1);
```

```
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ----------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic

  signal adc_n_dac: std_logic; -- Signal to make a mux

  ---------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ---------------------------------------
  signal slv_reg0                  : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel         : std_logic_vector(0 to 0);
  signal slv_reg_read_sel          : std_logic_vector(0 to 0);
  signal slv_ip2bus_data           : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack              : std_logic;
  signal slv_write_ack             : std_logic;

begin

  --USER logic implementation added here

          adc_n_dac <= slv_reg0(31);

          -----*****************************SECOND ATTEMPT************************-----
          with adc_n_dac select
                  SPI_MOSI <= ADC_SPI_MOSI when '0',
                                                  DAC_SPI_MOSI when others;
          with adc_n_dac select
                  SPI_SCK         <=      ADC_SPI_SCK when '0',
                                                  DAC_SPI_SCK when others;
          with adc_n_dac select
                  AD_CONV         <= ADC_AD_CONV when '0',
                                                  DAC_AD_CONV when others;
          with adc_n_dac select
                  DAC_CS          <=      ADC_DAC_CS when '0',
                                                  DAC_DAC_CS when others;
          with adc_n_dac select
                  SPI_SS_B <= ADC_SPI_SS_B when '0',
                                                  DAC_SPI_SS_B when others;
          with adc_n_dac select
                  FPGA_INIT_B <= ADC_FPGA_INIT_B when '0',
                                                          DAC_FPGA_INIT_B when others;
```

```
        with adc_n_dac select
                AMP_CS          <= ADC_AMP_CS when '0',
                                        DAC_AMP_CS when others;

        with adc_n_dac select
                SF_CE0          <= ADC_SF_CE0 when '0',
                                        DAC_SF_CE0 when others;
        --------************************************************************----------

----------***************FIRST ATTEMPT*************************----------
--  process(adc_n_dac, Bus2IP_Clk)
--  begin
--
--  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
--
--  case adc_n_dac is
--
--          when '0' =>
--           ---------------------------------------------------
--           SPI_MOSI <= ADC_SPI_MOSI;
--           SPI_SCK <= ADC_SPI_SCK;
--           ---------------------------------
--    AD_CONV <= ADC_AD_CONV;
--           DAC_CS <= ADC_DAC_CS;
--           SPI_SS_B <= ADC_SPI_SS_B;
--    FPGA_INIT_B <= ADC_FPGA_INIT_B;
--    AMP_CS <= ADC_AMP_CS;
--           SF_CE0 <= ADC_SF_CE0;
--
--           when others =>
--           ---------------------------------------------------
--           SPI_MOSI <= DAC_SPI_MOSI;
--           SPI_SCK <= DAC_SPI_SCK;
--           ---------------------------------
--    AD_CONV <= DAC_AD_CONV;
--           DAC_CS <= DAC_DAC_CS;
--           SPI_SS_B <= DAC_SPI_SS_B;
--    FPGA_INIT_B <= DAC_FPGA_INIT_B;
--    AMP_CS <= DAC_AMP_CS;
--           SF_CE0 <= DAC_SF_CE0;
----        when others =>
----                SPI_MOSI <= '0';
--          end case;
--          end if;
--          end process;
---------*********************************************************--------



  ------------------------------------------
  -- Example code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- The example code presented here is to show you one way of reading/writing
  -- software accessible registers implemented in the user logic slave model.
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
  --
```

```
  --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                    "1000"   C_BASEADDR + 0x0
  --                    "0100"   C_BASEADDR + 0x4
  --                    "0010"   C_BASEADDR + 0x8
  --                    "0001"   C_BASEADDR + 0xC
  --
  ------------------------------------------
  slv_reg_write_sel <= Bus2IP_WrCE(0 to 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(0 to 0);
  slv_write_ack     <= Bus2IP_WrCE(0);
  slv_read_ack      <= Bus2IP_RdCE(0);

  -- implement slave model software accessible register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        slv_reg0 <= (others => '0');
      else
        case slv_reg_write_sel is
          when "1" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              end if;
            end loop;
          when others => null;
        end case;
      end if;
    end if;

  end process SLAVE_REG_WRITE_PROC;

  -- implement slave model software accessible register(s) read mux
--  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0 ) is
--  begin
--
--    case slv_reg_read_sel is
--      when "1" => slv_ip2bus_data <= slv_reg0;
--      when others => slv_ip2bus_data <= (others => '0');
--    end case;
--
--  end process SLAVE_REG_READ_PROC;
  ------------------------------------------
  -- Example code to drive IP to Bus signals
  ------------------------------------------
  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                  (others => '0');

  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
  IP2Bus_Error <= '0';

end IMP;
```

ANEXO B: SOFTWARE DEL PROCESADOR EMBEBIDO

TestApp_Memory.c

```
// Located in: microblaze_0/include/xparameters.h
#include "xparameters.h"          //Contiene los parametros y direcciones de memoria de los driver
#include "stdio.h"                        //Para el uso RS232 como E/S del microblaze
#include "xutil.h"
/********************IP'S DEL CATALOGO*************************/
#include "xemaclite.h"              //Para el uso de la MAC ethernet
#include "string.h"
#include "xgpio_l.h"
/*********************OTROS ARCHIVOS*************************/
#include "helper.h"
#include "eth_udp_buf.h"
#include "estructuras.h"
/*********************IP'S DE USUARIO*************************/
#include "pedro_adc.h"
#include "pedro_dac.h"
#include "interface_mux.h"
/*****************************************************************/
Xuint8 instruccion=0x00;
Xuint32 DataReceived = 0x00000000;

/***Creacion de Buffer de memoria para la transmision y recepcion de paquetes ethernet***/
volatile Xuint32 trama_Tx[LONG_MAX_TRAMA_32BITS];
volatile Xuint32 trama_Rx[LONG_MAX_TRAMA_32BITS];
/*****************************************************************************************/
        Xuint8 led_i=0;
        Xuint8 leer_adc=0;
        Xuint8 data_adc=0;//int leer_adc=0;
        Xuint8 data_dac=0;
        Xuint8 select_mux=0;

/******************CODIGO PARA TESTEO DE LOS LEDS(VARIABLES)*********************/
        int count=0;
        int led_adc=0;
/******************CODIGO PARA TESTEO DE LOS LEDS(VARIABLES)*********************/

/*****************************************************************************************/
Xuint8 d[12]={   0x30, 0x30, 0x30, 0x30,  //leds
                                          0x30, 0x30, 0x30, 0x30,  //display
                                          0x30, 0x30, 0x30,0x30};  //adc

                //d[3], d[2], d[1] and d[0]  are used to give data to the leds
                //d[7], d[6], d[5] and d[4]  are used to give data to the display
                //d[11], d[10], d[9] and d[8]  are used to give data to the adc
/*****************************************************************************************/
Xuint8 to_displays = 0x00;
Xuint8 to_leds=0x00;
Xuint8 to_adc=0x00;
/*****************************************************************************************/

int converter(int data, int indice){

        Xuint8 mi=0;  //millares
        Xuint8 ce=0;  //centenas
        Xuint8 de=0;  //decenas
```

```
	Xuint8 un=0;  //unidades
	Xuint8 rm=0;  //resto de millares
	Xuint8 rc=0;  //resto de centenas

	mi = data/1000;
	rm = data%1000;

	ce = rm/100;
	rc = rm%100;

	de = rc/10;
	un = rc%10;

	//if(data>999) {
	d[indice+3]= 0x30+mi;
	d[indice+2]= 0x30+ce;
	d[indice+1]= 0x30+de;
	d[indice+0]= 0x30+un;
	//}
	/*else {
	d[indice+2]= 0x30+ce;
	d[indice+1]= 0x30+de;
	d[indice+0]= 0x30+un;
	}*/

	return d;
}

int main (void) {
	/**********Inicializacion para el manejo XEmacLite**********/
	static XEmacLite EmacLiteInstance;
	XEmacLite *EmacLiteInstancePtr = &EmacLiteInstance;
	inicializacion_xemaclite( EmacLiteInstancePtr );
	/***Creacion de un puntero al inicio del buffer de memoria creado para la transmision***/
	Xuint8 *TxBuf_p;
	TxBuf_p = (Xuint8 *)trama_Tx;
	/*** Creacion de un puntero al inicio del buffer de memoria creado para la recepcion ***/
	Xuint8 *RxBuf_p;
	RxBuf_p = (Xuint8 *)trama_Rx;
  int long_msje=0,long_trama=0;
	unsigned long x;

	Xuint8 data_test=0;
	Xuint8 dac_on=0;

	while(1){
	/*********************************DAC IS WORKING************************/
		//*
		if (data_test == 4096) {
		data_dac--;
		}
		else {
		data_test = data_dac;
		data_dac++;
		}
		INTERFACE_MUX_mWriteReg(XPAR_INTERFACE_MUX_0_BASEADDR, 0, 1);
		PEDRO_DAC_mWriteReg(XPAR_PEDRO_DAC_0_BASEADDR, 0, data_dac);
		//*/
	/*****************************************************************************/
```

```
	/*********************************ADC IS WORKING*************************/
	/*
	if(dac_on==0){
		INTERFACE_MUX_mWriteReg(XPAR_INTERFACE_MUX_0_BASEADDR, 0, 0);
		//data_adc=PEDRO_ADC_mReadReg(XPAR_PEDRO_ADC_0_BASEADDR, 0);
		//XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,data_adc);
		xil_printf("DATA ADC: %d\n\r",data_adc);
		//to_adc = XGpio_mReadReg( XPAR_DIP_SWITCHES_4BIT_BASEADDR, 0)*5;
		to_adc = PEDRO_ADC_mReadReg(XPAR_PEDRO_ADC_0_BASEADDR, 0);
		converter(to_adc, 8);
	}
	*/
	/*****************************************************************************/

	/*****************************25/05/12*************************/
	//XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,DataReceived);
	/*************************************************************/
leer_adc=0;
	/******************CODIGO PARA TESTEO DE LOS LEDS*********************/
	/*
	if(count==100){
		count=0;
		to_leds++;
		if(to_leds==255){
			to_leds=0;
		}
	}
	else{
		count++;
	}
	*/
	to_leds = 5*XGpio_mReadReg( XPAR_DIP_SWITCHES_4BIT_BASEADDR, 0);
//to_leds=255;
	//to_leds=PEDRO_ADC_mReadReg(XPAR_PEDRO_ADC_0_BASEADDR, 0);
	converter(to_leds, 0);
	/******************CODIGO PARA TESTEO DE LOS LEDS*********************/

	/*******************************************************************************/
		//to_adc = to_leds;
		//converter(to_adc, 8);
	/*******************************************************************************/

	/*************CODIGO PARA TESTEO DE LOS DISPLAYS*********************/
	to_displays = XGpio_mReadReg( XPAR_DIP_SWITCHES_4BIT_BASEADDR, 0)+12;
	converter(to_displays, 4);
	/*****************************************************************/

		if(leer_adc==0){

//Xuint8 mesg_buf[] = {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30};
		Xuint8 mesg_buf[] = {d[11],d[10],d[9],d[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]};

				long_msje = strlen( mesg_buf );		//uso del String.h para
medir el tamaño del mensaje
				//	xil_printf("Longitud_Mensaje: %d\n\r",long_msje);
				long_trama = construccion_trama( mesg_buf, long_msje,
TxBuf_p );
				//	xil_printf("Longitud_Trama: %d\n\r",long_trama);
```

```
/***************************Envia Instrucción*******************************/
                                    XEmacLite_Send(EmacLiteInstancePtr, (Xuint8
*)trama_Tx,(unsigned)long_trama);
/*********************Recibe instruccion*************************************/
                                    if(instruccion=XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8
*)trama_Rx)!=0x00){
                                    //if(XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8
*)trama_Rx)!=0x00){
                                    //xil_printf("Instruccion: %x\n\r",instruccion);
                                    //instruccion=XEmacLite_Recv(EmacLiteInstancePtr, (Xuint8
*)trama_Rx);
                                                instruccion = analisis_trama( RxBuf_p );
//          xil_printf("Instruccion: %x\n\r",instruccion);
                                    }
//xil_printf("Instruccion: %d\n\r",instruccion);
                                    if (instruccion==0x01){
                                            //envia MAC usando ARP
                                            long_trama=respuesta_arp(TxBuf_p);
                                            XEmacLite_Send(EmacLiteInstancePtr, (Xuint8
*)trama_Tx,(unsigned)long_trama);
                                    }
                                if (instruccion==0x05){
                                    //establece Data Recibida

          XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,DataReceived);
          //xil_printf("LLEGUE A ESTA LINEA");
                                  //to_leds = DataReceived; //PARA TESTEAR LOS LEDS VIRTUALES
AUTOMATICAMENTE
                                }
                                }
          }
   return 0;
}
```

helper.c

```
#ifndef HELPER_H                /* prevent circular inclusions */
#define HELPER_H                /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

// ****************************************************************************
//
// Helper Functions
//
// ****************************************************************************
// Check buffers to see if they Match
// returns 0 if the first n values of the buffers are the same
// returns -1 if the values are not the same
//

int chk_buf_match( void *buf1, void *buf2, int n ) {
unsigned char *p1 = (unsigned char *)buf1;
unsigned char *p2 = (unsigned char *)buf2;
while(n--) {
```

```
if( p1[n] != p2[n] )
return -1;
}
return 0;
}

// Copy first n values from source buffer to destination buffer
// returns 0
// Its is the callers responsibility to make sure that the pointers do not over
// run the buffers.
//

void copiar_datos( void *org, void *dst, int n ) {

        Xuint8 *org_p = (Xuint8 *)org;
        Xuint8 *dst_p = (Xuint8 *)dst;
        while(n--) {
                dst_p[n] = org_p[n];
        }
}

// ******************************************************************************
//
// CheckSum Function
//
// ******************************************************************************
// CheckSum 16 bit
// Add together the inital 'chksum' value and the next 'len' 16 bit values
// starting from the address 'buf1'.
// returns the final value.
// If the 16 bit addition overflows then add the overflow back into
// the checksum value.
// To produce the One's compliment the caller needs to compliment the returned value.
//

int chksum16( void *buf1, short len, int chksum ) {

        unsigned short *buf = buf1;
        int chksum16;
        while(len>0) {
                if (len==1)
                        chksum16 = ((*buf) & 0xFF00 );
                else
                        chksum16 = (*buf);
                chksum += chksum16;
                *buf++;
                len -= 2;
        }
        while( chksum >> 16 )
                chksum = (chksum & 0x0FFFF) + (chksum >> 16);
        return chksum;
}


#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

eth_udp_buf.h

```
#ifndef ETH_UDP_BUF_H		/* prevent circular inclusions */
#define ETH_UDP_BUF_H		/* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

#include "estructuras.h"
#include "helper.h"
#include "xparameters.h"//contiene los parametros y direcciones de memoria de los driver

extern Xuint32 DataReceived;

//*************************************************************
void inicializacion_xemaclite( XEmacLite *EmacLiteInstancePtr ) {
	XEmacLite_Initialize(EmacLiteInstancePtr, XPAR_ETHERNET_MAC_DEVICE_ID);
	XEmacLite_SetMacAddress(EmacLiteInstancePtr, org_mac_addr);
	XEmacLite_FlushReceive(EmacLiteInstancePtr);
}
//Funcion para el envio de MAC usando ARP
Xuint16 respuesta_arp( Xuint8 *buf ) {

	Xuint8 *buf_p;

	//Establecimiento de la trama ARP
	buf_p = buf + SIZE_ETHERNET_HEADER ;
	struct trama_arp *aTr_p;
	aTr_p = (struct trama_arp *)buf_p ;
	aTr_p->hw_type   = 0x0001;
	aTr_p->prot_type = 0x0800;
	aTr_p->hw_size   = 0x06;
	aTr_p->prot_size = 0x04;
	aTr_p->opcode    = 0x0002;
	copiar_datos( org_mac_addr, aTr_p->mac_org, 6 );
	copiar_datos( org_ip_addr , aTr_p->ip_org , 4 );
	copiar_datos( dst_mac_addr, aTr_p->mac_dst, 6 );
	copiar_datos( dst_ip_addr , aTr_p->ip_dst , 4 );

	//Establecimiento de la cabecera Ethernet
	buf_p = buf ;
	struct trama_eth *eTr_p;
	eTr_p = (struct trama_eth *)buf_p;
	eTr_p->type_length = 0x0806;
	copiar_datos( dst_mac_addr, eTr_p->mac_dst, 6 );
	copiar_datos( org_mac_addr, eTr_p->mac_org, 6 );

	return SIZE_ETHERNET_HEADER+SIZE_ARP_BODY;
}

//*************************************************************
int construccion_trama( Xuint8 *buf, int abuf_len, Xuint8 *rpybuf ) {

	//creacion de puntero a la trama
	Xuint8 *rbuf_p;
	int chksum2, reply_len;
	rbuf_p = rpybuf + SIZE_ETHERNET_HEADER + SIZE_IP_HEADER +
SIZE_UDP_HEADER ;
```

```
        copiar_datos( buf, rbuf_p, abuf_len );
        reply_len = abuf_len;
        //Establecimiento de la cabecera UDP
        struct trama_udp *uTr_p;
        rbuf_p = rpybuf + SIZE_ETHERNET_HEADER + SIZE_IP_HEADER ;
        uTr_p = (struct trama_udp *)rbuf_p;
        uTr_p->udp_org = org_puerto;
        uTr_p->udp_dst = dst_puerto;
        uTr_p->udp_len = reply_len + SIZE_UDP_HEADER;
        uTr_p->udp_chksum = 0;
        reply_len += SIZE_UDP_HEADER;
        //Establecimiento de la cabecera IP
        struct trama_ip *iTr_p;
        rbuf_p = rpybuf + SIZE_ETHERNET_HEADER ;
        iTr_p = (struct trama_ip *)rbuf_p ;
        iTr_p->v_hl_tos = 0x4500;
        iTr_p->len =reply_len + SIZE_IP_HEADER;
        iTr_p->id = 0x0000;
        iTr_p->offset = 0x0000;
        iTr_p->ttl_proto = 0x1111;
        iTr_p->chksum = 0;
        chksum2 = chksum16( rbuf_p, SIZE_IP_HEADER, 0 );
        chksum2 = (~chksum2) & 0xffff;
        iTr_p->chksum = chksum2;
        reply_len += SIZE_IP_HEADER;
        copiar_datos( org_ip_addr, iTr_p->ip_org, 4 );
        copiar_datos( dst_ip_addr, iTr_p->ip_dst, 4 );

        //Establecimiento de la cabecera Ethernet
        struct trama_eth *eTr_p;
        rbuf_p = rpybuf ;
        eTr_p = (struct trama_eth *)rbuf_p;
        eTr_p->type_length = 0x0800;
        copiar_datos( dst_mac_addr, eTr_p->mac_dst, 6 );
        copiar_datos( org_mac_addr, eTr_p->mac_org, 6 );
        reply_len += SIZE_ETHERNET_HEADER;

        return reply_len;
}

//*****************************************************************
//Funcion para analizar y procesar la trama recibida
Xuint8 analisis_trama( Xuint8 *buf) {
  Xuint32 temp         = 0x00000000;
        Xuint32 sentido      = 0x00000000;

        int length_trama,i;
        Xuint8 protocoloUDP;
        Xuint16 ttl_proto;
        struct trama_eth *eTr_p;
        eTr_p = (struct trama_eth *)buf;

        if ( eTr_p->type_length == 0x0806 ){
                return 0x01;
        }

        if ( eTr_p->type_length != 0x0800 ){
                return 0x02;
        }
```

```
	if (
	eTr_p->mac_dst[0]!=org_mac_addr[0]||
	eTr_p->mac_dst[1]!=org_mac_addr[1]||
	eTr_p->mac_dst[2]!=org_mac_addr[2]||
	eTr_p->mac_dst[3]!=org_mac_addr[3]||
	eTr_p->mac_dst[4]!=org_mac_addr[4]||
	eTr_p->mac_dst[5]!=org_mac_addr[5]){
		return 0x03;
	}

	buf += SIZE_ETHERNET_HEADER;
	struct trama_ip *iTr_p;
	iTr_p = (struct trama_ip *) buf;

	ttl_proto=iTr_p->ttl_proto;
	protocoloUDP=ttl_proto;
	if ( protocoloUDP != 0x11 ){
		return 0x04;
	}

	buf +=SIZE_IP_HEADER;

	struct trama_udp *uTr_p;
	uTr_p = (struct trama_udp *) buf;

	length_trama=uTr_p->udp_len - SIZE_UDP_HEADER;
	buf += SIZE_UDP_HEADER;

	for(i=0; i<length_trama ; i++) {
		switch(buf[i]){
//		case 0x2D:sentido=0x0001;break;
		case 0x30:temp=temp*10+0;break;
		case 0x31:temp=temp*10+1;break;
		case 0x32:temp=temp*10+2;break;
		case 0x33:temp=temp*10+3;break;
		case 0x34:temp=temp*10+4;break;
		case 0x35:temp=temp*10+5;break;
		case 0x36:temp=temp*10+6;break;
		case 0x37:temp=temp*10+7;break;
		case 0x38:temp=temp*10+8;break;
		case 0x39:temp=temp*10+9;break;
		default:temp=255;
		}
	}

	//DataReceived =((sentido<<16)|temp);
	DataReceived = temp;

	return 0x05;
}


#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

estructuras.h

```
#ifndef ESTRUCTURAS_H                /* prevent circular inclusions */
#define ESTRUCTURAS_H                /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

/* ESTOS SON LOS VALORES DE MAC E IP DEL HOST EN LINUX MINT*/

#include "xparameters.h"
//*****************************************************************************
//MAC , IP y Puerto UDP origen, esto es del FPGA
Xuint8 org_mac_addr[] = { 0x00, 0x18, 0x3e, 0x00, 0x01, 0x09 };
Xuint8 org_ip_addr[] = { 10, 0, 2, 80 };    //Ultima
Xuint16 org_puerto = 0x1136;//4406
//MAC , IP y Puerto UDP destino, esto es de la PC
// MAC de PC PEDRO DE VirtualBOX: 08-00-27-32-43-FD e IP: 10.0.2.15
// MAC de PC Linux Mint-HOST: B8:70:F4:65:E3:BF e IP: 10.0.2.15
Xuint8 dst_mac_addr[] = { 0xb8, 0x70, 0xf4, 0x65, 0xe3, 0xbf };  //Ultima
Xuint8 dst_ip_addr[] = { 10, 0, 2, 15 };    //Ultima
Xuint16 dst_puerto = 0x1135;//4405
//*****************************************************************************
//Structure for ETHERNET header
struct trama_eth{
        Xuint8 mac_dst[6];
        Xuint8 mac_org[6];
        Xuint16 type_length;
};
//Structure for IP header
struct trama_ip {
        Xuint16 v_hl_tos;
        Xuint16 len;
        Xuint16 id;
        Xuint16 offset;
        Xuint16 ttl_proto;
        Xuint16 chksum;
        Xuint8 ip_org[4];
        Xuint8 ip_dst[4];
};
//Structure for UDP header
struct trama_udp {
  Xuint16 udp_org;
  Xuint16 udp_dst;
  Xuint16 udp_len;
  Xuint16 udp_chksum;
};
//Structure for ARP response
struct trama_arp {
  Xuint16 hw_type;
  Xuint16 prot_type;
  Xuint8 hw_size;
  Xuint8 prot_size;
  Xuint16 opcode;
  Xuint8 mac_org[6];
  Xuint8 ip_org[4];
  Xuint8 mac_dst[6];
  Xuint8 ip_dst[4];
```

```
};

#define LONG_MAX_TRAMA_8BITS XEL_MAX_FRAME_SIZE
#define LONG_MAX_TRAMA_32BITS ((LONG_MAX_TRAMA_8BITS / sizeof(Xuint32)) + 1)
#define SIZE_ETHERNET_HEADER 14
#define SIZE_IP_HEADER 20
#define SIZE_UDP_HEADER 8
#define SIZE_ARP_BODY 28

#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

ANEXO C: INTERFACE DE USUARIO EN PLATAFORMA JAVA

CODIFICACIÓN JAVA – RX (INTERFACE DE USUARIO)

```
package Interface;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import java.net.*;
import java.io.*;
/**
 *
 * @author pedrol2
 */
public class TareaRx implements Runnable{

    private static byte[] buffer;
    private static byte[] data;
    /**Para cambiar el tipo de dato**/
    public static volatile int newdata;

    //public static volatile int xi=10;
    public static volatile int eje_x=560;

    public static volatile int dataLeds;
    public static volatile int dataDisplay;
    public static volatile int dataAdc;
    /*******************************/

    public void run() {
        // TODO code application logic here
        try{/*****Esta es la tarea encargada de la Rx*****/
            System.out.println("Se habilito tarea de Rx");
            /*******************************************/
            DatagramSocket socket = new DatagramSocket(4405);
            //buffer = new byte [1024];
            buffer = new byte [12]; // one more (from 3 to 6) to send data to display

            while(true){

                DatagramPacket datagrama = new DatagramPacket(buffer,buffer.length);
                socket.receive(datagrama);
                InetAddress hostDestino = datagrama.getAddress();
                //System.out.println("Data recibida :"+hostDestino);
                int puertoDestino =datagrama.getPort();
                data = datagrama.getData();

//                System.out.println("Mostrando data  "+data);
                //String cadena = new String(data,0,data.length);
                //data.lenght; el valor es 6

                String cadena = new String(data,8,data.length-8); // Data for LED's
                String cadena1 = new String(data,4,data.length-8); //Data for Display
                String cadena2 = new String(data,0,data.length-8); //Data for ADC

                //System.out.println("Data :  "+data);
                System.out.println("Tamaño de la data: "+data.length);
                System.out.println("Data para LED's:  "+cadena);
```

```
            System.out.println("Data para Display:  "+cadena1);
            System.out.println("Data para ADC:  "+cadena2);

            newdata = Integer.parseInt(cadena);

            dataLeds = Integer.parseInt(cadena);

            dataDisplay = Integer.parseInt(cadena1);

            dataAdc = Integer.parseInt(cadena2);

          }
       }
       catch(IOException e){
          e.printStackTrace(); //comentado debido a que netbeans lo sugería
          System.out.println(); // añadido después de haber comentado la línea superior
       }
    }


    public static int getdata() {
       return newdata;
    }

    public static int dataLeds() {
       return dataLeds;
    }

    public static int dataDisplays() {
       return dataDisplay;
    }

    public static int dataAdc() {
       return dataAdc;
    }

    public static int getdata_x() {
      return eje_x;
    }
}
```

CODIFICACIÓN JAVA – TX (INTERFACE DE USUARIO)

```
package Interface;

import java.net.*;
import java.io.*;
/**
 *
 * @author pedrol2
 */
public class TareaTx implements Runnable{

    private byte[] buffer;
    private volatile String cadena = "0";
    private volatile int data2send = 0;
    private String ip = "10.0.2.80";
```

```
    /**
     * @param args the command line arguments
     */
    public void run() {
        // TODO code application logic here

        /*****Esta es la tarea encargada de la Tx*****/
            System.out.println("Se habilito tarea de Tx");
        /*********************************************/
    while(true){
    if(Ventana.Tx_flag()==1){
        try{

            data2send = Ventana.datafromsw();

            cadena=String.valueOf(data2send);

            buffer = cadena.getBytes();

            InetAddress destino = InetAddress.getByName(ip);
            //InetAddress destino = InetAddress.getByAddress(ip);

            DatagramPacket datagrama = new DatagramPacket(buffer, buffer.length,destino, 4406);
            DatagramSocket socket = new DatagramSocket();

            //InetAddress hostDestino = datagrama.getAddress();
            //System.out.println("Dirección Destino "+hostDestino);
            socket.send(datagrama);
            socket.close();
        }
        catch (IOException e){
            e.printStackTrace(); //comentado por recomendacion de NETBEANS
            System.out.println("NO SE LOGRA COMUNICACION");
        }
        }
    }
  }

}
```

CODIFICACIÓN JAVA – VENTANA (INTERFACE DE USUARIO)

```
package Interface;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.Graphics.*;
/**********************/
import java.awt.Shape.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/**********************/

/**
```

```
 *
 * @author pedrol2
 */
public class Ventana extends JFrame implements Runnable, ActionListener{

    private double xp1, yp1, xp2, yp2;

    public static volatile int datafromsw;

    public static volatile int flag=0;

    /*********************************************/
      protected JSlider slider;
    /*********************************************/

//    private Graphics g;
    public Graphics g;

        //Creando los botones a usar
        JButton b1 = new JButton("SW7");
        JButton b2 = new JButton("SW6");
        JButton b3 = new JButton("SW5");
        JButton b4 = new JButton("SW4");
        JButton b5 = new JButton("SW3");
        JButton b6 = new JButton("SW2");
        JButton b7 = new JButton("SW1");
        JButton b8 = new JButton("SW0");

        //Cambiando la posición de los botones
//        this.setLayout(null);
//        b1.setBounds(50, 50, 100, 100);
        JPanel bottomPanel = new JPanel();

        JPanel holdAll = new JPanel();


    public Ventana() {
        super ("Laboratorio Virtual");

        /*********************************************/
    slider = new JSlider(JSlider.HORIZONTAL, 1, 5, 1);
    slider.setPaintLabels(true);
        /*********************************************/
        //Container c = getContentPane();
        //c.setLayout(new FlowLayout());
        /*********************************************/
 /*********************************************/
        //Cambiando el color por defecto de los botones
        b1.setBackground(Color.blue);
        b1.setLayout(null);
        b1.setBounds(5, 270, 60, 30);
        //botones.setBounds(-30, 270, 600, 60);

        b2.setBackground(Color.blue);
        b2.setLayout(null);
        b2.setBounds(70, 270, 60, 30);

        b3.setBackground(Color.blue);
        b3.setLayout(null);
```

```
        b3.setBounds(135, 270, 60, 30);

        b4.setBackground(Color.blue);
        b4.setLayout(null);
        b4.setBounds(200, 270, 60, 30);

        b5.setBackground(Color.blue);
        b5.setLayout(null);
        b5.setBounds(265, 270, 60, 30);

        b6.setBackground(Color.blue);
        b6.setLayout(null);
        b6.setBounds(330, 270, 60, 30);

        b7.setBackground(Color.blue);
        b7.setLayout(null);
        b7.setBounds(395, 270, 60, 30);

        b8.setBackground(Color.blue);
        b8.setLayout(null);
        b8.setBounds(460, 270, 60, 30);

        //Listen for actions on buttons 1 and 8.
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        b4.addActionListener(this);
        b5.addActionListener(this);
        b6.addActionListener(this);
        b7.addActionListener(this);
        b8.addActionListener(this);

        //      b1.doClick();
        b1.setActionCommand("disable01");
        b2.setActionCommand("disable02");
        b3.setActionCommand("disable03");
        b4.setActionCommand("disable04");
        b5.setActionCommand("disable05");
        b6.setActionCommand("disable06");
        b7.setActionCommand("disable07");
        b8.setActionCommand("disable08");

        bottomPanel.setLayout(null);
        //bottomPanel.setDoubleBuffered(true);
        bottomPanel.add(b1); bottomPanel.add(b2); bottomPanel.add(b3); bottomPanel.add(b4);
        bottomPanel.add(b5); bottomPanel.add(b6); bottomPanel.add(b7); bottomPanel.add(b8);

        /*************************************************/
        bottomPanel.add(slider);
        /*************************************************/
        getContentPane().add(bottomPanel);

        /*****************************************/
        /*****************************************/
        /*****************************************/
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setSize(1336,738); //1366x768 pixels
        this.setVisible(true);
    }
```

```
public void actionPerformed(ActionEvent e) {

   if("disable01".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW7");
      datafromsw=128;
      flag=1;
   }
   //else flag=0;
   if("disable02".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW6");
      datafromsw=64;
      flag=1;
   }
   //else flag=0;
   if("disable03".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW5");
      datafromsw=32;
      flag=1;
   }
   //else flag=0;
   if("disable04".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW4");
      datafromsw=16;
      flag=1;
   }
   //else flag=0;
   if("disable05".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW3");
      datafromsw=8;
      flag=1;
   }
   //else flag=0;
   if("disable06".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW2");
      datafromsw=4;
      flag=1;
   }
   //else flag=0;
   if("disable07".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW1");
      datafromsw=2;
      flag=1;
   }
   //else flag=0;
   if("disable08".equals(e.getActionCommand())){
      System.out.println("HOLA DESDE SW0");
      datafromsw=1;
      flag=1;
   }
   //else flag=0;
}

/**
 * @param args the command line arguments
 */

@Override
```

```
    public void update(Graphics g){
        paint(g);
    }

    @Override
    public void paint (Graphics g) {

        //super.paint(g);
        super.paintComponents(g);
        SubTitulos(g);
        Leds(g);
        Display7Seg(g);
        DataVsT(g);
        //Switches(g);
    }

//        Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
//        g2.setStroke(pincel);
//        g2.setColor(Color.black);

    public void PincelBlack(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /*Para colocar el grosor del pincel*/
        //Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        Stroke pincel = new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        g2.setStroke(pincel);
        g2.setColor(Color.BLACK);//g2.setColor(Color.white);
    }

    public void PincelCyan(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /*Para colocar el grosor del pincel*/
        //Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        Stroke pincel = new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        g2.setStroke(pincel);
        g2.setColor(Color.CYAN);
    }

    public void PincelGray(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /*Para colocar el grosor del pincel*/
        //Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        Stroke pincel = new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        g2.setStroke(pincel);
        g2.setColor(Color.GRAY);
    }

    public void PincelLightGray(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /*Para colocar el grosor del pincel*/
        //Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        Stroke pincel = new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        g2.setStroke(pincel);
        g2.setColor(Color.LIGHT_GRAY);
```

```
    }

    public void PincelBlue(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /*Para colocar el grosor del pincel*/
        //Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        g2.setStroke(pincel);
        g2.setColor(Color.BLUE);//g2.setColor(Color.white);
    }

    public void PincelWhite(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /*Para colocar el grosor del pincel*/
        //Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        Stroke pincel_1 = new BasicStroke(1.0f, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_MITER);
        g2.setStroke(pincel_1);
        g2.setColor(Color.WHITE);//g2.setColor(Color.white);
    }

    public void DibujarFondo(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        DibujarEjes(g);
        PincelGray(g);

        Rectangle2D grafica_ad = new Rectangle2D.Float(550, 130, 700, 200); // PARA GRAFICA
        g2.draw(grafica_ad);
        for(int i=550; i<1260; i=i+10  ){
            for(int k=130; k<330; k=k+10  ){
            Coordenadas(i, k, i, k);
            g.drawLine((int)xp1, (int)yp1, (int)xp2, (int)yp2);
            }
        }
    }
    public void Coordenadas(double x1, double y1, double x2, double y2){
        xp1=x1;
        yp1=y1;
        xp2=x2;
        yp2=y2;
    }

    public void DibujarEjes(Graphics g){
        PincelBlack(g);
        g.drawLine(560, 320, 1240, 320);
        g.drawLine(560, 140, 560, 320);
    }

    public void Display7Seg(Graphics g){

        Graphics2D g2 = (Graphics2D)g;
        PincelWhite(g);
        //Estos nos representarán los SEGMENTOS
        ///////DISPLAY DEL LADO DERECHO
        Rectangle2D r2d_dpd = new Rectangle2D.Float(395, 570, 10, 10); // DP DEL DISPLAY
DERECHO
        Rectangle2D r2d_fd = new Rectangle2D.Float(320, 470, 10, 45); // F DEL DISPLAY
```

```
DERECHO
      Rectangle2D r2d_gd = new Rectangle2D.Float(330, 515, 45, 10); // G DEL DISPLAY
DERECHO
      Rectangle2D r2d_ed = new Rectangle2D.Float(320, 525, 10, 45); // E DEL DISPLAY
DERECHO
      Rectangle2D r2d_dd = new Rectangle2D.Float(330, 570, 45, 10); // D DEL DISPLAY
DERECHO
      Rectangle2D r2d_cd = new Rectangle2D.Float(375, 525, 10, 45); // C DEL DISPLAY
DERECHO
      Rectangle2D r2d_bd = new Rectangle2D.Float(375, 470, 10, 45); // B DEL DISPLAY
DERECHO
      Rectangle2D r2d_ad = new Rectangle2D.Float(330, 460, 45, 10); // A DEL DISPLAY
DERECHO

      ///////DISPLAY DEL LADO IZQUIERDO
      Rectangle2D r2d_dpi = new Rectangle2D.Float(195, 570, 10, 10); // DP DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_fi = new Rectangle2D.Float(120, 470, 10, 45); // F DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_gi = new Rectangle2D.Float(130, 515, 45, 10); // G DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_ei = new Rectangle2D.Float(120, 525, 10, 45); // E DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_di = new Rectangle2D.Float(130, 570, 45, 10); // D DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_ci = new Rectangle2D.Float(175, 525, 10, 45); // C DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_bi = new Rectangle2D.Float(175, 470, 10, 45); // B DEL DISPLAY
IZQUIERDO
      Rectangle2D r2d_ai = new Rectangle2D.Float(130, 460, 45, 10); // A DEL DISPLAY
IZQUIERDO

      ///////DISPLAY DEL MEDIO
      Rectangle2D r2d_dpm = new Rectangle2D.Float(295, 570, 10, 10); // DP DEL DISPLAY
MEDIO
      Rectangle2D r2d_fm = new Rectangle2D.Float(220, 470, 10, 45); // F DEL DISPLAY MEDIO
      Rectangle2D r2d_gm = new Rectangle2D.Float(230, 515, 45, 10); // G DEL DISPLAY MEDIO
      Rectangle2D r2d_em = new Rectangle2D.Float(220, 525, 10, 45); // E DEL DISPLAY MEDIO
      Rectangle2D r2d_dm = new Rectangle2D.Float(230, 570, 45, 10); // D DEL DISPLAY MEDIO
      Rectangle2D r2d_cm = new Rectangle2D.Float(275, 525, 10, 45); // C DEL DISPLAY MEDIO
      Rectangle2D r2d_bm = new Rectangle2D.Float(275, 470, 10, 45); // B DEL DISPLAY MEDIO
      Rectangle2D r2d_am = new Rectangle2D.Float(230, 460, 45, 10); // A DEL DISPLAY MEDIO
      /*********************************************************************/
      g2.draw(r2d_ad); g2.draw(r2d_ai); g2.draw(r2d_am);
      g2.draw(r2d_bd); g2.draw(r2d_bi); g2.draw(r2d_bm);
      g2.draw(r2d_cd); g2.draw(r2d_ci); g2.draw(r2d_cm);
      g2.draw(r2d_dd); g2.draw(r2d_di); g2.draw(r2d_dm);
      g2.draw(r2d_ed); g2.draw(r2d_ei); g2.draw(r2d_em);
      g2.draw(r2d_fd); g2.draw(r2d_fi); g2.draw(r2d_fm);
      g2.draw(r2d_gd); g2.draw(r2d_gi); g2.draw(r2d_gm);
      g2.draw(r2d_dpd); g2.draw(r2d_dpi); g2.draw(r2d_dpm);

      /*********************************************************************/
      /**Añadiendo lógica al encendido y apagado de los DISPLAYS**/
      //IMPORTANTE: Tenemos que continuar con esto
      //int data=0;
      /*********************************************************************/
      //int data=0;
      //data=TareaRx.dataDisplays();
```

```
        int sel_ad=0, sel_am=0, sel_ai=0;

        int data=0x00;
        int bit0=0, bit1=0, bit2=0, bit3=0, bit4=0, bit5=0, bit6=0, bit7=0;

                //data = TareaRx.dataLeds();

                data = TareaRx.dataDisplays();

                bit0 = data & 0x01;
                bit1 = data & 0x02;
                bit2 = data & 0x04;
                bit3 = data & 0x08;
                bit4 = data & 0x10;
                bit5 = data & 0x20;
                bit6 = data & 0x40;
                bit7 = data & 0x80;

                sel_ad=1; sel_am=1; sel_ai=1;

    if(sel_ad==1){

                switch (bit0) {
                    //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                    default:
                        g2.setColor(Color.red);
                        g2.fill(r2d_ad);break;
                    case 0:
                        g2.setColor(Color.white);
                        g2.fill(r2d_ad);break;
                }
                switch (bit1) {
                    //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                    default:
                        g2.setColor(Color.red);
                        g2.fill(r2d_bd);break;
                    case 0:
                        g2.setColor(Color.white);
                        g2.fill(r2d_bd);break;
                }
                switch (bit2) {
                    //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                    default:
                        g2.setColor(Color.red);
                        g2.fill(r2d_cd);break;
                    case 0:
                        g2.setColor(Color.white);
                        g2.fill(r2d_cd);break;
                }
                switch (bit3) {
                    //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                    default:
                        g2.setColor(Color.red);
                        g2.fill(r2d_dd);break;
                    case 0:
```

```
                    g2.setColor(Color.white);
                    g2.fill(r2d_dd);break;
            }
            switch (bit4) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_ed);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_ed);break;
            }
            switch (bit5) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_fd);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_fd);break;
            }
            switch (bit6) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_gd);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_gd);break;
            }
            switch (bit7) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_dpd);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_dpd);break;
            }
            }


   if(sel_am==1){

            switch (bit0) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_am);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_am);break;
            }
            switch (bit1) {
```

```
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_bm);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_bm);break;
            }
            switch (bit2) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_cm);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_cm);break;
            }
            switch (bit3) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_dm);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_dm);break;
            }
            switch (bit4) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_em);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_em);break;
            }
            switch (bit5) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_fm);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_fm);break;
            }
            switch (bit6) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_gm);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_gm);break;
            }
```

```
            switch (bit7) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_dpm);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_dpm);break;
            }
            }


   if(sel_ai==1){

            switch (bit0) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_ai);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_ai);break;
            }
            switch (bit1) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_bi);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_bi);break;
            }
            switch (bit2) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_ci);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_ci);break;
            }
            switch (bit3) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
                  g2.setColor(Color.red);
                  g2.fill(r2d_di);break;
               case 0:
                  g2.setColor(Color.white);
                  g2.fill(r2d_di);break;
            }
            switch (bit4) {
               //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
               default:
```

```
                    g2.setColor(Color.red);
                    g2.fill(r2d_ei);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_ei);break;
            }
            switch (bit5) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_fi);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_fi);break;
            }
            switch (bit6) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_gi);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_gi);break;
            }
            switch (bit7) {
                //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                default:
                    g2.setColor(Color.red);
                    g2.fill(r2d_dpi);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(r2d_dpi);break;
            }
            }
    }

    public void Leds(Graphics g){

        Graphics2D g2 = (Graphics2D)g;
        PincelWhite(g);
        //PincelBlack(g);
        /*****************Dibujando Elipses***************/
        //Estas nos representarán los LED's
        Ellipse2D e2d_1 = new Ellipse2D.Float(50,150,30, 30); // LED07
        Ellipse2D e2d_2 = new Ellipse2D.Float(100,150,30,30); // LED06
        Ellipse2D e2d_3 = new Ellipse2D.Float(150,150,30,30); // LED05
        Ellipse2D e2d_4 = new Ellipse2D.Float(200,150,30,30); // LED04
        Ellipse2D e2d_5 = new Ellipse2D.Float(250,150,30,30); // LED03
        Ellipse2D e2d_6 = new Ellipse2D.Float(300,150,30,30); // LED02
        Ellipse2D e2d_7 = new Ellipse2D.Float(350,150,30,30); // LED01
        Ellipse2D e2d_8 = new Ellipse2D.Float(400,150,30,30); // LED00

        // Estas son las Elipses(en realidad círculos)
        g2.draw(e2d_1);
        g2.draw(e2d_2);
        g2.draw(e2d_3);
```

```
        g2.draw(e2d_4);
        g2.draw(e2d_5);
        g2.draw(e2d_6);
        g2.draw(e2d_7);
        g2.draw(e2d_8);

        /*****************************************************************/
        /**Añadiendo lógica al encendido y apagado de los LED's**/
        //IMPORTANTE: Tenemos que continuar con esto
        //int newdata=0;
        /*****************************************************************/
        int switches=0x00;
        int bit0=0, bit1=0, bit2=0, bit3=0, bit4=0, bit5=0, bit6=0, bit7=0;

                switches = TareaRx.dataLeds();

                bit0 = switches & 0x01;
                bit1 = switches & 0x02;
                bit2 = switches & 0x04;
                bit3 = switches & 0x08;
                bit4 = switches & 0x10;
                bit5 = switches & 0x20;
                bit6 = switches & 0x40;
                bit7 = switches & 0x80;

            //System.out.println("Bytebit: "+bit7+bit6+bit5+bit4+bit3+bit2+bit1+bit0);

      if(true){

                switch (bit0) {
                   //case 1: //EN TODOS LOS SWITHCES TENEMOS QUE CAMBIAR CASE 1 POR
DEFAULT:
                   default:
                      g2.setColor(Color.red);
                      g2.fill(e2d_8);break;
                   case 0:
                      g2.setColor(Color.white);
                      g2.fill(e2d_8);break;
                }

                switch (bit1) {
                   default:
                      g2.setColor(Color.red);
                      g2.fill(e2d_7);break;
                   case 0:
                      g2.setColor(Color.white);
                      g2.fill(e2d_7);break;
                }

                switch (bit2) {
                   default:
                      g2.setColor(Color.red);
                      g2.fill(e2d_6);break;
                   case 0:
                      g2.setColor(Color.white);
                      g2.fill(e2d_6);break;
                }

                switch (bit3) {
```

```
                default:
                    g2.setColor(Color.red);
                    g2.fill(e2d_5);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(e2d_5);break;
            }

            switch (bit4) {
                default:
                    g2.setColor(Color.red);
                    g2.fill(e2d_4);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(e2d_4);break;
            }

            switch (bit5) {
                default:
                    g2.setColor(Color.red);
                    g2.fill(e2d_3);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(e2d_3);break;
            }

            switch (bit6) {
                default:
                    g2.setColor(Color.red);
                    g2.fill(e2d_2);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(e2d_2);break;
            }

            switch (bit7) {
                default:
                    g2.setColor(Color.red);
                    g2.fill(e2d_1);break;
                case 0:
                    g2.setColor(Color.white);
                    g2.fill(e2d_1);break;
            }
        }
    }


    public void Switches(Graphics g){

        Graphics2D g2 = (Graphics2D)g;
        PincelBlue(g);
        /*Dibujando Rectángulos*/

        //Estos nos representarán los SWITCHES
        Rectangle2D r2d_1 = new Rectangle2D.Float(50, 300, 30, 30); // SW07
        Rectangle2D r2d_2 = new Rectangle2D.Float(100, 300, 30, 30); // SW06
        Rectangle2D r2d_3 = new Rectangle2D.Float(150, 300, 30, 30); // SW05
        Rectangle2D r2d_4 = new Rectangle2D.Float(200, 300, 30, 30); // SW04
        Rectangle2D r2d_5 = new Rectangle2D.Float(250, 300, 30, 30); // SW03
```

```
        Rectangle2D r2d_6 = new Rectangle2D.Float(300, 300, 30, 30); // SW02
        Rectangle2D r2d_7 = new Rectangle2D.Float(350, 300, 30, 30); // SW01
        Rectangle2D r2d_8 = new Rectangle2D.Float(400, 300, 30, 30); // SW00

        //Estos son los Rectángulos(en realidad cuadrados)
        g2.draw(r2d_1);
        g2.draw(r2d_2);
        g2.draw(r2d_3);
        g2.draw(r2d_4);
        g2.draw(r2d_5);
        g2.draw(r2d_6);
        g2.draw(r2d_7);
        g2.draw(r2d_8);
    }

    public void DataVsT(Graphics g){
        DibujarFondo(g);
        PuntoMovil(g);
    }


    double j=0; int xi=560;
    public void PuntoMovil(Graphics g){

        Graphics2D g2 = (Graphics2D)g;
        PincelBlack(g);
        /*Para colocar el grosor del pincel*/
        g2.setStroke(new BasicStroke(3.0f));

if(xi==1240){
   xi=560;

        }
 else{
        xi+=1;

        try {
        Thread.sleep(10);
        }catch(InterruptedException ex){
            System.out.println("error de Thread.sleep");
        }

        //j= -TareaRx.dataLeds()+250;
        //j= -TareaRx.dataLeds()*20+250;
        j= -TareaRx.dataAdc()/2.5+250;
        //j= -TareaRx.dataAdc()+250;  //j = -TareaRx.getdata()+250;
                                    //El signo negativo es importante //Debido a que cuando es positivo es
                                    //hacia la parte inferior
        //xi = TareaRx.getdata_x();

        g.drawLine((int)xi, (int)j, (int)xi, (int)j);

        }
    }


    public void SubTitulos(Graphics g){

        Graphics2D g2 = (Graphics2D)g;
```

```
            /*Colocando Subtítulo para los LED's*/
        g2.setColor(Color.blue);
            //g2.setFont(new Font("Arial", Font.ITALIC, 24));[B@ce16a
        g2.setFont(new Font("Arial", Font.PLAIN, 20));
        g2.drawString(" ********** PUERTO A: LED's ********* ",50,100);
            /*Colocando Subtítulo para los SWITCHES*/
        g2.drawString(" ******** PUERTO B: SWITCHES ******* ",50,250);
            /*Colocando Subtítulo para los DISPLAY's*/
        g2.drawString(" ****** DISPLAY DE 7 SEGMENTOS ****** ",50,400);
            /*Colocando Subtítulo para la GRAFICA de la DATA*/
        g2.drawString(" ********** DATA VS TIEMPO **********",700,100);

    }

    @Override
    public void run() {
      try{
       repaint();
      }catch(Exception ex){
          System.out.println("error");
      }

    }

    public static int datafromsw() {
        return datafromsw;
    }

    public static int Tx_flag() {
        return flag;
    }

}
```

CODIFICACIÓN JAVA – VIRTUALAB (INTERFACE DE USUARIO)

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Interface;

/**********************************/
import javax.swing.*;
import java.awt.Graphics.*;
/**********************************/
import java.awt.Shape.*;
/**********************************/

/**
 *
 * @author pedrol2
 */
public class VirtuaLab{

    protected JSlider slider;
```

```
    public VirtuaLab() {


    }

    public static void main(String[] args) {
        // TODO code application logic here

        /************Para poder llamar las tareas creadas**********/
        TareaRx RxUDP = new TareaRx();    //Tarea para Rx
        Thread Rx_task = new Thread(RxUDP);    //Lanzando tarea Rx

        TareaTx TxUDP = new TareaTx();    //Tarea para Tx
        Thread Tx_task = new Thread(TxUDP);    //Lanzando tarea Tx
        /************************************************************/


        /************************************************************/
        /****Añadiendo una entrada para ser visualiza en la ventana***/
        String entrada = JOptionPane.showInputDialog("What's your device?");

        int valor = Integer.parseInt(entrada);
        int valorcorrecto = 1; //int valorcorrecto = 16877;
        int start = 0; //start is used to begin the code below

        start = 1;

        if(start==1){

              if(valor==valorcorrecto){

                 JOptionPane.showMessageDialog(null, "Dispositivo aceptado");

                 /*****Entonces se realizará el pintado de la Gráfica**********/

                 Ventana v= new Ventana();
                 Thread Lab = new Thread(v);
                 Lab.start();

                 /************************************************************/
                 /********************************************************************/
                 System.out.println("Esta será la Interface Gráfica");
                 /************************************************************/
                 /*****Entonces se dará inicio a la comunicación UDP-TareaRx****/
                 Rx_task.start();
                 /*****Entonces se dará inicio a la comunicación UDP-TareaTx****/
                 Tx_task.start();

                 /******************/

                 /*****Entonces se dará inicio al testeo de estos led para poder realizar ******/
                 /*******el pintado de los leds en la interfaz de usuario*****************/
                 ///*
                 while(true){

                    v.repaint();
//                   try{
//                       Thread.sleep(100);
//                   }catch(Exception e){
```

```
//
//                  }

//                  try{
//                     v.repaint();
//                     //v.repaint(1, 0, 1300, 1300, 300);
//                     //v.repaint(0, 0, 1300, 200);
//                     Thread.sleep(100);
//                  }catch(Exception e){

//                  }
               }
               //*/

            }
            else{
               JOptionPane.showMessageDialog(null, "Valido solo para familia 16f");
               start = 0;
            }  //JOptionPane.showInputDialog(null,entrada,0);
      }
   }

}
```

BIBLIOGRAFÍA

[1] Rapid Prototyping Of Digital Systems Sopc Edition
-James O. Hamblen
-Tyson S. Hall
-Michael D. Furman

[2] Soft CPU Cores para FPGA's
[Online] Available:
http://www.1-core.com/library/digital/soft-cpu-cores/

[3] Ethernet Communication Interface for the FPGA - Cornell University
[Online] Available:
http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2011/mis47_ayg6/mis47_ayg6/

[4] FPGA implementation of Real-time Ethernet communication using RMII Interface.
[Online] Available:
http://www.idt.mdh.se/kurser/ct3340/ht10/FinalPapers/17-Khalilzad_Pourshakour.pdf

[5] Remote FPGA Lab With Interactive Control and Visualisation Interface.
[Online] Available:
http://remotefpga.com/m/u/2fpl11_RemoteFPGA.pdf

[6] Monitoreo y Control de Temperatura de un Estanque de Agua entre Chile y España usando Redes de Alta Velocidad.
[Online] Available:
http://www.redalyc.org/articulo.oa?id=11411205

[7] SOPC-Based Speech-to-Text Conversion.
[Online] Available:
http://www.altera.com/literature/dc/2006/i2.pdf

[8] FPGAs as versatile configurable I/O devices in Hardware-in-the-Loop Simulation*)
[Online] Available:
http://redesign.esi.nl/publications/boderc/pv_FPGAsVersatile_Lisbon.pdf

[9] Definición de FPGA dada por Xilinx
[Online] Available:
http://www.xilinx.com/products/silicon-devices/fpga/index.htm

[10] Sistema de Visualización de Imágenes a 8 Colores empleando la tarjeta de desarrollo Digilent Spartan 3
[Online] Available:
http://jupiter.utm.mx/~tesis_dig/11608.pdf

[11] Cyclone II FPGA Starter Development Kit – Guía de usuario
[Online] Available:
http://www.altera.com/literature/ug/ug_cii_starter_board.pdf

[12] Circuit Design with VHDL
Volnei A. Pedroni

[13] Experiences with Soft-Core Processor Design
[Online] Available:
http://www.eecg.toronto.edu/~brown/papers/raw05-plavec.pdf

[14] Soft CPU Cores para FPGA's
[Online] Available:
http://www.1-core.com/library/digital/soft-cpu-cores/

[15] Microblaze de Xilinx
[Online] Available:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/mb_ref_guide.pdf

[16] ProcesadorNios II-
[Online].Available:
http://www.altera.com/devices/processor/nios2/ni2-index.html

[17] Opencores.
[Online].Available:
http://www.opencores.org

[18] Computer Networking: Principles, Protocols and Practice.
[Online].Available:
http://www.saylor.org/site/wp-content/uploads/2012/02/Computer-Networking-Principles-Bonaventure-1-30-31-OTC1.pdf

[19] A Brief History of the Internet.
[Online].Available:
http://www.cs.ucsb.edu/~almeroth/classes/F10.176A/papers/internet-history-09.pdf

[20] Aniversario de Internet y la World Wide Web
[Online].Available:
https://www.itu.int/net/itunews/issues/2009/10/pdf/200910_34-es.pdf

[21] Research Projects during 1970's.
[Online].Available:
http://rogerdmoore.ca/PS/

[22] Networking Protocol Layers.
[Online] Available:
http://www.comptechdoc.org/independent/networking/protocol/protlayers.html

[23] The TCP/IP Protocol Suite
[Online] Available:
http://www.exa.unicen.edu.ar/catedras/comdat1/material/TP1-Ejercicio5-ingles.pdf

[24] Spartan-3E Starter Kit – Guía de usuario
[Online] Available:
http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf

[25] Especificación Oficial del Protocolo IP (RFC: 791)
[Online] Available:
http://tools.ietf.org/html/rfc791

[26] Protocolo UDP
[Online] Available:
http://www.pcvr.nl/tcpip/udp_user.htm#11_0

[27] Especificación Oficial del Protocolo UDP (RFC 768)
[Online] Available:
http://tools.ietf.org/rfc/rfc768.txt

[28] Link Layer
[Online] Available:
http://www.pcvr.nl/tcpip/link_lay.htm#2_1

[29] Curso de Programación
[Curso de Java]
Javier Cevallos

[30] Paquete Java
[Online] Available:
http://es.wikipedia.org/wiki/Paquete_Java

[31] AWT
[Online] Available:
http://docs.oracle.com/javase/8/docs/technotes/guides/awt/index.html

[32] Swing
[Online] Available:
http://docs.oracle.com/javase/8/docs/technotes/guides/swing/index.html

[33] Painting in AWT and Swing
[Online] Available:
http://www.oracle.com/technetwork/java/painting-140037.html

[34] Socket Datagrama
[Online] Available:
http://docs.oracle.com/javase/1.5.0/docs/api/java/net/class-use/DatagramSocket.html

[35] Switch SOHO 8 PORT
[Online] Available:
http://www.dlinkla.com/des-1008d

[36] DigilentPmodH B3™ 2A H – Bridge - Manual de referencia
[Online] Available:
http://digilentinc.com/Data/Products/PMOD-HB3/PmodHB3_rm_RevD.pdf

[37] MITx 6.002x: Circuits & Electronics
[Online] Available:
https://6002x.mitx.mit.edu/

[38] MicroBlaze Configuration Wizard
[Online] Available:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/platform_studio/ps_c_mcw_microblaze_configuration_wizard.htm

[39] LTC6912 Dual Programmable Gain Amplifiers with Serial Digital Interface
[Online] Available:
http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1154,C1009,C1121,P7596,D5359

[40] LTC1407A-1 Serial 14-bit Simultaneous Sampling ADCs with Shutdown
[Online] Available:
http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1155,C1001,C1158,P2420,D1295

[41] LTC2624 Quad DAC Data Sheet
[Online] Available:
http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1155,C1005,C1156,P2048,D2170