OPTIMIZACIÓN DEL HARDWARE EMBEBIDO EN LA FPGA DE LA TARJETA EOS Y LA VALIDACIÓN PARA SU USO EN DISPOSITIVOS HÁPTICOS

JAVIER MAURICIO MELO URIBE

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERÍAS Y ADMINISTRACIÓN
ESCUELA DE INGENIERÍA ELECTRÓNICA
BUCARAMANGA
2008

OPTIMIZACIÓN DEL HARDWARE EMBEBIDO EN LA FPGA DE LA TARJETA EOS Y LA VALIDACIÓN PARA SU USO EN DISPOSITIVOS HÁPTICOS

JAVIER MAURICIO MELO URIBE

Trabajo de grado presentado para optar por el título de
Ingeniero Electrónico

Directores:

PhD. Emilio José Sánchez Tapia
MsC. Claudia Leonor Rueda Guzmán

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERÍAS Y ADMINISTRACIÓN
ESCUELA DE INGENIERÍA ELECTRÓNICA
BUCARAMANGA
2008

NOTA DE ACEPTACIÓN

_____

_____

_____

PRESIDENTE DEL JURADO

_____
JURADO

_____
JURADO

BUCARAMANGA, ___ DE _____DEL 2008

AGRADECIMIENTOS

Primero y antes que nada, quiero dar gracias a Dios, por estar a mi lado en cada paso que doy, por las personas que ha puesto en mi camino y que han sido mi fortaleza para lograr las metas que me he propuesto.

Agradecer hoy más que nunca a mi familia porque a pesar de no estar presentes físicamente a mi lado, se que desde mi país, Colombia, aúnan esfuerzos en pos de mi bienestar. A mis padres Mario e Inés y a mi hermano Alexander, por el apoyo incondicional y la constante voz de aliento que me impulsa y mantiene por el camino de la vida.

Agradezco además, a mi Director de Proyecto Emilio Sánchez Tapia, por su colaboración y por la paciencia que ha sabido tener, haciendo que hoy este proyecto se haya consolidado como una realidad.

Quiero extender mis agradecimientos a todos aquellos que de una u otra forma han estado a mi lado no sólo siendo partícipes de este trabajo sino también brindándome su amistad: a Mildred, Iñaki, Imanol y Jorge (mis compañeros de trabajo), a Gerardo, Ioseba y Dani.

Es mi deseo además agradecer muy especialmente a Josune, quien siempre ha dado un toque de alegría y dulzura a las interminables horas de trabajo.

ÍNDICE

1 INTRODUCCIÓN 1
1.1 ACERCAMIENTO A LOS DISPOSITIVOS HÁPTICOS 2
1.2 DESARROLLO TECNOLÓGICO DE LOS SISTEMAS HÁPTICOS 2
1.3 CONTROL DE DISPOSITIVOS HÁPTICOS 4
1.4 LIMITACIONES DE LOS SISTEMAS HÁPTICOS 4
1.5 APLICACIONES DE LOS HÁPTICOS 5
1.5.1 Aplicaciones Médicas 6
1.5.2 Aplicaciones en el Diseñó Mecánico 6
1.5.3 Aplicaciones en la Industria del Entretenimiento 7
1.5.4 Aplicaciones en las Artes Gráficas 8
2 PLANTEAMIENTO DEL PROBLEMA 9
2.1 OBJETIVO GENERAL 9
2.2 OBJETIVOS ESPECIFICOS 9
2.3 DEFINICIÓN DEL PROBLEMA 10
2.4 JUSTIFICACIÓN 10
2.5 SOLUCIONES ACTUALES A LA PROBLEMÁTICA 12
3 ESTADO DEL ARTE 14
3.1 SEGMENTACIÓN DE INSTRUCCIONES Y PROCESOS 14
3.2 CIRCUITOS ELECTRÓNICOS DEDICADOS 15
3.3 SISTEMAS DE COPROCESAMIENTO DE DATOS 16
3.3.1 Módulo para Controlador PID 16
3.3.2 Controlador para Interfaz Táctil 17
4 DISPOSITIVOS LÓGICOS PROGRAMABLES 19
4.1 TIPOS DE DISPOSITIVOS LÓGICOS PROGRAMABLES 19
4.1.1 Definiciones y Terminología Empleada 20
4.1.2 Evolución de los Dispositivos Lógicos Programables 21
4.2 FPGAs DISPONIBLES COMERCIALMENTE 24
4.3 XILINX SPARTAN 3 25

4.3.1 Bloque Lógico Configurable (CLB) 26
4.3.2 Bloques Entrada/Salida (IOB) 26
4.3.3 Bloque RAM 26
4.3.4 Bloques de Multiplicación 27
4.3.5 Manejador Digital de Reloj (DCM) 27
4.4 MICROPROCESADORES SOFT – CORE 27
4.5 MICROBLAZE 28
5 DISEÑO DEL MODELO DE HARDWARE 31
5.1 DISEÑO DEL MODELO 32
5.2 ENTORNO ISE DE XILINX 32
5.3 EDICIÓN DE CÓDIGO VHDL MEDIANTE EL ISE 33
5.3.1 INSTANCIAMIENTO Y MAPEADO DE UN COMPONENTE 35
5.4 EDICIÓN MEDIANTE CAPTURA DE BLOQUES ESQUEMÁTICOS (ECS) 38
5.5 REPRESENTACIÓN GRÁFICA DE DIAGRAMAS DE ESTADO 40
5.6 HERRAMIENTA DE DISEÑO CORE GENERATOR 42
5.7 PROGRAMACIÓN MEDIANTE SIMULINK HDL CODER 44
5.8 SÍNTESIS DEL MODELO 47
5.8.1 Identificación de Errores 48
5.8.2 Síntesis 48
5.8.3 Optimización a Bajo Nivel 49
5.9 IMPLEMENTACIÓN DEL MODELO 49
5.9.1 Traducción 51
5.9.2 Mapeado 52
5.9.3 Implementación 52
6 DISEÑO DE UN SISTEMA EMBEBIDO EN FPGA 53
6.1 INTRODUCCIÓN AL ENTORNO EDK DE XILINX 53
6.2 CONSTRUCCIÓN DEL PROYECTO DE HARDWARE 54
6.2.1 Interfaz del XPS 62
6.2.2 Archivos de Configuración del Sistema 65
6.3 CREACIÓN Y ADICIÓN DE NUEVOS PERIFÉRICOS 68
6.3.1 Creación de un Nuevo Periférico 68
6.3.2 Adición de un Nuevo Periférico 72
6.3.3 Configuración del Bus FSL 79

6.4 COMPILACIÓN DEL PROYECTO DE HARDWARE Y GENERACIÓN DEL BITSTREAM 82
6.5 CONSTRUCCIÓN DEL PROYECTO DE SOFTWARE 84
6.5.1 Configuración de los Parámetros Software del MicroBlaze 84
6.5.2 Creación de una Aplicación Software 85
7 CONSTRUCCIÓN DE HARDWARE EMBEBIDO PARA LA TARJETA EOS 89
7.1 DESCRIPCIÓN DE LA TARJETA EOS 89
7.2 DESARROLLO DE HARDWARE EMBEBIDO PARA LA TARJETA EOS 91
7.2.1 Diseño del Modelo de una IP Core para la Tarjeta EOS 91
7.2.3 Creación del Nuevo Periférico 95
7.2.4 Descripción del Código VHDL de la IP Core Implementada 96
7.2.5 Adición de la Nueva IP Core a la Tarjeta EOS 101
7.3 DESARROLLO DE CONTROLADORES PARA LA TARJETA EOS 104
8 MODELOS DE CONTACTO 108
8.1 MODELO ELÁSTICO 109
8.1.1 Modelo Elástico Lineal 110
8.1.2 Modelo Elástico No Lineal 110
8.2 MODELO VISCOELÁSTICO 111
8.3 MODELO VISCOELÁSTICO NO LINEAL 112
8.4 MODELO SEMIVISCOELÁSTICO 113
8.5 OPEN LOOP FORCE PULSES 114
8.6 BRAKING PULSE 115
9 VALIDACIÓN DE LA TARJETA EOS 118
9.1 ANÁLISIS DEL TIEMPO DE EJECUCIÓN PARA EL CÁLCULO DE FUNCIONES TRIGONOMÉTRICAS 118
9.1.1 Cálculo de Funciones Trigonométricas Utilizando la IPSC 118
9.1.2 Cálculo de Funciones Trigonométricas Directamente en Software 119
9.1.3 Cálculo de Funciones Trigonométricas Utilizando la Unidad de Coma Flotante del MicroBlaze 120
9.2 ANÁLISIS DEL TIEMPO DE EJECUCIÓN PARA EL CONTROL DEL PHANTOM 121
9.2.1 Conversión a Coordenadas Articulares 123
9.2.2 Cinemática Directa 125

9.2.3 Jacobiana Transpuesta 131
9.2.4 Análisis del Tiempo de Ejecución del Lazo de Control 133
9.3 PRUEBAS UTILIZANDO MODELOS DE CONTACTO 134
9.3.1 Modelo Elástico 135
9.3.2 Modelo Viscoelástico 137
10 CONCLUSIONES 139
11 REFERENCIAS 140
ANEXO 1 142
ANEXO 2 146
ANEXO 3 149
ANEXO 4 151
ANEXO 5 157

ÍNDICE DE FIGURAS

Figura 1.1 *Interacción con el Sistema Háptico.* 2
Figura 1.2 *Aplicación de Telemedicina.* 6
Figura 1.3 *LHIfAM Diseñado por el CEIT.* 7
Figura 1.4 *Manipulador para Videojuegos.* 7
Figura 1.5 *Diseño Gráfico.* 8
Figura 2.1 *DSP dSPACE 1104.* 12
Figura 3.1 *Segmentación en Bloques Funcionales.* 15
Figura 3.2 *Algoritmo de un PID implementado en Hardware Reconfigurable.* 17
Figura 3.3 *Interfaz Háptica Controlada por FPGA.* 18
Figura 4.1 *FPDs por Capacidad Lógica.* 23
Figura 4.2 *Bloques Lógicos de una FPGA.* 24
Figura 4.3 *Fabricantes de FPGAs.* 25
Figura 4.4 *Estructura de Bloques Lógicos de la Spartan 3.* 25
Figura 4.5 *Estructura de un Bloque Lógico.* 26
Figura 4.6 *Arquitectura del MicroBlaze.* 29
Figura 4.7 *Bus de Comunicaciones FSL.* 30
Figura 5.1 *Flujo de Diseño para FPGA Xilinx.* 31
Figura 5.2 *Asistente para Creación de un Nuevo Proyecto.* 33
Figura 5.3 *Estructura de un Componente en VHDL.* 34
Figura 5.4 *Instanciamiento y Mapeo de un Componente.* 36
Figura 5.5 *Captura de Esquemáticos Mediante el ISE.* 39
Figura 5.6 *Bloque Funcional de Mayor Jerarquía.* 39
Figura 5.7 *Bloque Simbólico Generado por VHDL.* 40
Figura 5.8 *Plataforma de Diseño StateCAD.* 41
Figura 5.9 *Código VHDL Generado por StateCAD.* 42
Figura 5.10 *Asistente de Diseño del Core Generator.* 43
Figura 5.11 *Asistente para Configuración de una IP Core.* 44
Figura 5.12 *Interfaz Gráfica del HDL Coder.* 45
Figura 5.13 *Sistema Diseñado en Simulink.* 46

Figura 5.14 *Código VHDL Generado por HDL Coder.* 47
Figura 5.15 *Fases del proceso de Síntesis.* 48
Figura 5.16 *Interfaz Gráfica del PACE.* 50
Figura 5.17 *Constraints Generadas a través del PACE.* 51
Figura 5.18 *Fases del Proceso de Implementación.* 51
Figura 6.1 *Ventana de Inicio XPS.* 54
Figura 6.2 *Creación del Directorio del Proyecto.* 55
Figura 6.3 *Creación de un Nuevo Diseño.* 56
Figura 6.4 *Selección del Tipo de Tarjeta Empleada.* 57
Figura 6.5 *Especificación del Tipo de FPGA Utilizado.* 58
Figura 6.6 *Identificación del Empaquetamiento de la FPGA.* 59
Figura 6.7 *Definición de Parámetros del Microblaze.* 59
Figura 6.8 *Selección de IP Cores.* 60
Figura 6.9 *Configuración de la Memoria y la Depuración.* 61
Figura 6.10 *Reporte Final del Diseño Creado.* 62
Figura 6.11 *Interfaz Gráfica del XPS.* 63
Figura 6.12 *Panel de Información.* 63
Figura 6.13 *Estructura del Sistema.* 64
Figura 6.14 *Ventana de Reportes.* 64
Figura 6.15 *Archivos de Configuración del Sistema.* 65
Figura 6.16 *Archivo de Configuración de Hardware.* 66
Figura 6.17 *Archivo de Descripción de Software.* 67
Figura 6.18 *Archivo de Constraints.* 67
Figura 6.19 *Asistente para la Creación de Nuevos Periféricos.* 68
Figura 6.20 *Creación del Modelo para un Nuevo Periférico.* 69
Figura 6.21 *Identificación del Nombre y la Versión del Hardware.* 70
Figura 6.22 *Elección del Bus FSL.* 71
Figura 6.23 *Configuración Maestro – Esclavo del Bus FSL.* 71
Figura 6.24 *Creación de Ejemplos y Finalización del Proceso.* 72
Figura 6.25 *Adición de un Nuevo Periférico.* 73
Figura 6.26 *Especificación de Archivos Fuente.* 74
Figura 6.27 *Adición de Archivos Fuente HDL.* 75
Figura 6.28 *Especificación de los Buses de Comunicación.* 76

Figura 6.29 *Reporte de Configuración del Bus FSL.* 77
Figura 6.30 *Finalización y Reporte de Datos de la Configuración.* 78
Figura 6.31 *Adición de una Nueva IP Core.* 79
Figura 6.32 *Configuración del Bus FSL en el MicroBlaze.* 80
Figura 6.33 *Conexión de Buses FSL.* 81
Figura 6.34 *Configuración de Puertos FSL.* 82
Figura 6.35 *Diagrama de Bloques del Sistema Implementado.* 83
Figura 6.36 *Configuración de los Parámetros Software del MicroBlaze.* 85
Figura 6.37 *Ventana para Agregar un Nuevo Proyecto de Software.* 86
Figura 6.38 *Identificación de la Nueva Aplicación* 86
Figura 6.39 *Adición de Códigos Fuente al Proyecto de Software* 87
Figura 6.40 *Programa de Prueba para el MicroBlaze.* 87
Figura 6.41 *Ejecución del Programa de Prueba* 88
Figura 7.1 *Tarjeta EOS Desarrollada por el CEIT* 89
Figura 7.2 *Periféricos de la Tarjeta EOS* 90
Figura 7.3 *Generación del Modelo en el CoreGenerator* 92
Figura 7.4 *Definición de Parámetros para la IP Core.* 93
Figura 7.5 *Configuración del Sincronismo de las Entradas y Salidas.* 94
Figura 7.6 *Configuración del Protocolo de Acceso.* 94
Figura 7.7 *Adición de Canales FSL.* 96
Figura 7.8 *Inicio del Asistente para Configuración de Periféricos.* 102
Figura 7.9 *Adición de Códigos Fuente y Archivos Netlist.* 102
Figura 7.10 *Actualización de los Repositorios del Proyecto.* 103
Figura 7.11 *Configuración de de Puertos y Canales FSL.* 104
Figura 8.1 *Representación del modelo de fuerza de pared elástica.* 109
Figura 8.2 *Comparativa de los modelos elástico y viscoelástico* [Gil, J. J, 2003]. 111
Figura 8.3 *Comparativa elástico vs. viscoelástico no lin*eal [Gil, J. J, 2003]. 113
Figura 8.4 *Comparativa de los modelos elástico y semiviscoelástico* [Gil, J. J, 2003]. 114
Figura 8.5 *Gráfico de Fuerza vs. Tiempo* [Salcudean, 1996]. 116
Figura 8.6 *Gráfico de Penetración vs. Tiempo* [Salcudean, 1996]. 117
Figura 9.1 *Cálculo de Seno y Coseno en la IPSC.* 119
Figura 9.2 *Cálculo de Seno y Coseno en el MicroBlaze.* 120
Figura 9.3 *Cálculo de Seno y Coseno Utilizando Unidad de Coma Flotante* 121

Figura 9.4 *PHANToM 1.0 Premium*.....121
Figura 9.5 *Lazo de Control Implementado en Posición.*.....122
Figura 9.6 *Lazo de Control Implementado en Fuerza.*.....123
Figura 9.7 *Dimensiones y parámetros del PHANToM 1.0.*.....124
Figura 9.8 *Relación de diámetros entre el motor y la articulación*.....125
Figura 9.9 *Aplicación del algoritmo de Denavit-Hartenberg*.....127
Figura 9.10 *Modelo obtenido con la Robotics Toolbox de Matlab*.....128
Figura 9.11 *Cálculo de la Cinemática Directa del PHANToM.*.....130
Figura 9.12 *Cálculo de Pares Utilizando la Jacobiana Transpuesta.*.....132
Figura 9.13 *Tiempo Empleado en el Lazo de Control Utilizando la IPSC.*.....133
Figura 9.14 *Tiempo Empleado en el Lazo de Control Implementado en Software.*.....134
Figura 9.15 *Prueba 1 Modelo Elástico.*.....135
Figura 9.16 *Prueba 2 Modelo Elástico.*.....136
Figura 9.17 *Prueba 3 Modelo Elástico.*.....136
Figura 9.18 *Prueba 1 Modelo Viscoelástico.*.....137
Figura 9.19 *Prueba 2 Modelo Viscoelástico.*.....138
Figura 9.20 *Prueba 3 Modelo Viscoelástico.*.....138

RESUMEN GENERAL DE TRABAJO DE GRADO

TITULO: OPTIMIZACIÓN DEL HARDWARE EMBEBIDO EN LA FPGA DE LA TARJETA EOS Y LA VALIDACIÓN PARA SU USO EN DISPOSITIVOS HÁPTICOS

AUTOR(ES): JAVIER MAURICIO MELO URIBE

FACULTAD: Facultad de Ingeniería Electrónica

DIRECTOR(A): CLAUDIA RUEDA, EMILIO SANCHEZ

RESUMEN

Este proyecto final de carrera tiene como objetivo, optimizar la arquitectura de hardware embebido que posee la tarjeta EOS diseñada por el CEIT. A partir de este objetivo, se realiza un estudio investigativo enfocado en la mejora de el tiempo de ejecución de las funciones trigonométricas que se emplean en el diseño de modelos de control de sistemas hápticos. Para llevar al cumplimiento del objetivo final, es necesario realizar un recuento de los avances tecnológicos desarrollados en el campo de los dispositivos lógicos programables, pues la tarjeta EOS está construida en base a una FPGA. Otro aspecto importante a tener en cuenta, es el conocimiento de las distintas herramientas para el diseño de modelos hardware, basados en lenguaje HDL. Este trabajo se centra en las herramientas de desarrollo que proporciona Xilinx y además se presentan otras aplicaciones que permiten la edición de código sintetizable a partir de modelos gráficos. Por otro lado, se realiza una completa descripción de las etapas que se deben seguir, para realizar la construcción de un sistema embebido, utilizando hardware reconfigurable. De esta manera se presenta un recuento a modo de manual de usuario, para conocer el funcionamiento de la plataforma de diseño EDK de Xilinx. De esta forma con este trabajo se aporta una herramienta para comprender el funcionamiento, programación y mejora de la tarjeta EOS, brindando un soporte desde el punto de vista teórico y práctico, para futuras aplicaciones que sean desarrolladas con esta tarjeta.

PALABRAS CLAVES: Hápticos, Realidad Virtual, sistema embebido, Hardware reconfigurable.

ABSTRACT OF THESIS PROJECT

TITLE: OPTIMIZATION AND VALIDATION OF FPGA'S EMBEDDED HARDWARE OF EOS BOARD TO USE ON HAPTICS DEVICES CONTROL.

AUTHOR(S): JAVIER MAURICIO MELO URIBE

DEPARTMENT: Electronic Engineering

DIRECTOR: CLAUDIA RUEDA, EMILIO SANCHEZ

ABSTRACT

The present Thesis project has as principal goal, to optimize the EOS embedded board's hardware architecture designed by CEIT. The aim objective in this research is focused on improving the execution time of trigonometric functions that are used in the design of haptics control systems. To accomplish this goal, it is necessary to make a state of the art about the technological advances in the programmable logic devices, principally because EOS card is built on a FPGA. Another important aspect to consider is the knowledge of different tools that are used to hardware's design based on language HDL. This work focuses on the development tools provided by Xilinx and other applications that allow editing synthesized code from graphical models. Also, there is a complete description of the steps that must be followed to realize the design and implementation of an embedded system using reconfigurable hardware. In this way it presents a summarize user's manual to learn the operation of the platform designed by Xilinx EDK. This work provides the necessary tools for understanding the operation, programming and improving of the EOS board, providing theoretical and practical support, for future applications that could be developed with this system.

KEYWORDS: Embedded system, FPGA, Haptic Device

1 INTRODUCCIÓN

¿Acaso es posible conocer algo tan sólo a través del tacto? Desde el punto de vista cognitivo, la función táctil ha sido un centro de discusión científica, aportando distintas apreciaciones que en algunos casos llegan a ser contradictorias. En ocasiones el tacto es definido como una función sensorial independiente, que permite a los individuos establecer una relación directa con el entorno que los rodea. Pero en otros casos, la función táctil es asumida como parte de la percepción visual, integrando un sistema redundante que permite la adquisición de información espacial y que relaciona al individuo con el ambiente y las propiedades de los objetos con los que entra en contacto en su actividad cotidiana. Es el caso de las personas ciegas, que a pesar de no contar con la función visual, logran a través del sentido del tacto crear una aproximación del entorno que les rodea.

Por otro lado, el predominio de la función visual parece ser tan evidente en las personas con visión normal, que se asigna una utilidad menor al tacto, reduciendo sus dominios sólo a la interacción directa con el entorno, en actividades como la corrección de postura o la de sujetar objetos ya sea con el fin de ser movidos o transformados.

El tacto difiere de la visión y la audición en la medida que éste depende del contacto que sus receptores puedan establecer con el medio circundante, y que además los receptores se encuentran distribuidos en todo el cuerpo. Por esta razón se dice que el tacto tiene una limitación proximal y su estudio se limita a la zona de contacto entre los objetos y el individuo [Hatwell, 2000].

Es necesario distinguir dos aspectos, que aunque resultan complementarios con frecuencia se mezclan conduciendo a interpretaciones erróneas. La percepción cutánea y la percepción cinestésica pueden ser identificadas, porque esta última, utiliza el sistema motor y propioceptivo en la actividad exploratoria de la mano, lo que a su vez activa el sistema hombro–brazo-mano.

En la percepción cutánea debido a que el segmento corporal es de carácter estacionario, sólo las capas superficiales de la piel que se encuentren bajo la acción de deformaciones mecánicas participarán en las actividades de procesamiento perceptivo.

En la percepción cinestésica, las deformaciones de los músculos, articulaciones y tendones resultantes de la actividad exploratoria, son añadidas a la percepción cutánea. Por lo tanto, el procesamiento de la actividad háptica es mucho más complejo, porque integra señales cutáneas y prioceptivas. Los movimientos exploratorios dependen directamente de la actividad neuronal, sin importar que éstos sean de tipo involuntario (se generan internamente, sin estimulo externo) o voluntario (orientado a un objetivo específico).

1.1 ACERCAMIENTO A LOS DISPOSITIVOS HÁPTICOS

Una interfaz háptica es un dispositivo motorizado e instrumentado que permite a un usuario humano, tocar y manipular objetos dentro de un entorno virtual [2]. La interfaz háptica interviene entre el usuario y el entorno virtual, obteniendo como resultado, un contacto mecánico con el usuario y un contacto eléctrico con el entorno virtual (*Fig.1.1)*.

El controlador es el encargado de reproducir señales de fuerza y movimiento, a partir de un evento de colisión, que después serán percibidas por el usuario a través del dispositivo mecánico acoplado al sistema.

Es el dispositivo mecánico quien recibe las fuerzas calculadas por el controlador y las convierte en fuerzas reales que serán realimentadas al usuario a través de los actuadores que posee.

Un dispositivo háptico diseñado y controlado debidamente, reflejará de forma realista sobre el mecanismo robótico, el comportamiento de un objeto inmerso en un entorno virtual, permitiendo al usuario interactuar con el entorno simulado existente sólo dentro del computador.

Figura 1.1 *Interacción con el Sistema Háptico.*

1.2 DESARROLLO TECNOLÓGICO DE LOS SISTEMAS HÁPTICOS

En gran medida, los sistemas hápticos surgen como la evolución de los sistemas teleoperados, campo que había sido objeto de estudio anteriormente en diferentes ámbitos de la robótica.

Un sistema teleoperado relaciona a un usuario humano con un ambiente físico remoto (ambiente real), a diferencia del sistema háptico donde el ambiente con el que interactúa el usuario es de carácter virtual creado de forma computacional.
Los sistemas teleoperados poseen un manipulador maestro y otro esclavo. El manipulador maestro tiene una funcionalidad similar a la de un sistema háptico, es decir, es manejado directamente por el usuario. Mientras que el manipulador esclavo es quien establece un contacto mecánico directo con el ambiente remoto.

Los primeros sistemas teleoperados fueron desarrollados en 1940, y su aplicación específica era en el manejo de materiales radioactivos. Estos sistemas se empleaban como extensión de los brazos y las manos del operario, para permitir la manipulación directa de elementos peligrosos. De esta manera el operario estaba aislado del contacto directo con el área de trabajo, pero sin embargo se obtenía una realimentación de la sensación táctil muy parecida a la que se generaba con la manipulación directa.

Cuando se introdujo el control electromecánico en los sistemas teleoperados, únicamente se tenía motorización en el manipulador esclavo, reproduciendo en este, los movimientos y las fuerzas captadas mediante sensores en el manipulador maestro. Pero sin la motorización en el manipulador maestro, la capacidad de manipulación y la sensación táctil del usuario disminuían, comparados con la sensibilidad percibida anteriormente mediante el acoplo mecánico directo, pues la interacción de fuerzas presentes en el manipulador esclavo no podía ser realimentada en este modo de operación.

Con la introducción de motores en el manipulador maestro, las fuerzas o posiciones (error dinámico) captadas por los sensores en el manipulador esclavo, podían ser reflejadas o realimentadas al usuario. Por lo tanto, el concepto de interacción bilateral de fuerzas fue introducido, por consiguiente, la manipulación y la plena capacidad de sentir el vínculo mecánico fueron reestablecidos.

En analogía con el desarrollo de los sistemas teleoperados, el diseño de dispositivos para interactuar con entornos virtuales, se inició con mecanismos unidireccionales que no permitían obtener al usuario una realimentación táctil de los eventos que ocurrían durante las experiencias simuladas. Es el caso de los guantes de realidad virtual o de otros mecanismos de interacción como el ratón o el joystick, que no cuentan con motorización que permita realimentar en fuerza la interacción mecánica entre el usuario y el ambiente virtual [Kurfess, 2005].

Actualmente la utilización de simuladores es muy frecuente, pues constituye una herramienta de entrenamiento que aporta todo el realismo de una situación cotidiana, pero alejado de los peligros y costos que un evento real implica. Pero incrementar el grado de realismo de la simulación para que el usuario se sienta inmerso en una situación real, requiere la inclusión de dispositivos que faciliten la interacción entre el ambiente simulado y el usuario.

Un caso muy particular es el de los simuladores de vuelo, donde es necesario que el usuario obtenga una simulación muy cercana a la realidad, no sólo desde el punto de vista visual, también se requiere que el ambiente recree los fenómenos dinámicos que ocurren durante el vuelo de un avión. Este objetivo se cumple agregando a los

dispositivos de la cabina de simulación, una plataforma móvil recree la dinámica del evento simulado.

De esta manera, una interfaz háptica es a la vez un dispositivo de entrada y de visualización, con dos vías de flujo de información a través de un único contacto mecánico.

1.3 CONTROL DE DISPOSITIVOS HÁPTICOS

Como en la implementación de cualquier algoritmo de control, el estudio de la estabilidad del sistema, resulta ser un tema clave para obtener un óptimo desempeño del sistema en cuestión. Desde cualquier tipo de oscilación o comportamiento inestable, puede provocar una realimentación desde el ambiente virtual que sea poco natural o realista y en algún caso, perjudicial para el usuario.

El análisis de los sistemas hápticos, no resulta una tarea fácil, por el contrario, se puede incluso llegar a difíciles soluciones que utilizan parámetros basados en las técnicas de control no lineal.

Es muy común encontrar que los modelos hápticos obtenidos, sean de carácter no lineal. Esto se debe principalmente a que se introduzca como parte fundamental del análisis realizado, el modelo dinámico del robot y del usuario [Siciliano, 2006].

Una herramienta muy útil para asegurar la estabilidad de un dispositivo háptico, es aplicar la teoría de la pasividad. Como ya se ha definido antes, un sistema háptico no es más que una interfaz robótica interactiva, de esta manera, si el sistema se comporta de forma pasiva, entonces se puede garantizar que el sistema sea estable.

Además se puede establecer que el comportamiento de un usuario humano, es pasivo en el rango de frecuencias de interés para los sistemas hápticos. De esta manera, se puede generalizar el estudio en modelos de tipo lineal o no lineal, para caracterizar tanto los dispositivos como los ambientes virtuales empleados sin que ello afecte la estabilidad del sistema.

El estudio de la pasividad, consigue asegurar de manera efectiva la estabilidad del sistema para cualquier usuario que lo manipule, pero no cabe duda que es una condición mucho más restrictiva que la estabilidad [Gil, J. J, 2003].

1.4 LIMITACIONES DE LOS SISTEMAS HÁPTICOS

Las distintas aplicaciones desarrolladas en el campo de los dispositivos hápticos, resultan ser de un alto nivel de complejidad, requiriendo para su implementación de hardware especializado y un considerable consumo de potencia. En nivel de realismo requerido,

implica realizar un gran desarrollo tanto a nivel de hardware, como también en estructuras de programación para controlar la actividad del dispositivo.

La complejidad de estos dispositivos, obliga a contar con instalaciones y equipamiento que dificultan la portabilidad del sistema. El desarrollo de dispositivos hápticos versátiles y de perspectivas comerciales, implica que éstos sean económicos, portables y con un reducido consumo de potencia.

Tanto la incertidumbre de la dinámica del usuario como la dinámica de la interfaz, implican que el sistema tomado en su conjunto, sea de carácter no lineal. Esto implica que la complejidad en el diseño e implementación de controladores aumente.

Las limitaciones de hardware de las interfaces hápticas reducen la fidelidad con la que se puede simular la interacción en una sesión háptica. Mejoras en la precisión de los sensores, la capacidad de los actuadores, o la transparencia del dispositivo mecánico empleado, son indispensables para obtener resultados satisfactorios. Para que se obtenga un grado de realismo óptimo, los hápticos deben operar a frecuencias superiores a 1 khz; por debajo de ésta frecuencia la calidad de la simulación decrece considerablemente y de allí la necesidad de utilizar tarjetas de adquisición de datos de alto rendimiento para satisfacer estos requerimientos de funcionamiento.

Comercialmente existen tarjetas de adquisición de datos con las cuales se realiza la captura, tanto de posición como de fuerza, para poder cerrar el lazo de control. Estas tarjetas poseen un costo muy alto, debido a que suelen ser computadores embebidos o complejos dispositivos de procesamiento de señales. Pero a su vez aportan grandes ventajas desde el punto de vista de programación, pues en la mayoría de los casos poseen compatibilidad con entornos de programación de alto nivel como LabView o Simulink.

Si se tiene en cuenta la inversión económica que se requiere actualmente para contar con una interfaz háptica, nace una vía para la investigación y el desarrollo en este campo, que permita la masificación de esta tecnología, pues aún se encuentra relegada a labores investigativas o en instalaciones muy complejas en donde su uso es indispensable.

En el caso específico de este trabajo de investigación, se pretende encontrar una solución de bajo costo implementada en FPGA, para reemplazar las tarjetas de adquisición de datos comerciales que actualmente son utilizadas en el desarrollo de hápticos en el CEIT.

1.5 APLICACIONES DE LOS HÁPTICOS

El campo de aplicación de los dispositivos hápticos es muy amplio y continuamente se realizan nuevos aportes, debido a la utilización cada vez más frecuente de equipos electrónicos en las actividades cotidianas.

Principalmente los hápticos han se han empleado en la medicina, el diseño mecánico, el entretenimiento y las artes gráficas. Siempre con la premisa de utilizar herramientas reales pero reflejando su efecto en un ambiente virtual.

1.5.1 Aplicaciones Médicas

Manipulación de micro y macro robots para cirugías mínimamente invasivas. De igual forma existen aplicaciones de diagnóstico remoto, en donde el personal médico puede tratar a un paciente a distancia, pero sin embargo a través de manipuladores, logra obtener manifestaciones táctiles para completar un diagnóstico preciso.

El entrenamiento médico también posee importantes aplicaciones de desarrollo en los sistemas hápticos. Se reduce el riesgo y se aumenta el nivel de preparación de los estudiantes, practicando en simuladores que recrean situaciones encontradas en la realidad.

Figura 1.2 *Aplicación de Telemedicina.*

1.5.2 Aplicaciones en el Diseñó Mecánico

Se ha desarrollado la integración de los dispositivos hápticos en sistemas CAD (Computer Aided Design), en los que el diseñador puede libremente manipular componentes mecánicos, inmerso en un ambiente de ensamblaje. De esta forma se pueden simular procesos de montaje industrial, no sólo visualmente sino además con reflexión de fuerzas. Sensaciones de peso y el contacto mecánico entre superficies contribuye en las tareas de aprendizaje del personal en actividades relacionadas con su área de trabajo específica.

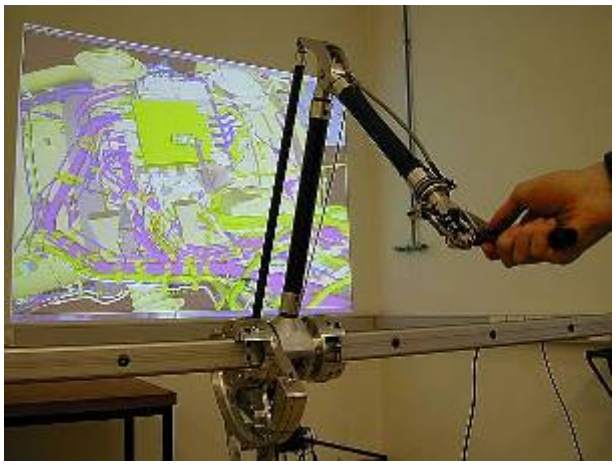

Figura 1.3 *LHIfAM Diseñado por el CEIT.*

1.5.3 Aplicaciones en la Industria del Entretenimiento

El mercado del entretenimiento se encuentra en continua expansión. Existen todo tipo de consolas de video juegos y simuladores, que pretenden trasportar a sus usuarios a ambientes simulados, donde puedan experimentar con acciones que en la vida real no puedan llevar a cabo. Deportes extremos, de aventura, simuladores de vuelo y toda clase de actividades que realizan las personas, son elaboradas dentro de un ambiente virtual. Por medio de los dispositivos hápticos el usuario puede sentir y manipular elementos virtuales generados en estos ambientes, como herramientas, fluidos o interactuar con todo tipo de situación.

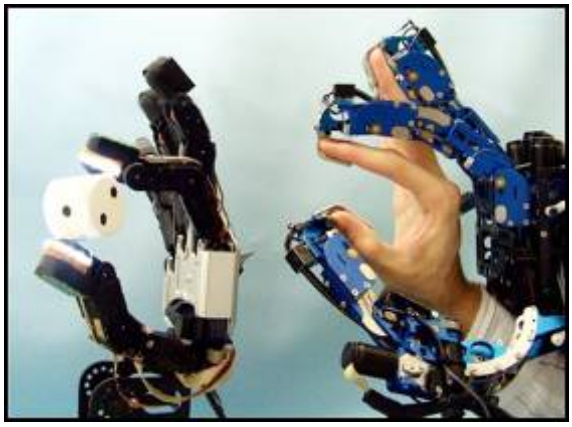

Figura 1.4 *Manipulador para Videojuegos.*

1.5.4 Aplicaciones en las Artes Gráficas

Exhibiciones virtuales de arte, salas de concierto y museos en los cuales el usuario puede acceder remotamente para interactuar con instrumentos musicales o en algunos casos tocar y sentir los atributos de los elementos presentes.

Figura 1.5 *Diseño Gráfico.*

También se han introducido dispositivos hápticos en el campo del diseño gráfico, donde el usuario puede utilizar como mecanismo para interactuar con el sistema informático de diseño, dispositivos que se asemejan a las herramientas de trabajo que usaría en la realidad. Esto otorga mayor versatilidad a las actividades de diseño, debido a que para el usuario la interfaz es transparente y no sugiere para este un cambio representativo en sus actividades cotidianas.

2 PLANTEAMIENTO DEL PROBLEMA

Se dice que todo problema nace a raíz de una dificultad. En el presente capítulo se presenta una descripción global de los factores que motivaron la realización de este proyecto y que en su conjunto configuran el objeto o problema de investigación. Además se incluye un recuento de las soluciones que actualmente se le han dado a éstas dificultades y se hace hincapié en las mejoras que este trabajo puede llegar a aportar.

2.1 OBJETIVO GENERAL

- Implementar el sistema de control de un dispositivo háptico de 1Gdl, en un procesador de bajo costo y desarrollo autónomo como lo es una FPGA.

2.2 OBJETIVOS ESPECIFICOS

- Establecer el estado del arte correspondiente al diseño y modelado de sistemas hápticos desde el punto de vista del control.

- Analizar los modelos de contacto y de control por impedancia que se pueden aplicar en el desarrollo de una palanca háptica de 1Gdl.

- Sugerir correcciones y adaptar los algoritmos de control de una palanca háptica de 1gdl, para que sea implementado correctamente en una FPGA.

- Comprobar experimentalmente el comportamiento del sistema háptico desarrollado en comparación con los resultados obtenidos implementando el control en un equipo comercial.

- Proponer posibles mejoras y aplicaciones futuras, para el sistema háptico desarrollado, en diferentes campos de la ingeniería y la industria.

2.3 DEFINICIÓN DEL PROBLEMA

Muchas de las aplicaciones desarrolladas en el campo de los sistemas hápticos, se ven altamente restringidas por la velocidad de procesamiento de información de los equipos encargados de controlar el sistema.

Es sabido que el cálculo de funciones matemáticas de cierta complejidad en sistemas programables de bajo nivel como los microcontroladores resulta ineficiente. En el caso de los hápticos se puede decir, que precisan de respuesta inmediata para no incidir sobre la estabilidad del sistema controlado. Al mismo tiempo se descarta el uso de un sistema informático complejo, como lo es un computador personal, pues se desaprovecha en gran medida la potencia de cálculo que este posee, además incrementa el costo del sistema implementado.

Comúnmente se utilizan como sensores de posición, encoders acoplados al rotor del motor. Debido a la precisión requerida y a la alta frecuencia a la que operan los encoders, no es posible a través de software leer y procesar el volumen de información necesario para que el sistema funcione de correctamente, siendo imperativo utilizar dispositivos que permiten realizar este tipo de operación en Hardware. El costo y la poca versatilidad de los equipos que comercialmente se consiguen, conducen a encontrar una solución práctica a esta necesidad.

Con la aparición de dispositivos programables en lenguaje de descripción de Hardware VHDL, se abre una opción para desarrollar sistemas de procesamiento de información que no dependan de la velocidad de ejecución de instrucciones programadas en software, por el contrario, esta operación queda totalmente asignada en Hardware, aumentado así el rendimiento del sistema diseñado.

Actualmente para desarrollo de dispositivos hápticos en el laboratorio de mecánica aplicada del CEIT (Centro de Investigaciones Técnicas de Guipuzkoa), se utilizan sistemas de procesamiento comerciales y de allí la necesidad de crear un sistema de bajo costo desarrollado totalmente por el CEIT, que permita cumplir con los requerimientos de funcionamiento de los hápticos y al mismo tiempo ganar versatilidad y autonomía en el desarrollo de estos equipos.

2.4 JUSTIFICACIÓN

Desde el inicio del las ciencias informáticas, se ha pretendido simular de manera virtual las situaciones y actividades que el hombre realiza en su vida cotidiana, es decir se pretende llevar a un mundo simulado, el entorno real en el que el hombre se desenvuelve.

Preservar la seguridad e integridad de las personas, se convierte en uno de los principales objetivos del desarrollo tecnológico actual; evitar que el hombre se enfrente a situaciones en las que pueda poner en peligro no sólo su integridad sino también la de

otros, permite explorar nuevos campos de investigación que lleven a encontrar soluciones tecnológicas que garanticen un aprendizaje eficaz, pero desarrollado en un ambiente virtual alejado de las posibles contingencias del mundo real.

Debido al avance y complejidad de los sistemas mecánicos actuales, cada vez es más difícil establecer una relación directa entre el operario y el elemento final de actuación del dispositivo que este emplea en su actividad; es decir, se obtiene tan sólo una realimentación visual de las operaciones que se están llevando a cabo, pero el operario no puede establecer por medio de sus demás sentidos, los fenómenos que se presentan en el proceso que se está ejecutando.

Históricamente el ser humano ha interactuado con los computadores por medio de canales unidireccionales. Información visual y auditiva es transmitida desde el computador hacia el operador, como respuesta al proceso ejecutado. Del mismo modo a través de periféricos como el teclado, joystick o el ratón se transfieren los requerimientos del operador a la máquina.

Una interfaz háptica expresa una sensación cinestésica a un operador humano, que interactúa con un ambiente virtual generado por computador. De esta manera el háptico permite a un usuario tocar, sentir, manipular, crear, y cambiar objetos tridimensionales simulados dentro de un ambiente virtual, añadiendo el sentido del tacto a la experiencia virtual. El usuario puede no sólo enviar información al ambiente virtual, sino también recibir o retroalimentar la comunicación en forma de sensación táctil sobre alguna parte del cuerpo.

Ahora bien, debido a la complejidad de los cálculos que se deben realizar para estimar correctamente las relaciones cinemáticas de la etapa mecánica del sistema y de esta manera obtener con precisión la respuesta en fuerza, es necesario contar con equipamiento capaz de llevar a cabo este proceso sin que esto disminuya las prestaciones del sistema. Comercialmente es posible encontrar una alta variedad de equipos que cumplen con los requerimientos exigidos, pero su uso está limitado debido al costo económico que sugiere la adquisición de este tipo de elementos.

Con miras a la masificación comercial y a nivel investigativo de los sistemas hápticos, se hace necesario encontrar soluciones que permitan reducir los costos en el ensamblaje del mismo. De esta manera, se pretende implementar el control de un sistema háptico, utilizando para ello una plataforma de desarrollo de bajo costo y gran versatilidad como lo es una FPGA, que permita obtener resultados de funcionamiento óptimo pero al mismo tiempo tener las ventajas y facilidades que brinda un equipo de desarrollo propio.

La implementación de un sistema embebido desarrollado en la FPGA, apunta reunir de manera sistemática las ventajas que tiene la implementación de funciones directamente en hardware (VHDL), pero además tener acceso a un entorno de desarrollo de software como lo es µclinux, que permite realizar una programación eficiente y de alto rendimiento que es imprescindible en esta aplicación.

Es de resaltar que para la consecución de los objetivos propuestos, será necesario analizar y discriminar los elementos que conforman la etapa de control del háptico, identificando cuales del ellos se deben implementar en lenguaje de descripción de

Hardware VHDL para posteriormente ser sintetizados y alojados en la FPGA (Spartan III, Xilinx) y cuales dado su poca complejidad de cálculo pueden mantenerse operando directamente en software, para garantizar de esta manera, que la dinámica del sistema propuesto sea estable.

2.5 SOLUCIONES ACTUALES A LA PROBLEMÁTICA

La construcción de un dispositivo háptico requiere afrontar diferentes retos tecnológicos que permitan llegar a la conclusión de los objetivos planteados. Actualmente el Departamento de Mecánica Aplicada del CEIT, traza como línea de trabajo la investigación y desarrollo de interfaces hápticas, apuntando a una evolución tecnológica de diseño propio.

El desarrollo e implementación de una interfaz háptica requiere la unión de un dispositivo mecánico que sea transparente para el usuario y que le permita a este interactuar con el ambiente virtual.

Del mismo modo es necesario dotar al dispositivo háptico, con una etapa electrónica que permita capturar la información proveniente de los sensores y a su vez alojar el sistema de control que calculará la respuesta que se ha de enviar a los actuadores.

Actualmente se utilizan como tarjetas de control y adquisición de datos, dispositivos comerciales DSP dSPACE 1104 que brindan un soporte hardware adecuado para aplicaciones que posean máximo dos grados de libertad (2 GdL) si se usan encoders, o hasta 8 GdL si se emplean para la adquisición de datos las entradas analógicas de la tarjeta.

Esta restricción se observa en la *Fig. 2.1.* dónde se puede apreciar que esta tarjeta sólo cuenta con dos entradas para lectura de encoders.

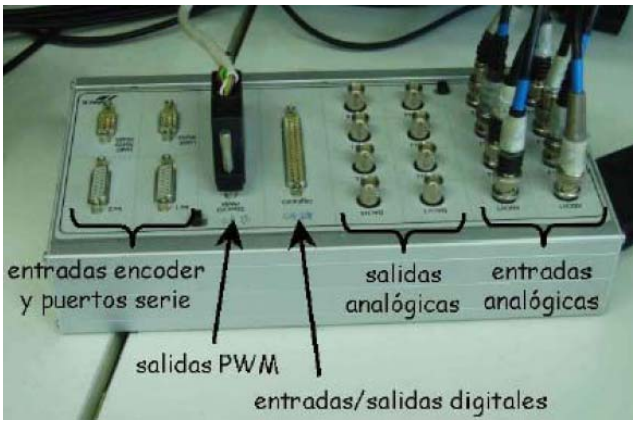

Figura 2.1 *DSP dSPACE 1104.*

Los dispositivos hápticos que se desarrollan actualmente en el CEIT, poseen más de 2 GdL implementados con encoders, haciendo necesario añadir electrónica adicional a la tarjeta para efectuar la lectura de dichos encoders.

Esta solución aunque es funcional, resulta problemática pues a medida que aumenta el número de grados de libertad empleados, igualmente aumenta la electrónica adicional necesaria. Además de la restricción física que presenta la tarjeta, existe otro factor importante a tener en cuenta; el precio de este tipo de dispositivo oscila entre los 4000€ y 9000€, dependiendo de las aplicaciones que se le vayan a dar a la tarjeta (educacional o licencia comercial).

Teniendo en cuenta todo lo descrito, resulta conveniente encontrar una solución de bajo costo que permita conectar dispositivos hápticos de 6 GdL y a su vez implementar el sistema de control utilizando programación de alto nivel para tal fin.

3 ESTADO DEL ARTE

Existe abundante bibliografía referente a las aplicaciones, desarrollo y evolución de los dispositivos Hápticos. Este capítulo se centra en aquellas publicaciones en las que se menciona una relación directa entre el uso de FPGAs y el control de dispositivos Hápticos, observando las soluciones presentadas y teniendo en cuenta las ventajas e inconvenientes que se han expuesto en cada una de ellas.

3.1 SEGMENTACIÓN DE INSTRUCCIONES Y PROCESOS

La segmentación tanto de instrucciones como de procesos permite el mejor aprovechamiento de los recursos de procesamiento de los equipos electrónicos. Esta segmentación es similar al orden implantado en una línea de montaje industrial, en donde un producto pasa por distintas estaciones de montaje, antes de llegar a obtener un producto totalmente terminado.

Cada una de estas estaciones a pesar de ser independientes, se encuentran especializadas en la tarea que les ha sido asignada y siempre realizan la misma actividad a lo largo de la etapa de producción. De esta manera por cada ciclo de trabajo habrá un producto totalmente terminado que saldrá de la línea de montaje, mejorando la productividad y la calidad del proceso (*Fig. 3.1).*

Tomando como referencia el ejemplo de la línea de montaje y aplicándolo en la arquitectura de un sistema de procesamiento, se puede inferir que teniendo dispositivos especializados en una tarea específica e interconectándolos en una única unidad de trabajo, se puede incrementar la eficiencia global del dispositivo.

En [Pearson, 2006] se hace referencia a la utilización de técnicas de segmentación de instrucciones (pipeline) implementadas directamente en hardware sobre una FPGA. El modelo de control utilizado, es segmentado en diferentes bloques funcionales que se implementan por separado en la FPGA. A cada bloque se le asigna como entrada, el resultado obtenido en el bloque anterior y a su vez éste lo transmitirá al bloque siguiente. Cuando todos los bloques funcionales hayan sido inicializados con un ciclo de trabajo, cada componente operará simultáneamente con una pequeña parte del algoritmo implantado, dando como resultado un máximo aprovechamiento de las utilidades hardware en un mínimo periodo de tiempo. Todo esto conlleva a un óptimo procesamiento de información en tiempo real y además se obtienen las ventajas del uso de hardware reconfigurable implementado en la FPGA.

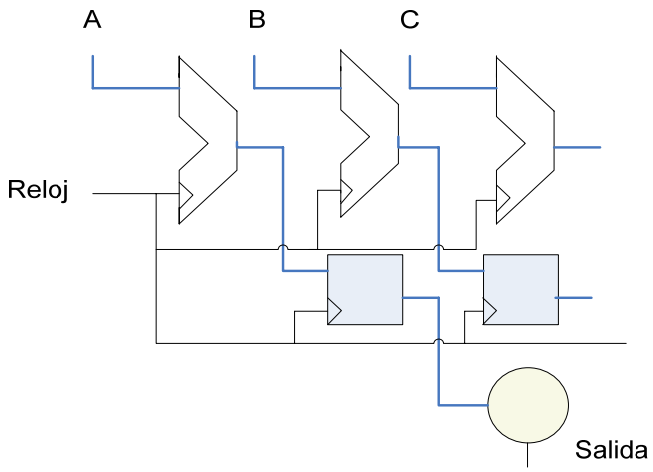

Figura 3.1 *Segmentación en Bloques Funcionales.*

3.2 CIRCUITOS ELECTRÓNICOS DEDICADOS

Existen varias maneras de aumentar la velocidad de cómputo de un dispositivo electrónico: procesador más rápido, múltiples procesadores o la implementación de circuitos dedicados.

Los circuitos dedicados permiten dividir un bloque de trabajo, y compartir recursos con el procesador central, ya sea este un microcontrolador o algún dispositivo de procesamiento superior. Esta alternativa resultaba costosa y de difícil implantación desde el punto de vista del hardware, pero con la masificación de las FPGAs hoy en día es una solución totalmente viable.

Generalmente existen dos opciones a la hora de implementar un algoritmo en un dispositivo electrónico: Utilizar un procesador de propósito general, lo cual resulta económico y de fácil implantación pero de eficiencia reducida, o utilizar hardware diseñado a medida, que eleva el rendimiento del dispositivo, pero al mismo tiempo su costo de fabricación. La lógica reconfigurable permite combinar estas dos soluciones y obtener altas velocidades hardware con flexibilidad en software, además reducir costos pues es posible reutilizar el hardware en varias aplicaciones, únicamente cambiando la programación interna [Oliver, 2001].

3.3 SISTEMAS DE COPROCESAMIENTO DE DATOS

Un coprocesador es una unidad de procesamiento local utilizada como suplemento de las funciones del procesador principal. Las operaciones ejecutadas por un coprocesador pueden ser de tipo aritmético en coma flotante, procesamiento gráfico, procesamiento de señales, procesado de texto o cualquier tipo de proceso que requiera un nivel considerable de potencia computacional.

La función de un coprocesador es evitar que el procesador principal tenga que realizar tareas de cómputo intensivo, acelerando el rendimiento del sistema como consecuencia de la descarga de trabajo en el procesador principal y porque suelen ser procesadores especializados que realizan las tareas para las que están diseñados más eficientemente.
Las características de los dispositivos FPGA actuales permiten incluir diseños específicos para una determinada tarea, y uno o varios microprocesadores embebidos, que pueden estar implementados en la propia lógica reconfigurable o insertados como IP Cores hardware.

Con el propósito de acelerar los cálculos requeridos para recrear un ambiente virtual de interacción háptica, en [Rebello, 2004] se utilizan algoritmos implementados en bloques de hardware dentro de una FPGA, con el fin de realizar el cálculo de colisiones entre los puntos de interacción de los objetos presentes en entorno virtual de trabajo, y además el cálculo de reflexión de fuerza de dichas colisiones. El uso de una FPGA como coprocesador de cálculo en esta aplicación, aporta mejoras en la velocidad de ejecución de la aplicación y vuelve flexibles las tareas de reconfiguración de los módulos hardware.

Existen muchas aplicaciones ya implementadas en módulos de hardware para FPGA que se insertan como dispositivos de coprocesamiento para mejorar la eficiencia de los sistemas; a continuación se presentan dos aplicaciones que tienen relación con el proyecto desarrollado: Módulo para controlador PID y Controlador para una interfaz háptica.

3.3.1 Módulo para Controlador PID

La utilización de controladores de tipo comercial, sugiere una inversión económica considerable más aún cuando se trata de un sistema de control digital. La utilización de hardware reconfigurable en FPGA para el diseño de un controlador digital PID se presenta en [Zhao, 2005], con el objetivo de analizar la velocidad de operación, el área física ocupada y el consumo de potencia del mismo (*Fig. 3.2)*.

La flexibilidad obtenida utilizando una FPGA como plataforma de desarrollo, otorga beneficios importantes, como la posibilidad de realizar tareas de reconfiguración en caliente, sin que ello implique la salida de operación del sistema.

El uso de microprocesadores embebidos dentro de la propia FPGA, hace posible tener un manejo eficiente del consumo de potencia. Se obtiene la capacidad de ejecutar el software diseñado y a la vez tener hardware reconfigurable, todo en un único CI; de allí la mejora notable en el consumo de potencia.

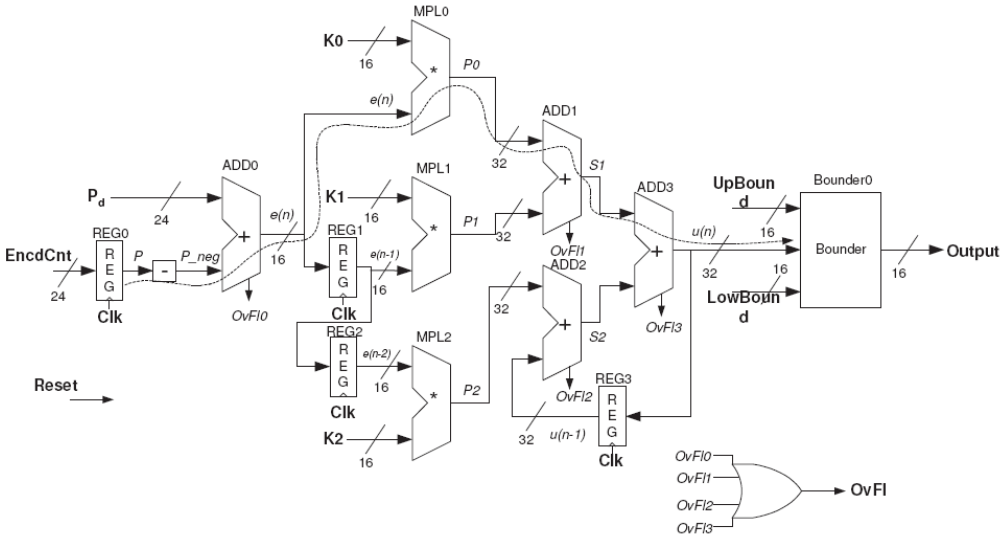

Figura 3.2 *Algoritmo de un PID implementado en Hardware Reconfigurable.*

Con frecuencia se utilizan técnicas de codiseño Hardware – Software, para dividir de manera óptima las secuencias de código que se implementarán directamente en Software y las que dada su complejidad de cálculo serán alojadas en Hardware. La combinación de las técnicas de codiseño y la flexibilidad para reconfigurar en tiempo de ejecución, logran optimizar la asignación de funciones entre a etapa Hardware en FPGA y el uso de Software dinámico.

3.3.2 Controlador para Interfaz Táctil

Una de las aplicaciones más frecuentes implementadas en Hardware sobre FPGA, es la adquisición y tratamiento de señales. Comercialmente existen dispositivos que cumplen con los requerimientos técnicos necesarios en este tipo de aplicación, pero resultan costosos y poco flexibles, pues se está limitado por las especificaciones del fabricante.

Los dispositivos hápticos necesitan para lograr emular el contacto entre un individuo y un ambiente virtual, múltiples entradas de datos provenientes de los sensores implantados en el sistema. Estos sensores muchas veces necesitan de un tratamiento independiente, para interpretar las mediciones que realizan, de este modo de acuerdo con el tipo de sensores que se emplee en determinada aplicación, será necesario contar con un dispositivo que interprete cada uno de los formatos empleados por cada sensor. Comúnmente en hápticos se emplean encoders diferenciales, dispositivos piezoeléctricos y acelerómetros para identificar las magnitudes físicas que intervienen en un evento

determinado. Todo esto requiere de un dispositivo con la capacidad de albergar todo el conjunto de señales sin que ello implique una inversión en hardware especializado por cada tipo de sensor. Del mismo modo es necesario contar con una interfaz de salida que permita transmitir a los actuadores la respuesta en fuerza calculada en el sistema.

Una solución económica y viable, es implementar el hardware requerido dentro de una FPGA. Construyendo un completo sistema embebido, se puede albergar en un mismo proyecto de hardware distintos bloques de adquisición y salida, dispuestos como IP Cores y controlados desde un microprocesador embebido en el mismo dispositivo.

En [Holbein, 2005] se presenta un controlador para interfaz táctil, implementado en FPGA, con el objetivo de utilizar una solución de bajo costo y consumo de potencia reducido, en la implementación de un sistema háptico (*Fig. 3.3)*. La utilización de la FPGA, permite superar limitaciones en la excitación tanto en amplitud como en frecuencia de los actuadores (vibradores) de un guante háptico. Al mismo tiempo permite recoger toda la información sensorial alojada en el guante y transmitirla al algoritmo de control.

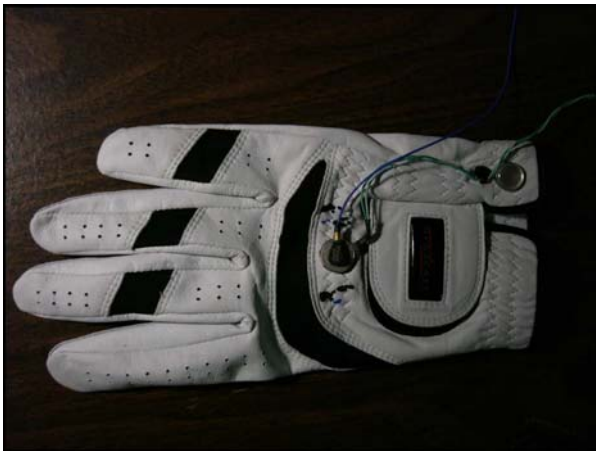

Figura 3.3 *Interfaz Háptica Controlada por FPGA.*

4 DISPOSITIVOS LÓGICOS PROGRAMABLES

En este capítulo se presenta un recuento de la evolución tecnológica de los dispositivos lógicos programables desde sus inicios hasta los últimos aportes realizados en este campo, esto con el fin de situar el presente trabajo en la línea de desarrollo tecnológico actual.

Es necesario conocer la arquitectura interna de la FPGA para así entender el proceso de estructuración de un sistema electrónico embebido, que constituye uno de los aspectos más relevantes de este proyecto. A continuación se hace una breve descripción de la estructura interna de la Spartan 3 y a su vez se presentan las características más importantes del microcontrolador embebido MicroBlaze.

4.1 TIPOS DE DISPOSITIVOS LÓGICOS PROGRAMABLES

Un dispositivo lógico programable o PLD (Programmable Logic Device), es un dispositivo cuyas características pueden ser modificadas y almacenadas mediante programación. En los últimos años el uso de este tipo de dispositivo se ha incrementado notablemente dada su versatilidad, seguridad y ahorro económico que sugiere su utilización.

La lógica programable, como se puede inferir de su nombre, es una familia de componentes que contienen conjuntos de elementos lógicos (AND, OR, NOT, LATCH, FLIP-FLOP) que pueden configurarse en cualquier función lógica que el usuario desee y que el componente soporte.

Las arquitecturas empleadas, generalmente varían de acuerdo a la aplicación que se pretenda implementar, pero normalmente se utilizan matrices de compuertas AND y OR para representar todo el conjunto de funciones lógicas que se puedan presentar en el desarrollo de la aplicación.

Actualmente los PLDs se utilizan en casi todos los equipos de control industrial, telecomunicaciones, oficina y de uso doméstico, pues la tendencia que se ha impuesto hoy en día, es la de reducir el tamaño y aumentar la portabilidad de los equipos mencionados anteriormente.

Existen varias clases de dispositivos lógicos programables: ASICs, FPGAs, PLAs, PROMs, PALs, GALs, y PLDs complejos. Cada uno de ellos presenta un conjunto de características particulares que lo diferencia y que le aporta un valor agregado a la hora de ser elegido para una aplicación específica.

4.1.1 Definiciones y Terminología Empleada

A continuación se presentan los acrónimos utilizados para referirse a los dispositivos lógicos programables nombrados a lo largo de este texto:

FPD (Field Programmable Device): se refiere a cualquier tipo de circuito integrado utilizado para implementer Hardware digital, donde el circuito integrado (CI) puede ser configurado por el usuario final para realizar diferentes diseños. Otro nombre empleado para los FPDs es Dispositivo Lógico Programable (PLD); aunque los PLDs en su definición abarcan los mismos dispositivos que los FPDs, se ha designado esta especificación a un tipo particular de dispositivo.

PROM (Programmable Read Only Memory): es un PLD utilizado en su función básica como memoria, que puede ser programado indefinidamente por el usuario, ingresando un patrón de almacenamiento específico. Una PROM es un sistema combinacional completo que permite realizar cualquier función lógica con las n variables de entrada, ya que dispone de 2n términos productos. Están muy bien adaptadas para aplicaciones tales como: tablas, generadores de caracteres, convertidores de códigos, etc.

PLA (Programmable Logic Array): se considera a los PLAs como un FPD con funciones reducidas. Posee dos estructuras lógicas, un nivel AND y otro nivel OR; las dos estructuras son programables.

PAL (Programmable Array Logic): es una variación de los PLAs; poseen dos estructuras lógicas una fija (OR) y otra programable (AND). Esta configuración limita el número de términos OR que se pueden implementar. Su característica más relevante es la alta velocidad a la que pueden operar.

SPLD (Simple PLD): se refiere de manera general a cualquier tipo de PLD, usualmente a cualquier PLA o PAL.
CPLD (More Complex PLD): consiste en un arreglo de múltiples bloques de SPLDs en un CI individual. En distintas bibliografías son denominados como Enhanced PLD (EPLD), Super PAL o Mega Pal. Las CPLDs poseen la velocidad de operación de las PALs, pero son mucho más complejas.

FPGA (Field Programmable Gate Array): es un tipo de FPD construido con una estructura general que permite obtener una muy alta capacidad lógica. Las FPGAs ofrecen más recursos lógicos que las CPLDs pero al mismo tiempo permiten tener un amplio número de entradas. Son especialmente utilizadas en el prototipado de CI pues son fácilmente reprogramables y el código de programación empleado de reutilizable modularmente.

HCPLDs (High Capacity PLDs): es un sencillo acrónimo empleado para referirse tanto a CPLDs como FPGAs. Este término ha sido utilizado en la literatura técnica para proporcionar una fácil manera de referirse a éstos dos tipos de dispositivo.

4.1.2 Evolución de los Dispositivos Lógicos Programables

El primer tipo de dispositivo programable en el que se pudo implementar circuitos lógicos fueron los PROMs, en los cuales las líneas de direccionamiento pueden ser usadas como circuitos lógicos de entrada y las líneas de información como salidas. Las funciones lógicas, sin embargo, rara vez requieren más que unos pocos productos de términos, y un dispositivo PROM dedica un decodificador completo para el direccionamiento de este tipo de operación. Por lo tanto los PROMs, presentan una arquitectura ineficiente para implementar complejos circuitos lógicos y con poca frecuencia son utilizados en la práctica para este tipo de propósito.

Los primeros dispositivos desarrollados específicamente para implementar circuitos lógicos, fueron los Field Programmable Logic Array (FPLA), o simplemente denominados PLAs. Un PLA posee dos niveles de compuertas lógicas, un nivel AND y otro OR. Un PLA esta estructurado de manera que a cualquiera de sus entradas (o sus complementos) se pueda aplicar el producto lógico AND en su nivel respectivo. Cada salida del nivel AND corresponde por lo tanto a un producto de términos de sus entradas. De forma similar cada salida del nivel OR puede ser configurado para producir la suma lógica de cualquiera de las salidas del nivel AND. Con esta estructura, los PLAs son una buena solución para implementar funciones lógicas en forma de suma de productos. También son muy versátiles puesto que tanto los términos AND y los términos OR pueden tener n entradas (esta característica es definida a menudo como ancho de compuertas AND y OR).

Cuando los PLAs fueron introducidos por Philips, a comienzos de los años 70, se argumentó que resultaban muy costosos de fabricar y ofrecían un rendimiento muy bajo de velocidad. Ambas desventajas se debieron a la configuración de dos niveles lógicos implementada, pues dificultaba el proceso de producción y agregaban significativos retrasos de propagación.

Para superar estas deficiencias, fueron desarrollados los Programmable Array Logic (PAL). Los PALs presentaban un único nivel programable, consistente en un nivel AND que se comunica con un nivel de compuertas OR de carácter fijo. Para compensar la falta de generalidad adquirida con esta arquitectura, debido a la diferencia en número de entradas y salidas que se pueden obtener, diversas variantes se han producido con diferente número de entradas y salidas, y además con varios tamaños de compuertas OR.

Los PALs usualmente contienen flip-flops conectados a las salidas de las compuertas OR, con los cuales es posible implementar circuitos secuenciales. Los dispositivos PAL son importantes porque cuando se introdujeron provocaron un efecto profundo en el diseño de Hardware digital, y representan la base de algunas de las más sofisticadas arquitecturas utilizadas en la actualidad. Variantes de la estructura básica de los PALs son utilizadas en muchos otros productos conocidos con diferentes denominaciones o acrónimos.

Todos los PLDs básicos, incluyendo PLAs, PALs y variantes de los PAL, son dispositivos agrupados en una categoría independiente denominada Simple Plus (SPLDs), cuyas

características más importantes son su bajo costo y su alta velocidad de rendimiento pin a pin.

Como la tecnología se encuentra en un constante avance, se logró producir dispositivos con capacidades y prestaciones más altas que los primeros SPLDs. La dificultad con el incremento de la capacidad de la sólida arquitectura de los SPLDs, es que la estructura de los niveles lógico programables creció también rápidamente en tamaño cuando el número de entradas y salidas fue incrementando. La única forma factible de proporcionar gran capacidad en los dispositivos basados en arquitecturas SPLD, es entonces integrar múltiples SPLDs en un único chip y proporcionar una interconexión programable, que permita conectar conjuntamente los bloques de SPLDs. Muchos FPDs comerciales existen hoy en día en el mercado que utilizan esta estructura básica y son comúnmente nombrados como Complex PLDs (CPLDs).

Los primeros CPLDs presentados, fueron construidos por Altera, cuya primer familia de CI fue llamada Classic EPLDs, y en las siguientes tres series producidas se llamaron MAX 5000, MAX 7000 y MAX 9000. Debido al rápido crecimiento comercial presentado por los FPDs complejos, otros fabricantes desarrollaron dispositivos en la categoría de los CPLD y en la actualidad hay una gran variedad de opciones disponibles en el mercado.

Los CPLDs proporcionan una capacidad de procesamiento lógico superior a sus antecesores, equivalente a más de 50 SPLDs básicos (*Fig. 4.1)*, pero presentan una dificultad para extender este tipo de arquitectura en versiones de alta densidad. Para construir FPDs con alta capacidad lógica es necesario plantear otro enfoque. Los CI de propósito general de mayor capacidad lógica disponibles hoy en día utilizan una estructura tradicional de arreglo de compuertas, comúnmente denominadas como Mask Programmable Gate Arrays (MPGAs).

Los MPGAs poseen un arreglo prefabricado de transistores que pueden ser personalizados mediante la lógica implementada por el usuario, conectando los transistores a través de vías definidas en dicha lógica. La personalización se realiza durante la fabricación del CI especificando la interconexión entre metales y esto significa que para que un usuario pueda emplear un MPGA, deberá esperar un largo periodo de tiempo en esta instancia, conllevando a que esta fase sea la más costosa en el proceso de fabricación. Aunque los MPGAs no se enmarcan dentro de los FPDs, son mencionados aquí, porque motivaron el diseño de dispositivos equivalentes, programables por el usuario: Field Programmable Gate Arrays (FPGA).

Tanto los MPGAs como las FPGAs, están compuestos por un arreglo de elementos de circuito independiente, llamada lógica de bloques y por la interconexión de recursos. Pero la configuración de la FPGA es realizada a través de programación por el usuario final.

Como el único tipo de FPD que soporta muy alta capacidad lógica, las FPGAs han sido responsables del mayor avance en el diseño de circuitos digitales [Brown, 1996].

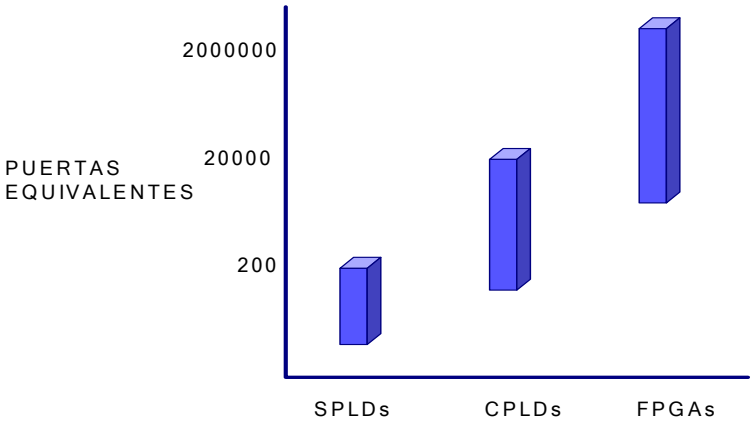

Figura 4.1 *FPDs por Capacidad Lógica.*

Las FPGAs utilizan el concepto de Complex Logic Block (CLB); este es configurable y además permite no sólo realizar el ruteado sobre el dispositivo, sino también cada bloque lógico puede ser configurado independientemente de manera óptima. La arquitectura de una FPGA consiste en arreglos de Celdas Lógicas, las cuales se comunican unas con otras mediante canales de conexiones verticales y horizontales.

Cada celda lógica es funcionalmente similar a los bloques lógicos de un CPLD. La diferencia radica en que la FPGA normalmente utiliza generadores de funciones (LUT) en lugar de compuertas lógicas (*Fig 4.2)*. Cada uno de estos generadores es como una memoria, en donde en vez de implementar la función lógica mediante compuertas, se precalcula el resultado y se almacena en el generador. Las entradas al generador funcionan como un bus de direcciones, y mediante las diferentes combinaciones de las entradas al generador se selecciona el resultado correcto [ZeidmanI, 1999]. Esto le da una gran densidad al dispositivo ya que se maneja un gran número de generadores, pero el tiempo de propagación al implementar una función lógica en estos generadores es menor al que se necesitaría si se utilizaran compuertas.

En general una celda lógica tiene menos funcionalidad que la combinación de sumas de productos y macroceldas de un CPLD, pero como cada FPGA tiene una gran cantidad de celdas lógicas es posible implementar grandes funciones utilizando varias celdas lógicas en cascada.

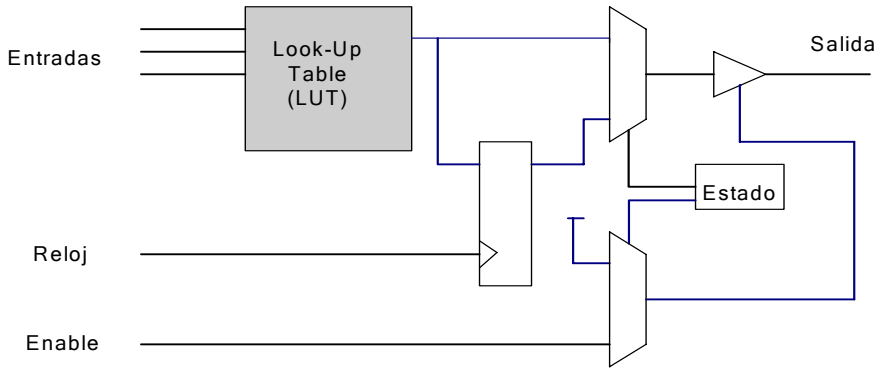

Figura 4.2 *Bloques Lógicos de una FPGA.*

4.2 FPGAs DISPONIBLES COMERCIALMENTE

Como uno de los principales sectores de crecimiento de la industria de semiconductores, el mercado de las FPGAs resulta ser muy volátil. Continuamente el grupo de empresas que participan en el desarrollo de este campo, cambia rápidamente sus objetivos de producción y desarrollo, haciendo difícil predecir que productos serán los abanderados, cuando el desarrollo tecnológico impuesto alcance su estado estable. Por esta razón, es difícil nombrar a todos los fabricantes presentes en el mercado, pues no todos apuntan a obtener un desarrollo generalizado que se imponga comercialmente; tan sólo se hará referencia a los fabricantes que aporten equipamiento de amplio impacto comercial en el mercado.

Actualmente existe una cantidad considerable de fabricantes de tecnología FPGA, aunque una gran tajada del mercado es dominada por dos compañías, Xilinx y Altera (*Fig. 4.3)*, que han sido los dominantes en este sector históricamente desde los inicios de esta tecnología. Los otros fabricantes presentes son: Atmel, Lattice, Actel, Cypress, Quicklogic, Chip Express, DynaChip, AT&T, Gatefield, HammerCores, Lucent, Motorola, QuickTurn, Vantis [Brown, 1996].

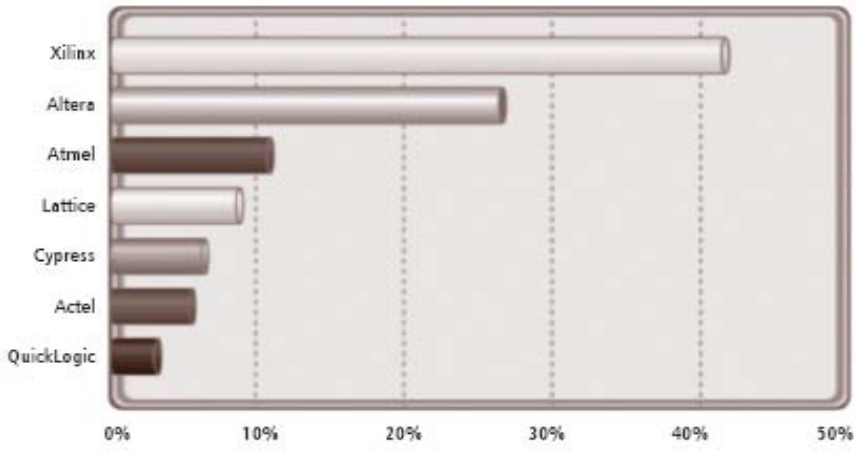

Figura 4.3 *Fabricantes de FPGAs.*

4.3 XILINX SPARTAN 3

Como ya se ha mencionado, una FPGA es un dispositivo programable, cuya estructura está formada por un arreglo de bloques lógicos que interactúan entre sí. La FPGA dispone de diversos componentes lógicos (CLB - Slice), ensamblados a partir de elementos de electrónica básica como flip-flops, buffers, multiplexores que son utilizados para conformar una compleja plataforma de desarrollo programable.

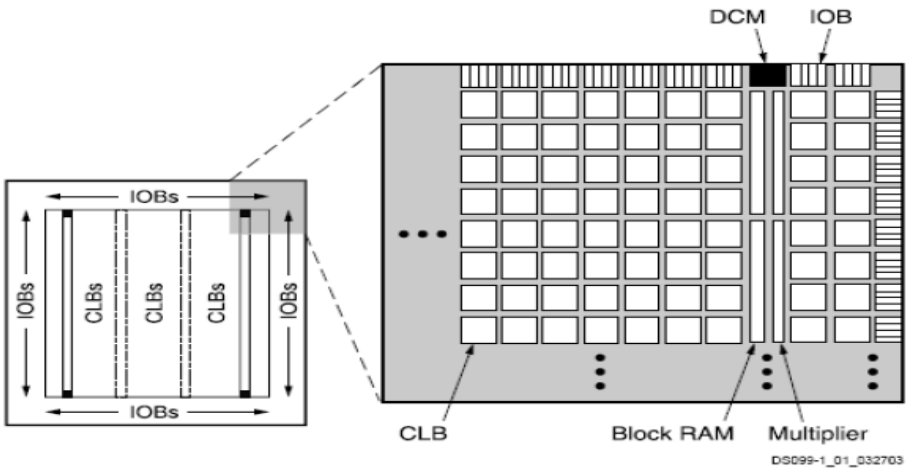

Figura 4.4 *Estructura de Bloques Lógicos de la Spartan 3.*

La arquitectura de la Spartan 3 está compuesta por cinco elementos funcionales, que a su vez pueden ser programados de manera independiente dados los requerimientos del diseñador. Estos elementos son: Bloque lógico configurable, bloque entrada/salida, bloque RAM, bloque de multiplicación y manejador digital de reloj.

4.3.1 Bloque Lógico Configurable (CLB)

Los CLB o bloques lógicos, constituyen la unidad básica de una FPGA. Un bloque lógico ha de ser capaz de entregar una función lógica y reprogramable y para ello utiliza tablas de valores preasignados o tablas look–up (LUT). Una tabla de look-up básicamente es una memoria con un circuito de control que se encargará de cargar los datos preasignados. Cuando se aplica una dirección de entrada a la tabla, esta devuelve a la salida el valor requerido; de esta manera la resolución obtenida dependerá del numero de datos preasignados en la tabla.

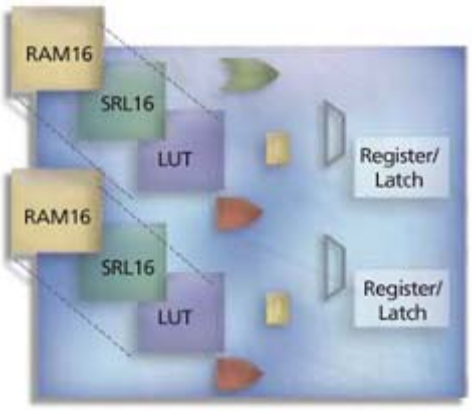

Figura 4.5 *Estructura de un Bloque Lógico.*

4.3.2 Bloques Entrada/Salida (IOB)

Controlan el flujo de datos entre los pines de entrada/salida y la lógica interna alojada en el dispositivo. Cada IOB soporta un flujo de carácter bidireccional y en el caso de la Spartan 3, existe la posibilidad de manejar 24 formatos estándar de comunicación.

4.3.3 Bloque RAM

Los bloques RAM constituyen células de memoria en las cuales serán alojados datos del programa de configuración que se esté ejecutando. Físicamente el bloque RAM tiene dos puertos (A y B) de acceso independiente, cuya estructura es completamente simétrica haciendo posible utilizar los puertos A y B de manera intercambiable ya sea para operaciones de lectura o escritura.

Las unidades RAM están organizadas por columnas, que a su vez están formadas por bloques RAM de 18 kbit, a los cuales le es asignado un multiplicador dedicado. Para obtener una organización más compleja sin comprometer el rendimiento por limitaciones de tiempo, es necesario utilizar aplicaciones software de enrutamiento (routing) en el proceso de síntesis del proyecto.

4.3.4 Bloques de Multiplicación

Los bloques de multiplicación aceptan dos números binarios de 18 bits como entrada y a su salida se obtiene como resultado el producto de éstos. Es posible encontrar situaciones en las que se implementan multiplicadores dedicados, esto con el fin de optimizar funciones lógicas en las que sea necesario trabajar con estructuras en suma de productos.

4.3.5 Manejador Digital de Reloj (DCM)

Un bloque de manejo digital de reloj proporciona calibración interna y además soluciones enteramente digitales para distribución, retardo, multiplicación, división y cambio de fase de las señales de reloj. En la arquitectura estándar suelen alojarse dos bloques DCM en la parte superior y dos en la parte inferior de la FPGA, aunque en arquitecturas más complejas como la utilizada en la Spartan 3 pueden encontrarse bloques DCM en los laterales, todos conectados por medio de una basta red que transmite señales entre ellos y los demás bloques funcionales[1].

4.4 MICROPROCESADORES SOFT – CORE

Un procesador soft-core es la implementación mediante lenguaje de descripción de hardware (HDL) de un microprocesador, ensamblado mediante bloques de lógica básica sobre una FPGA o algún PLD similar [Marschner, 2007]. El principal campo de aplicación de estos dispositivos se ha centrado en el desarrollo de ASIC (circuito integrado de aplicación específica), teniendo en cuenta el ahorro tanto en espacio como en consumo de potencia que supone su utilización.

Una de las características más relevantes de los procesadores soft-core es su flexibilidad, pues permite al diseñador integrar sobre una plataforma configurable, un número determinado de procesadores dispuestos conforme a los requerimientos del sistema.

[1] Spartan-3 FPGA Family: Complete Data Sheet. www.xilinx.com

Comercialmente existen varios procesadores tipo soft-core, pero dada la posición favorable que tienen Xilinx y Altera dentro del mercado de los dispositivos programables se ha masificado la implementación de sus propias IP Core (Intellectual Property). En el caso de Xilinx ofrece los procesadores MicroBlaze y PicoBlaze incluidos en la plataforma de desarrollo Xilinx EDK, mientras que altera proporciona el procesador Nios II en su propia plataforma de desarrollo para sistemas embebidos. Cabe resaltar que estas Cores no son de código abierto y el desarrollo de aplicaciones se encuentra limitado por los parámetros del fabricante.

Existen versiones de procesadores soft-core implementadas en código abierto como el LatticeMicro32, LatticeMicro8 y versiones clonadas de los procesadores de Xilinx denominadas PacoBlaze y OpenFire. Los procesadores de código abierto poseen una gran ventaja y es permitir al diseñador generar e implementar su propio paquete de instrucciones, esto con el fin de optimizar el rendimiento del sistema en general.

4.5 MICROBLAZE

El MicroBlaze es un procesador soft-core creado por Xilinx para la implementación sobre sus series de FPGAs Virtex y Spartan. La arquitectura del MicroBlaze es del tipo Harvard de 32 bits y utiliza una estructura RISC (Reduced Instruction Set Computer) debido a las restricciones de espacio y a la simplicidad requerida para ser implementado sobre una FPGA (*Fig. 4.6)*. Si se comparan las prestaciones del MicroBlaze con las de cualquier otro microprocesador comercial, ciertamente resultarán inferiores pues el objetivo de un procesador soft-core es obtener un sistema compacto ajustado a los requerimientos básicos del sistema.

De esta manera el MicroBlaze posee 87 instrucciones soportadas, si se consideran diferentes las instrucciones que operan con valores inmediatos de aquellas que realizan la misma operación con registros. Adicionalmente, cada instrucción se ha elegido para que el tamaño de la ALU también sea reducido. Las instrucciones que requieran un procesamiento complejo habrán de realizarse en un hardware específico, diseñado utilizando los restantes recursos de la FPGA[2].

[2] MicroBlaze Processor Reference Guide. www.xilinx.com

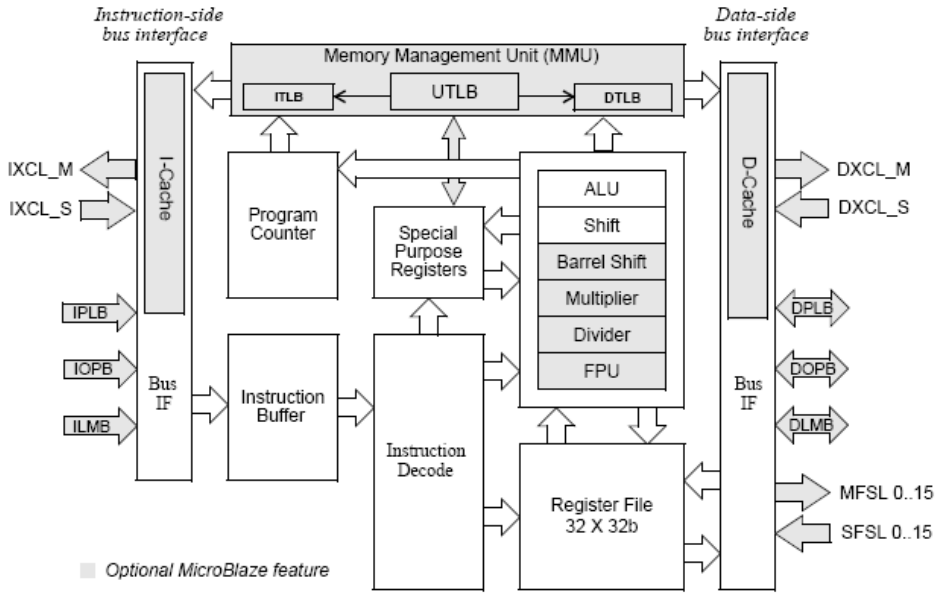

Figura 4.6 *Arquitectura del MicroBlaze.*

Un aspecto muy importante a tener en cuenta es la disposición de los buses de comunicación implementados en el MicroBlaze. Xilinx utiliza el estándar CoreConnect creado por IBM, para conectar los diferentes bloques que conforman el sistema, pues como ya se ha mencionado el MicroBlaze es una IP core que necesita interactuar con los demás bloques lógicos alojados en la FPGA. El estándar define tres tipos de buses, cada uno con unas características de velocidad y conectividad:

LMB (Local Memory Bus): Bus síncrono de alta velocidad, utilizado para conectar periféricos y los bloques de memoria interna de la FPGA. Solo admite un maestro en su implementación para MicroBlaze y puede ser utilizado tanto para instrucciones como para datos. Este bus es compatible con el PLB (Processor Local Bus) incluido en el estándar CoreConnect, pero a diferencia de éste, el LMB no admite varios maestros ni tamaños de palabra diferentes a 32 bits.

OPB (On-chip Peripheral Bus): Bus síncrono utilizado para conectar periféricos con tiempos de acceso variables. Soporta varios maestros y la conectividad de muchos periféricos es sencilla gracias a su identificación por multiplexación distribuida. Soporta transferencias de tamaño de palabra dinámico.

DCR (Device Control Register): Bus síncrono diseñado para una conexión tipo anillo de periféricos con ancho de banda muy bajo. Solo soporta un maestro y está diseñado para minimizar el uso de lógica interna.

Además de estos tres buses existe otra alternativa para conectar el MicroBlaze con los demás periféricos. Se trata del protocolo FSL (Fast Simple Link), un bus de comunicaciones unidireccional con disposición maestro–esclavo punto a punto y con gestión de datos FIFO. El MicroBlaze dispone de 8 canales FSL pareados, es decir, 8 en condición maestro y 8 en esclavo.

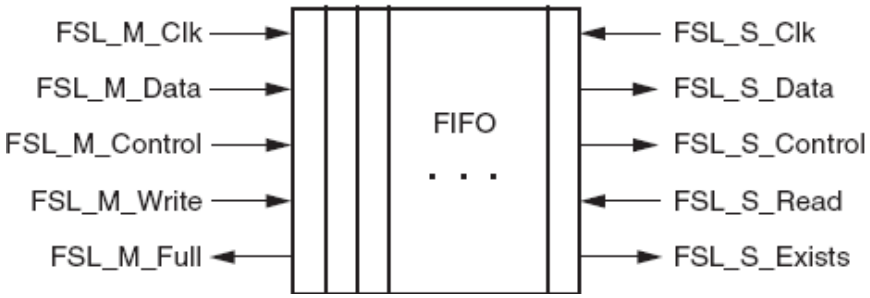

Figura 4.7 *Bus de Comunicaciones FSL.*

Dentro de las interfaces de comunicación descritas, el bus FSL es quién posee la velocidad de operación más alta (800 MB/s), y por esta razón se ha utilizado en el desarrollo de este proyecto, para conectar las IP cores diseñadas con el MicroBlaze. La configuración de esta interfaz será descrita en los próximos apartados.

5 DISEÑO DEL MODELO DE HARDWARE

Los lenguajes de descripción de hardware (VHDL, Verilog, ABEL) permiten diseñar circuitos digitales para su posterior implementación sobre dispositivos programables como CPLDs y FPGAs. De esta manera la descripción del circuito digital se hace desde una interfaz de programación y posteriormente se implanta a nivel de compuertas lógicas sobre la plataforma programable en hardware.

El proceso de diseño de un bloque de hardware para implementación sobre FPGA consta de tres etapas fundamentales: Diseño del Modelo, Síntesis e Implementación (*Fig.5.1)*. Existe una cuarta etapa que es la simulación del modelo, pero esta no es indispensable para la construcción del bloque hardware, así que no se tendrá en cuenta como parte de este documento.

Cada uno de los fabricantes de FPGAs posee su propio entorno de programación en lenguaje VHDL, en este caso dada la utilización de dispositivos Xilinx, se profundizará en el entorno de desarrollo Xilinx ISE 9.2i y en las herramientas que este posee.

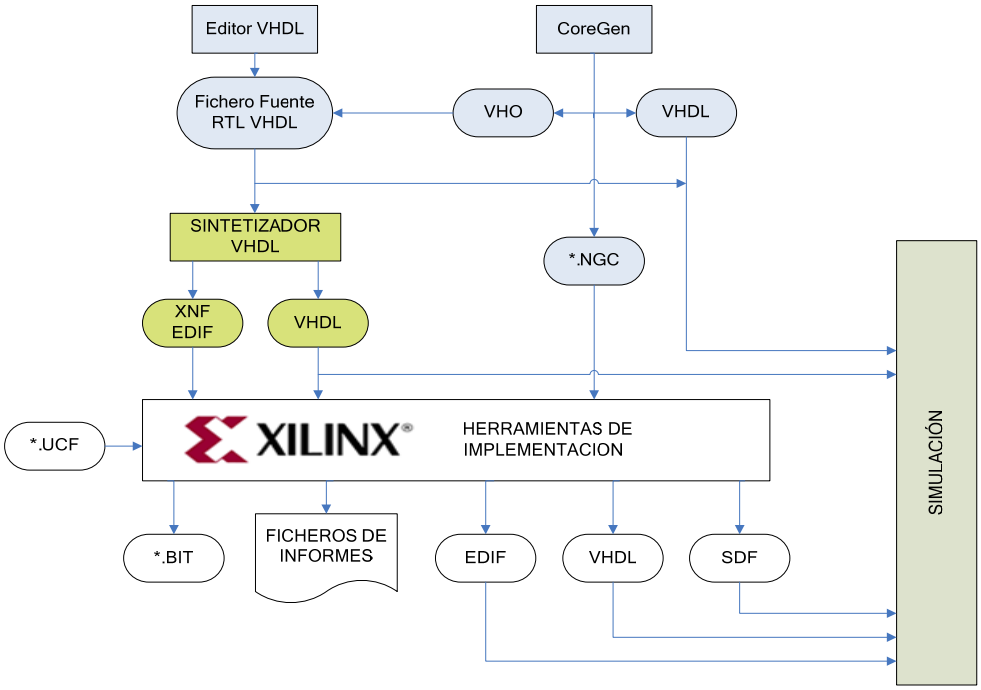

Figura 5.1 *Flujo de Diseño para FPGA Xilinx.*

5.1 DISEÑO DEL MODELO

El diseño del modelo es la primera fase en el proceso de construcción de un nuevo componente que será implementado en una FPGA. Diseñar un modelo implica traducir al lenguaje VHDL, el comportamiento y los distintos estados de operación del dispositivo que se quiere implementar.

Para editar un modelo en lenguaje VHDL es posible utilizar cualquier editor de texto, pero resulta mucho más práctico utilizar editores especializados en el lenguaje VHDL pues aportan una interfaz de programación adecuada para la labor del programador.

Actualmente existen herramientas de programación que permiten elaborar códigos VHDL a partir de esquemas prediseñados, que en gran medida facilitan la generación de código sintetizable para FPGA. Es de resaltar que en este proyecto se han utilizado dos herramientas de código prediseñado (CoreGenerator y Simulink HDL Coder), con el fin de analizar las ventajas e inconvenientes que presentan a la hora de generar código sintetizable.

5.2 ENTORNO ISE DE XILINX

Xilinx proporciona el entorno de programación ISE, en el cual es posible realizar un diseño completo basado en lógica programable, es decir, el ISE incluye todas las herramientas necesarias para construir un bloque de hardware, desde el diseño del modelo hasta la implementación directa sobre la FPGA. En este apartado se tendrán en cuenta sólo las herramientas de edición de código sintetizable que se incluyen en el ISE.

De acuerdo a las necesidades del diseñador, el ISE proporciona tres diferentes alternativas para generar código sintetizable, estas son, edición de código a través de lenguajes de descripción de hardware como VHDL, Verilog o ABEL, captura de bloques esquemáticos o representación gráfica de diagramas de estado.

El proceso para crear un nuevo proyecto desde la aplicación Project Navigator, es equivalente para cada una de las aplicaciones mencionadas, puesto que el ISE cuenta con un asistente de configuración que va guiando al usuario en el proceso de configuración del nuevo proyecto (*Fig. 5.2)*. La diferencia radica en el tipo de fuente o entrada de código que se le define al asistente pues es allí donde el diseñador elige la herramienta que utilizará en la edición de su aplicación.

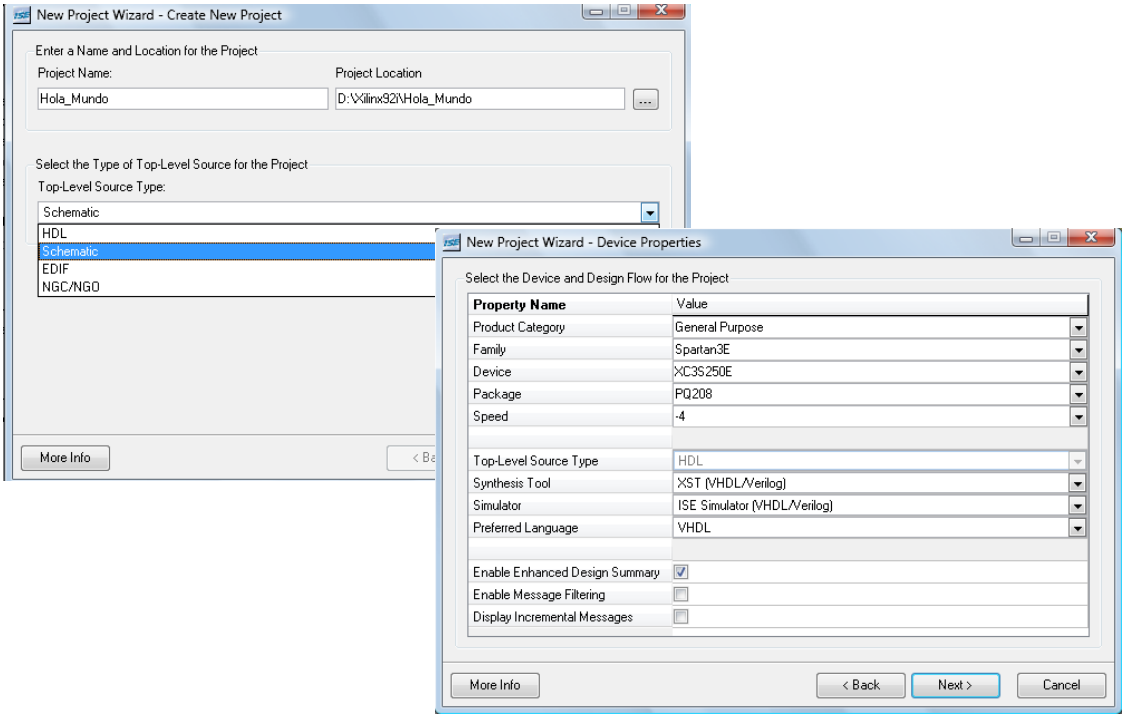

Figura 5.2 *Asistente para Creación de un Nuevo Proyecto.*

5.3 EDICIÓN DE CÓDIGO VHDL MEDIANTE EL ISE

El entorno de desarrollo ISE de Xilinx posee un aspecto parecido al de los entornos de programación convencionales como puede ser C++ o Visual Basic, y emplea una distribución de ventanas, en las que están alojadas las distintas herramientas tanto de control como de visualización del proyecto desarrollado.

Un módulo descrito en lenguaje VHDL consta de dos unidades básicas de diseño, la entidad y la arquitectura. En la entidad se definen tanto las entradas como las salidas que tendrá el dispositivo, mientras que en la arquitectura se define el comportamiento como tal del dispositivo.

Si se toma como referencia un caso sencillo en el que se quiere implementar un sumador de dos dígitos, en la entidad se definirían las entradas o sumandos y además la salida, que corresponderá al resultado de la operación realizada.

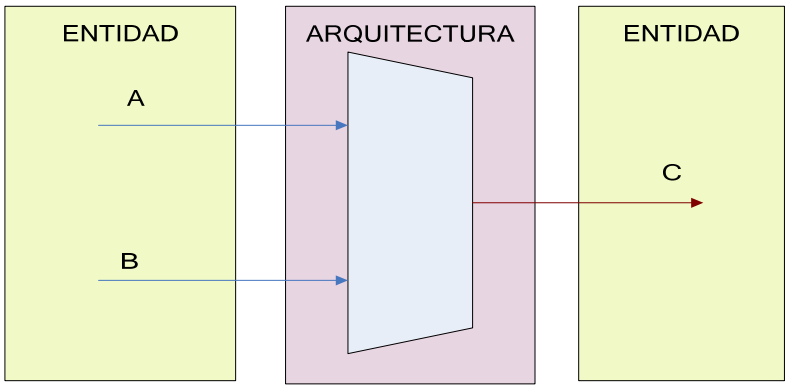

Figura 5.3 *Estructura de un Componente en VHDL.*

Las definiciones de los puertos de comunicaciones (entradas y salidas) en la entidad del modelo creado, servirán para conectar el dispositivo con los demás componentes con que cuente el proyecto. Normalmente en el desarrollo de un proyecto se utiliza una estructura jerárquica que facilita el acople modular de todo el proyecto. De esta manera por medio de los parámetros de identificación definidos en la entidad es posible invocar el módulo específico, en este caso el sumador, y asignarle las conexiones correspondientes para que funcione de manera adecuada.

El nombre que se le asigna a la entidad, servirá para identificar el componente creado e invocarlo cuando sea necesario como se verá más adelante.

```
entity Sumador is
port ( A: in std_logic_vector  (3 downto 0);
       B: in std_logic_vector  (3 downto 0);
       C: out std_logic_vector (4 downto 0));
end Sumador;
```

Como ya se había mencionado anteriormente, en la arquitectura se define el comportamiento o el proceso en general que llevará a cabo el dispositivo. De esta manera es en la arquitectura en donde se traduce al lenguaje de programación, las características que hemos previsto para el diseño.

La arquitectura está ligada a la entidad, pues todas las funciones descritas en la arquitectura, harán referencia a los puertos especificados en la entidad.

```
architecture Behavioral of Sumador is


begin
    process (A,B)
            begin
                C <= A + B;
            end process;
end Behavioral;
```

A su vez en la arquitectura es posible invocar componentes que hayan sido especificados en una jerarquía menor con respecto a la que se esté trabajando. Es así como se logra establecer relación entre los diferentes bloques de un proyecto [Ashenden, 1990]. Este proceso se define como Instanciamiento y es necesario realizarlo cada vez que se invoque a un módulo ya elaborado y que se quiera reutilizar en el proyecto actual.

5.3.1 INSTANCIAMIENTO Y MAPEADO DE UN COMPONENTE

La reutilización de bloques hardware es una de las principales ventajas de utilizar el lenguaje VHDL. De esta manera es posible utilizar desarrollos hechos con anterioridad y estructurar un nuevo proyecto de manera modular.

El instanciamiento consiste en el llamado de una entidad o componente ya existente, dentro de una nueva arquitectura. Cabe resaltar que el instanciamiento se debe realizar sobre la arquitectura de un componente de mayor jerarquía
Además de instanciar el componente será necesario realizar el mapeado de los puertos del mismo, para y así conectar las entradas y salidas de este, con los diferentes elementos descritos dentro de la arquitectura del nuevo dispositivo.

Es necesario también definir señales o vías de comunicación que conectarán todos los puntos internos del dispositivo, pues a estos no se tendrá acceso directo desde el exterior.

Un componente puede ser reutilizado dentro de una arquitectura cuantas veces se desee, pero es necesario especificar un nombre diferente para cada llamado que se haga, pues el sistema los identificará como elementos distintos así realicen la misma función.

Para comprender mejor el proceso de instanciamiento y mapeo de un componente, a continuación se presenta un ejemplo en el cual se reutilizará el módulo de suma diseñado en el apartado anterior. El nuevo componente tendrá dos módulos de suma, cuyas salidas se conectarán a un multiplexor que seleccionará cual de ellas irá a la salida general del nuevo componente (*Fig. 5.4)*.

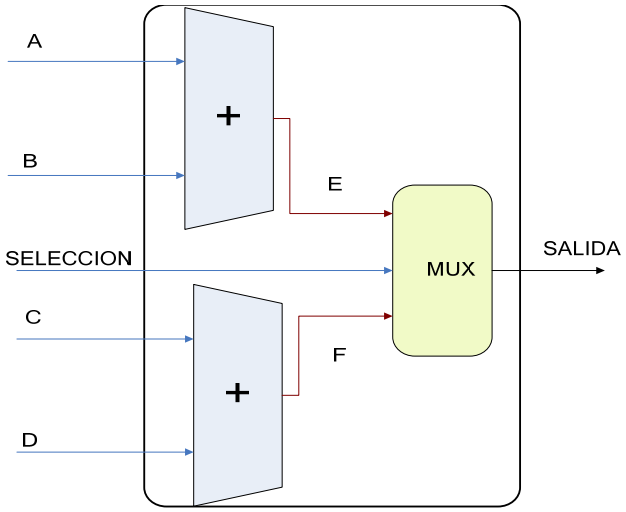

Figura 5.4 *Instanciamiento y Mapeo de un Componente.*

El primer paso para crear un proyecto a partir de módulos prediseñados, es generar un módulo VHDL principal o módulo "top" que será el módulo de mayor jerarquía en el proyecto.

El nombre que se le asigne al módulo principal, será el identificador del nuevo componente, pues a su vez este módulo podrá ser reutilizado por una entidad de jerarquía mayor.

Para definir las características del módulo principal es necesario seguir todos los pasos descritos anteriormente, es decir, definir la entidad, la arquitectura y las señales que se manejarán dentro del módulo.

En la entidad se definen las entradas y salidas del módulo principal. Los nombres asignados a cada uno de los puertos serán los identificadores que permitan relacionar el nuevo componente con etapas de diseño de mayor jerarquía. Del mismo modo, los parámetros provenientes de jerarquías menores no aparecerán en la entidad, pues éstas relaciones se efectúan directamente en la arquitectura.

```
entity modulo_top is
port (              A: in std_logic_vector (3 downto 0);
                    B: in std_logic_vector (3 downto 0);
                    C: in std_logic_vector (3 downto 0);
                    D: in std_logic_vector (3 downto 0);
                    Seleccion: in  bit;
                    Salida: out std_logic_vector (4 downto 0));

end modulo_top;
```

En la arquitectura se realiza el instanciamiento y posteriormente el mapeado de todos los componentes que se utilizarán. A su vez es necesario definir las señales que operarán en el sistema para conectar los bloques internamente:

```
architecture Behavioral of modulo_top is

signal E,F       : std_logic_vector (4 downto 0); -- Definición de Señales

component Sumador                                 -- Definición de Componentes

port (                  A: in std_logic_vector (3 downto 0);
                        B: in std_logic_vector (3 downto 0);
                        C: out std_logic_vector (4 downto 0));
end component;

component mux

port (
                    Seleccion       : in  bit;
                    CHA             : in  std_logic_vector (4 downto 0);
                    CHB             : in  std_logic_vector (4 downto 0);
                    Salida          : out std_logic_vector (4 downto 0));
end component;
```

Después de la definición de la arquitectura es necesario mapear los parámetros definidos para cada componente. Es decir, se efectuará la relación entre los parámetros definidos en la entidad del módulo principal, con los parámetros de las identidades de cada uno de los componentes empleados.

En el caso que uno de los parámetros no se pueda relacionar directamente debido a la estructura propia del dispositivo, será necesario crear señales para conectar los módulos. Las señales serán definidas antes de realizar el mapeado de puertos, y deberán tener las mismas especificaciones de los puertos que van a conectar. Como ya se había mencionado, las señales no tendrán representación externa, y si en algún caso fuese necesario ingresar o ver sus datos desde el exterior, será necesario definirlas como puertos [Ashenden, 1990].

```
begin

U1: Sumador
PORT MAP (
        A               => A,
        B               => B,
        C               => E
                                );
U2: Sumador
PORT MAP (
        A               => C,
        B               => D,
        C               => F
                                );
U3: mux
PORT MAP (
        CHA             => E,
        CHB             => F,
        Seleccion       => Seleccion,
        Salida          => Salida
                                );
end Behavioral;
```

5.4 EDICIÓN MEDIANTE CAPTURA DE BLOQUES ESQUEMÁTICOS (ECS)

Además de la programación mediante lenguaje de descripción de hardware, el entorno de programación ISE de Xilinx, permite utilizar herramientas gráficas para editar el comportamiento del dispositivo diseñado. De esta manera el programador no necesitará conocer a profundidad la sintaxis de un lenguaje de programación, tan sólo se emplearán conocimientos de electrónica digital.

La programación gráfica permite al usuario tener una representación visual de su propio diseño, de esta manera es más fácil para el usuario materializar sus ideas, sin tener que reflexionar en primera instancia, cómo plasmar su pensamiento en un lenguaje de programación.

La aplicación para captura de esquemáticos Xilinx ECS, posee una amplia librería de componentes prediseñados, con los cuales es posible construir la estructura de nuevos componentes, teniendo siempre como base dispositivos electrónicos básicos como puertas lógicas o flip-flops.

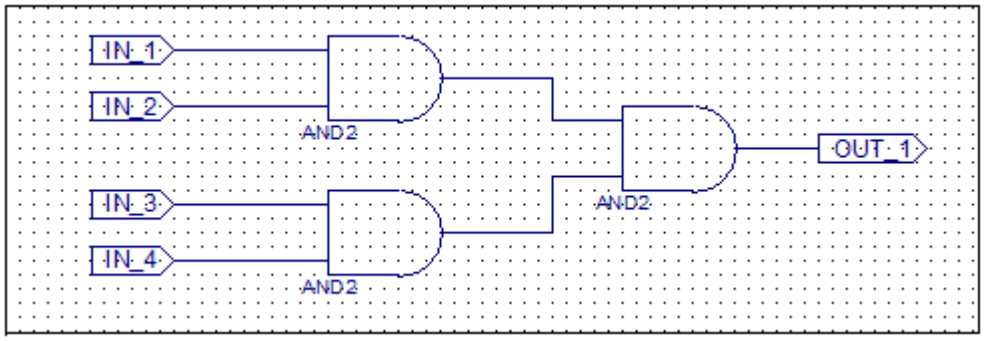

Figura 5.5 *Captura de Esquemáticos Mediante el ISE.*

En el caso de la captura de esquemáticos, no es necesario realizar un instanciamiento como tal de los dispositivos de menor jerarquía. En este caso se crearán nuevos Símbolos o bloques funcionales, que serán accesibles desde aplicaciones de jerarquía mayor.

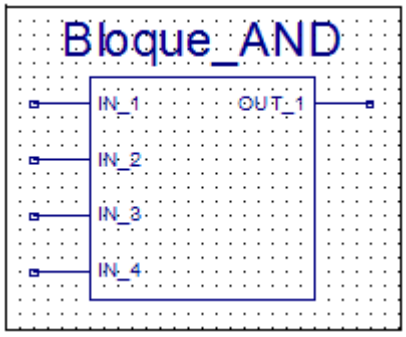

Figura 5.6 *Bloque Funcional de Mayor Jerarquía.*

De igual forma a partir de código escrito en lenguaje de descripción de hardware, es posible generar bloques simbólicos para utilizarlos sobre el editor de esquemas, de esta manera el diseñador obtiene dos formas complementarias para realizar el desarrollo de código sintetizable para FPGA.

A continuación se presenta un bloque simbólico generado a partir del código VHDL implementado en el sumador multiplexado descrito anteriormente en este trabajo:

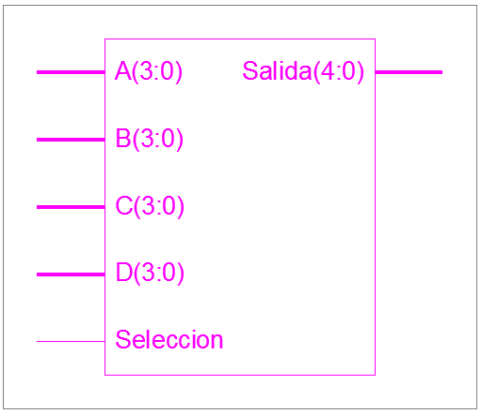

Figura 5.7 *Bloque Simbólico Generado por VHDL.*

El código gráfico generado en Xilinx ECS, es igualmente sintetizable como las aplicaciones desarrolladas en lenguaje de descripción de hardware.

5.5 REPRESENTACIÓN GRÁFICA DE DIAGRAMAS DE ESTADO

Otra de las aplicaciones para edición de código sintetizable mediante herramientas gráficas es el StateCAD o Graficador de Diagramas de Estado. El StateCAD es una aplicación incluida en el ISE, a la cual se puede acceder de manera independiente, o simplemente desde un proyecto ya definido, agregando una nueva fuente de código, en este caso nombrada como State Diagram.

El StateCAD permite crear mediante una interfaz gráfica la representación de diagramas de estado (*Fig. 5.8)*. Es mucho más sencillo e intuitivo para el diseñador, observar la secuencia lógica de los estados del proceso que está diseñando. Igualmente como ya se había mencionado en la captura de esquemáticos, el usuario no necesitará conocer a fondo la sintaxis de programación, pues tan sólo se requiere definir cada uno se los parámetros específicos para cada estado, así como también las restricciones y flujo de variables presentes en el diseño.

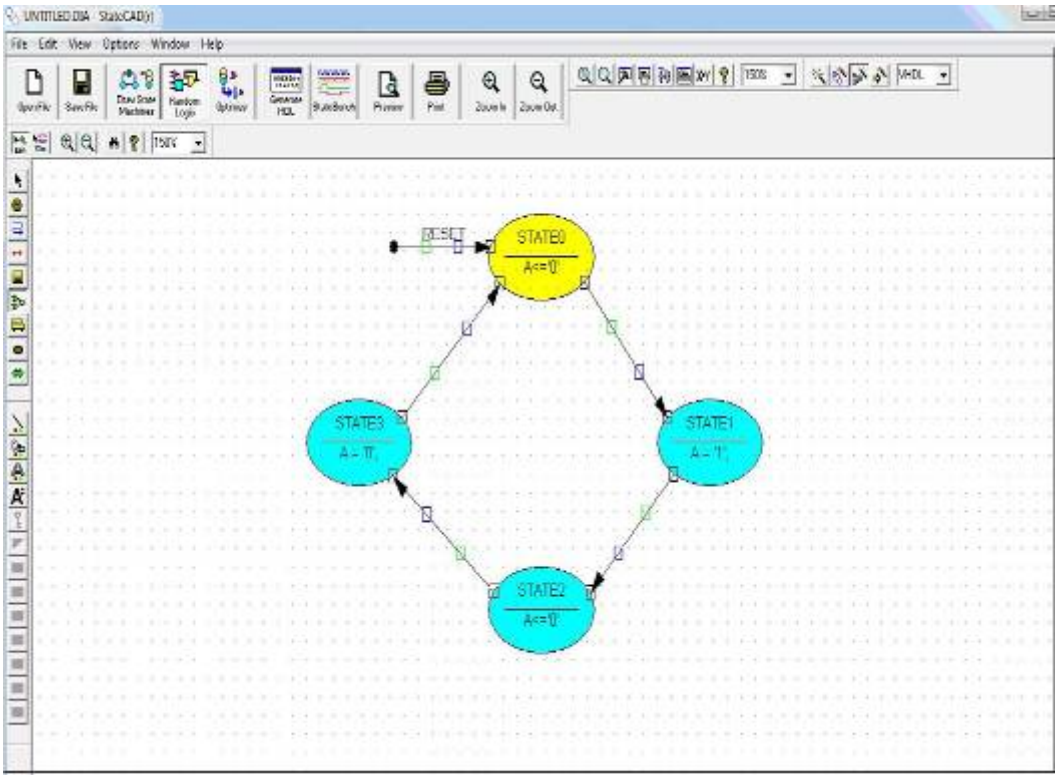

Figura 5.8 *Plataforma de Diseño StateCAD.*

Uno de los principales aportes del StateCAD, es la posibilidad de generar a partir del esquema gráfico, código modificable en lenguaje de descripción de hardware (*Fig. 5.9)*. De esta manera el diagrama de estados generado puede integrarse en cualquier otro proyecto de trabajo, siendo posible agregar mejoras o adaptar el código a las necesidades del usuario[3].

Igualmente la herramienta StateCAD incluye un módulo para realizar la simulación del sistema generado. Es importante comprobar mediante la simulación, que el sistema se comporte como es esperado, antes de generar el código VHDL y migrarlo a otra aplicación, pues resulta mucho más cómoda la detección y corrección de errores dentro de este entorno de desarrollo.

[3] Xilinx ISE 8.1 Design Suite Software Manuals and Help, www.xilinx.com

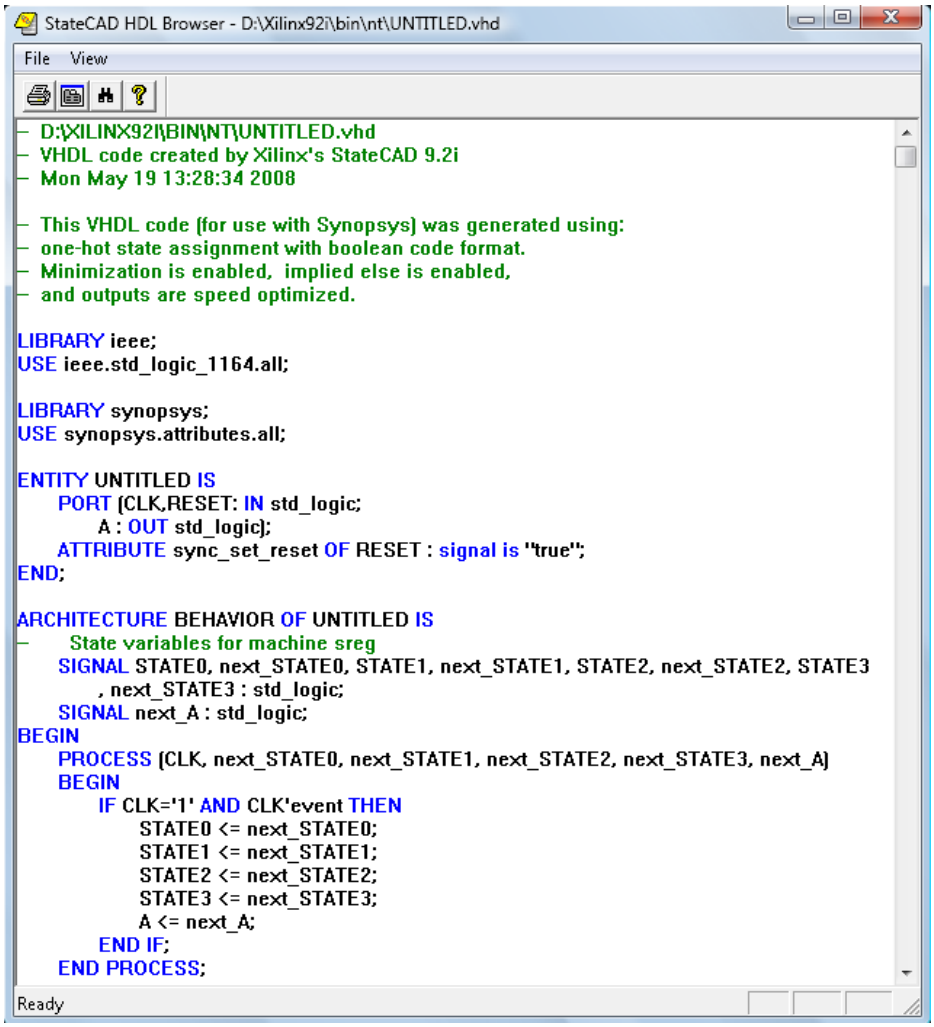

Figura 5.9 *Código VHDL Generado por StateCAD.*

Existe además la posibilidad de utilizar un asistente para optimización del sistema construido, esto con el fin de sacar el mejor provecho al dispositivo en que vaya a implementar este diseño. Dependiendo del dispositivo empleado (FPGA o CPLD), el asistente ajusta la estructura del programa, para que esta sea óptima y brinde el mejor rendimiento.

5.6 HERRAMIENTA DE DISEÑO CORE GENERATOR

El sistema Core Generator (CG) de Xilinx es una herramienta de diseño que ofrece bloques de código prediseñados IP Core, configurables a las necesidades del usuario. CG proporciona desde funciones lógicas básicas hasta complejas estructuras de análisis y procesamiento de señales.

A través del asistente de diseño que posee el CG (*Fig. 5.10)*, se genera un nuevo proyecto, seleccionando los parámetros hardware del dispositivo en que será implementada la nueva Core.

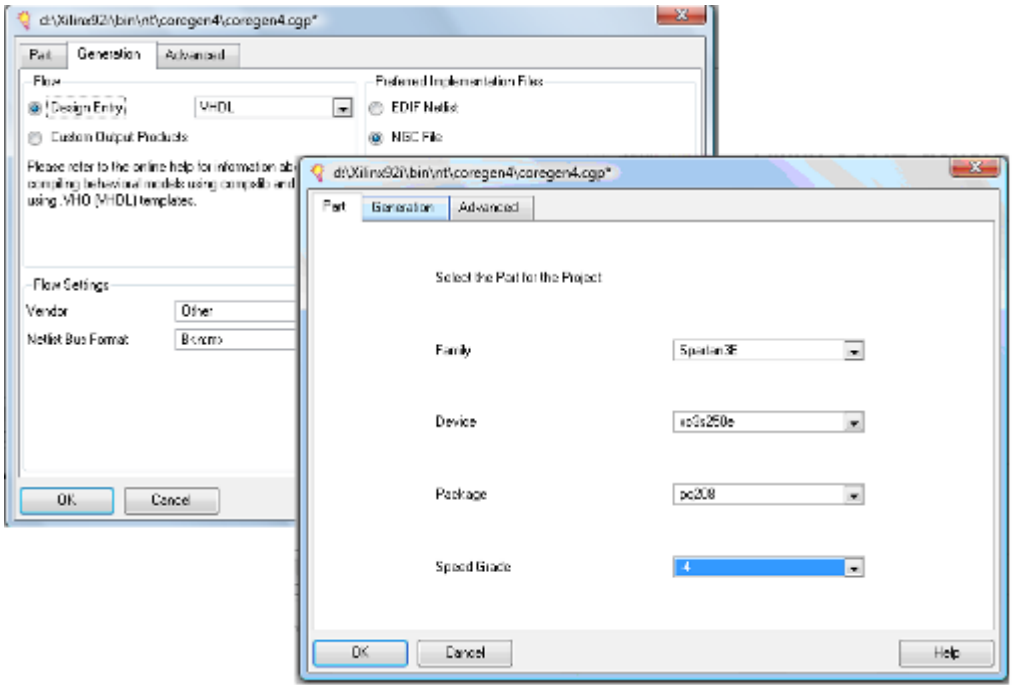

Figura 5.10 *Asistente de Diseño del Core Generator.*

Todas las IP Core disponibles en CG permiten ajustar sus parámetros de funcionamiento con el fin de brindar mayor versatilidad al diseñador. Pero es de resaltar que el código generado está restringido y no se puede acceder ni modificarse por el usuario. Tan sólo se generan los ficheros que describen como se debe efectuar el instanciamiento y mapeo de la IP Core desde una entidad de mayor jerarquía.

La implementación de funciones mediante las IP Core que proporciona el CG, facilita las labores de diseño de nuevos bloques funcionales. Un caso particular en que se refleja esta situación es la implementación de funciones trigonométricas, que dada su complejidad de cálculo, es necesario programarlas directamente en hardware y así no afectar el rendimiento del sistema en general: Si el usuario tuviese que implementar toda la estructura de programación para representar una función trigonométrica, necesitaría calcular mediante algún método iterativo o tablas, los valores de la función que necesitase implementar.

CG utiliza en sus IP Core, tablas de referencia o Look-Up Tables (LUTs) para almacenar los resultados o valores de respuesta de las funciones implementadas. De esta manera el usuario tendrá tan solo que definir la resolución con la que se obtendrán los datos a partir de las tablas.

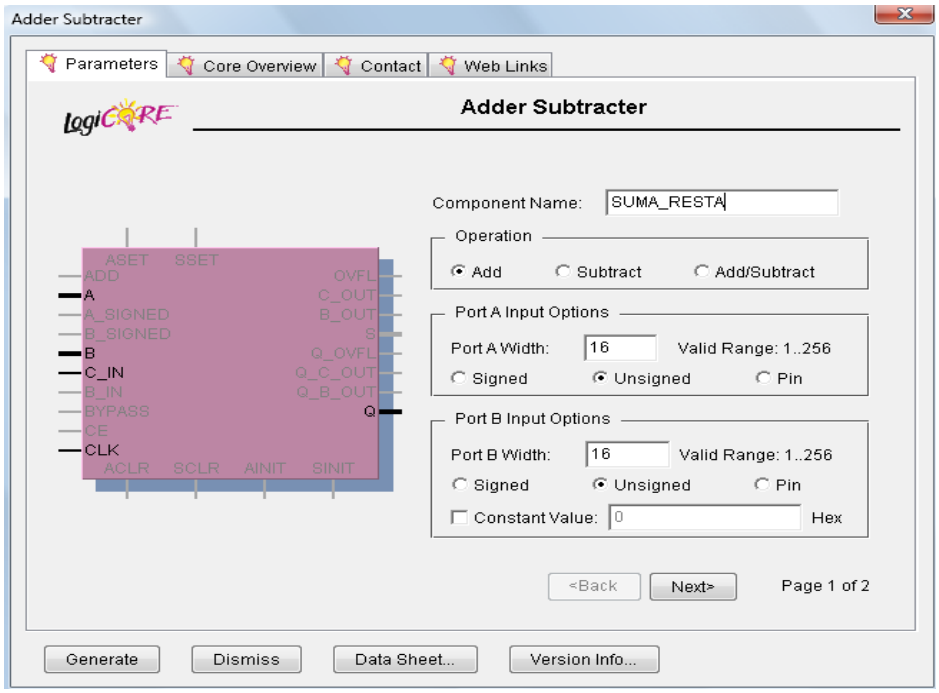

Figura 5.11 *Asistente para Configuración de una IP Core.*

Cada IP Core posee su propio asistente para configuración (*Fig. 5.11)*, en el que el usuario puede definir los parámetros específicos que tendrá el componente generado. El nombre asignado al componente, será al mismo tiempo la identificación para la entidad que podrá ser invocada desde componentes de mayor jerarquía. Es necesario tener esto en cuenta para realizar de manera adecuada el instanciamiento y mapeo del componente.

5.7 PROGRAMACIÓN MEDIANTE SIMULINK HDL CODER

Otra herramienta que permite la obtención de código sintetizable en lenguaje de descripción de hardware, es la Toolbox de Matlab denominada HDL Coder. Esta herramienta permite utilizar el entorno de programación por bloques de Simulink para diseñar todo tipo de circuitos digitales y a su vez obtener el código VHDL o Verilog que describe al componente diseñado[4].

HDL Coder posee una interfaz gráfica (*Fig. 5.12)* en la cual el usuario puede seleccionar los parámetros específicos para la generación del código en lenguaje de descripción de

[4] www.mathworks.com/products/slhdlcoder

hardware (VHDL o Verilog). Es decir, el diseño implementado será totalmente ejecutable en Simulink, pero el HDL Coder permite traducir el diagrama de bloques diseñado a lenguaje de descripción de hardware.

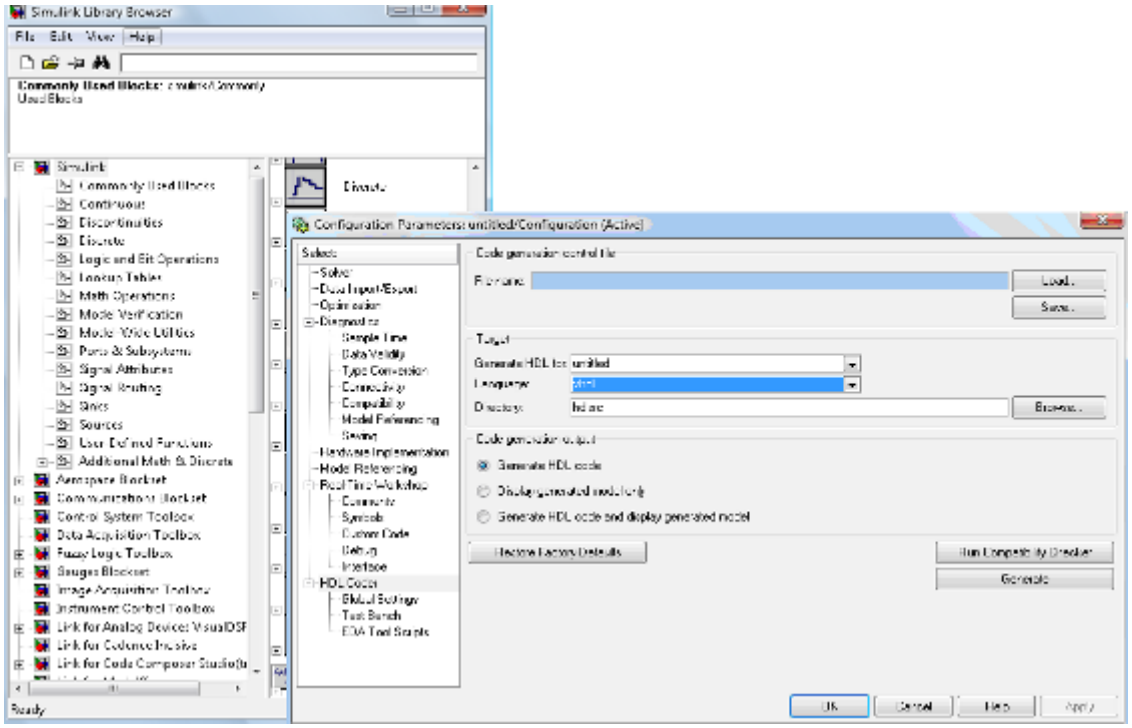

Figura 5.12 *Interfaz Gráfica del HDL Coder.*

Como es sabido, Simulink permite construir y simular mediante bloques todo tipo de sistemas. HDL Coder no tiene la capacidad de traducir a lenguaje de descripción de hardware, todos los bloques disponibles en Simulink. De esta manera es necesario tener en cuenta la disponibilidad de los bloques utilizados, para que la generación del código en VHDL o Verilog sea la esperada.

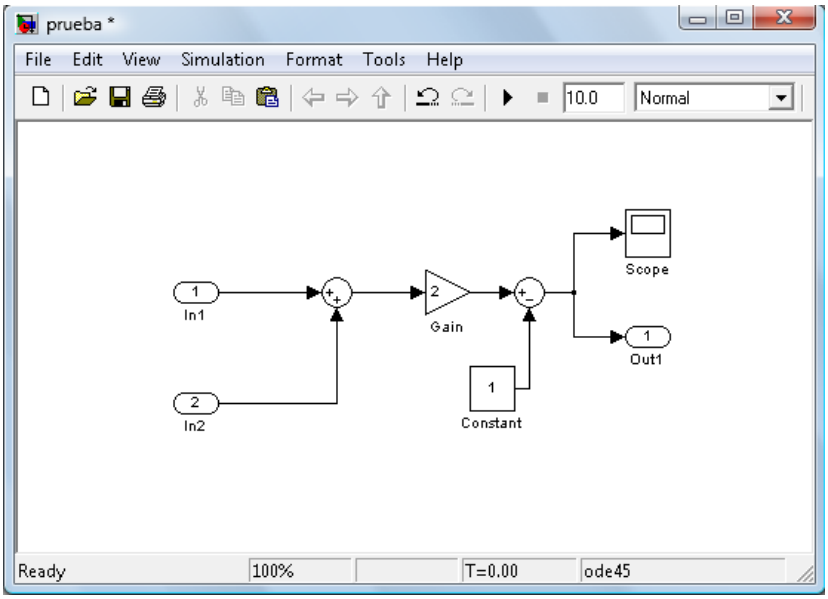

Figura 5.13 *Sistema Diseñado en Simulink.*

Un caso particular en el que los bloques de Simulink no son compatibles para ser traducidos a descripción de hardware, es la utilización de funciones trigonométricas y funciones de interpolación. Por esta razón Simulink incluye bloques de tablas de valores precalculados (LUTs), que posibilitan la implementación de éstas funciones en hardware.

Se debe resaltar que el código VHDL o Verilog obtenido a partir del HDL Coder, no está depurado, y en ocasiones habrá que ajustar su estructura para que funcione correctamente en el dispositivo de destino.

Además de esto, es necesario tener en cuenta las especificaciones del dispositivo en que será implementado el diseño, por ejemplo si posee unidad de coma flotante o no, pues HDL Coder define el formato de las variables utilizadas con parámetros similares a los utilizados en Simulink. Por eso es necesario ajustar dicho formato para que el sistema no consuma más recursos de los que verdaderamente le son necesarios.

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.numeric_std.ALL;
4
5  ENTITY prueba IS
6    PORT( clk                               :   IN    std_logic;
7          reset                             :   IN    std_logic;
8          clk_enable                        :   IN    std_logic;
9          In1                               :   IN    real;  -- double
10         In2                               :   IN    real;  -- double
11         ce_out                            :   OUT   std_logic;
12         Out1                              :   OUT   real  -- double
13         );
14 END prueba;
15
16
17 ARCHITECTURE rtl OF prueba IS
18
19   -- Component Declarations
20   COMPONENT Timing_Controller
21     PORT( clk                             :   IN    std_logic;
22           reset                           :   IN    std_logic;
23           clk_enable                      :   IN    std_logic;
24           enb                             :   OUT   std_logic;
25           enb_1_1_1                       :   OUT   std_logic
26           );
27   END COMPONENT;
28
29   -- Component Configuration Statements
30   FOR ALL : Timing_Controller
31     USE ENTITY work.Timing_Controller(rtl);
32
33   -- Constants
```

Figura 5.14 *Código VHDL Generado por HDL Coder.*

Una de las principales aplicaciones del HDL Coder, es el diseño de sistemas para procesamiento digital de señales. Tanto la programación de DSPs y de FPGAs puede ser implementada mediante bloques de Simulink y posteriormente traducida a un lenguaje hardware que permita su ejecución en dispositivos programables.

5.8 SÍNTESIS DEL MODELO

Una vez validado el diseño del modelo se puede proceder a la síntesis de éste. El ISE posee una herramienta denominada Xilinx Synthesis Technology (XST) la cual realiza el proceso de síntesis de componentes creados en VHDL o Verilog[5]. El proceso de síntesis se puede dividir en tres fases: Identificación de errores, Síntesis y Optimización de bajo nivel como se observa en la (*Fig. 5.15)*.

[5] XST Users Guide. www.xilinx.com

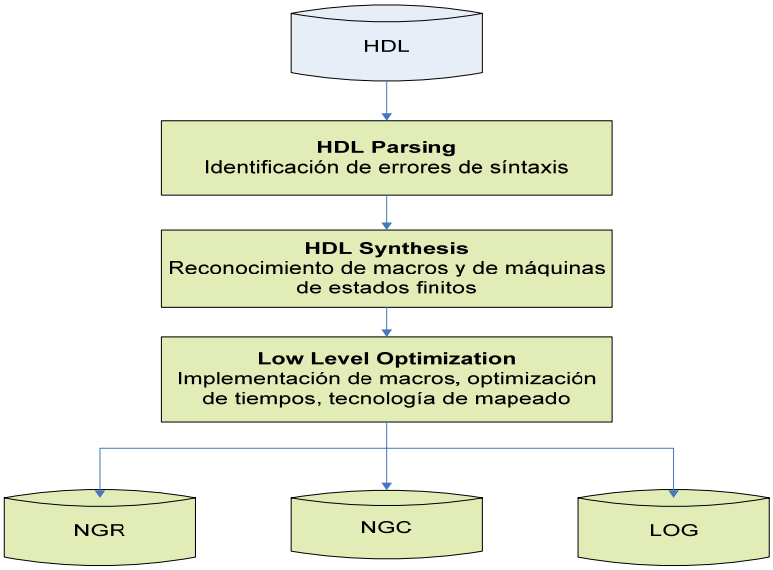

Figura 5.15 *Fases del proceso de Síntesis.*

5.8.1 Identificación de Errores

En la identificación de errores se verifica la sintaxis del código desarrollado por el usuario, teniendo en cuenta las restricciones del lenguaje de descripción de hardware utilizado. Si la verificación de la sintaxis resulta fallida, no se podrá seguir adelante en el proceso de síntesis, y será necesario corregir los errores que se especifiquen en el reporte generado.

5.8.2 Síntesis

Durante la síntesis, XST analiza los códigos HDL e intenta inferir diseños específicos o macros (como MUX, RAM, sumadores) con el fin de mejorar la estructura de programación. El propósito de la síntesis es tratar de simplificar diseños complejos, y redefinirlos por medio de elementos de lógica básica; de esta manera el espacio físico requerido en el dispositivo programable se reduce y además posibilita el uso de frecuencias de operación mucho más altas.

El reconocimiento de máquinas de estados finitos, también hace parte del proceso de síntesis. XST reconoce máquinas de estado independientemente del estilo empleado en su construcción. De esta manera el propósito del XST es encontrar el algoritmo más eficiente para implementar la máquina de estados identificada.

5.8.3 Optimización a Bajo Nivel

Durante la optimización a bajo nivel, XST transforma los macros inferidos y la lógica general a una tecnología de implementación específica. De esta forma se optimizan los macros seleccionados y se reporta al usuario cuales no tuvieron modificaciones.

Como resultado de esta última fase se obtiene una lista de conexiones en formato EDIF (Electronic Design Interchange Format) o XNF (Xilinx Netlist Format), aunque XST generaliza nombrándolos como ficheros ngc, que serán las entradas del proceso de implementación del modelo.

5.9 IMPLEMENTACIÓN DEL MODELO

El primer paso para la implementación del modelo es la definición de las restricciones de usuario (constraints), es decir, se realiza la asignación de pines para cada una de las entradas o salidas especificadas en el código fuente.

Es posible editar el archivo de constraints directamente en texto, agregando al proyecto un archivo fuente de extensión ucf. De esta forma el usuario asigna un pin físico de la FPGA al parámetro de entrada o salida definido. Editar manualmente el archivo de constraints puede resultar una tarea difícil, pues se requiere conocer el identificador interno de cada pin y además la sintaxis específica para que el programa interprete el archivo.

El ISE incluye una herramienta gráfica denominada PACE, que permite al usuario asignar fácilmente los pines de la FPGA. La herramienta presenta un listado con las entradas y salidas definidas por el usuario en la entidad del código fuente, y al mismo tiempo muestra la distribución de pines en la FPGA, especificando las características individuales de cada pin como se observa en la *Fig. 5.16*.

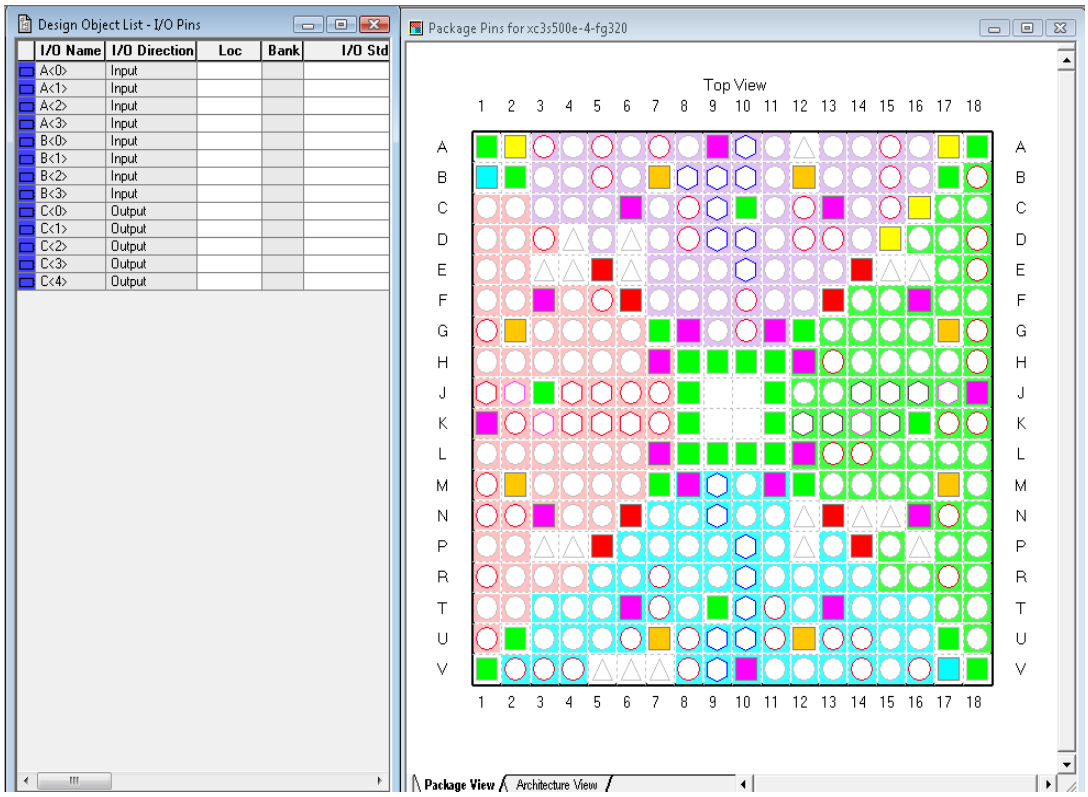

Figura 5.16 *Interfaz Gráfica del PACE.*

La asignación de pines se realiza introduciendo en la tabla de asignación el identificador correspondiente al pin que se quiera utilizar. Los pines están distribuidos en un arreglo bidimensional de donde es posible deducir el identificador correspondiente.

Se debe tener en cuenta que dependiendo de la FPGA que se utilice, la distribución de pines variará. Además existen pines cuya utilización está restringida para el usuario, pues están reservados internamente por el sistema.

Una vez realizada la asignación gráfica de los pines, sólo basta con guardar el archivo de constraints generado, para que automáticamente se agregue a la estructura del proyecto desarrollado (*Fig. 5.17)*. Este archivo puede ser editado por el usuario, para realizar correcciones o para agregar nuevos parámetros al diseño implementado.

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "A<0>"  LOC = "A14"  ;
NET "A<1>"  LOC = "A13"  ;
NET "A<2>"  LOC = "B14"  ;
NET "A<3>"  LOC = "B13"  ;
NET "B<0>"  LOC = "C14"  ;
NET "B<1>"  LOC = "D14"  ;
NET "B<2>"  LOC = "D11"  ;
NET "B<3>"  LOC = "E11"  ;
NET "C<0>"  LOC = "E9"  ;
NET "C<1>"  LOC = "F7"  ;
NET "C<2>"  LOC = "G6"  ;
NET "C<3>"  LOC = "J13"  ;
NET "C<4>"  LOC = "K14"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

```

Figura 5.17 *Constraints Generadas a través del PACE.*

Después de la creación del archivo de constraints es posible comenzar el proceso de implementación. En general, la implementación es adecuar el proyecto creado, a las capacidades físicas del dispositivo destino. Este proceso puede dividirse en tres fases: Traducción, Mapeado e Implementación (*Fig. 5.18)*.

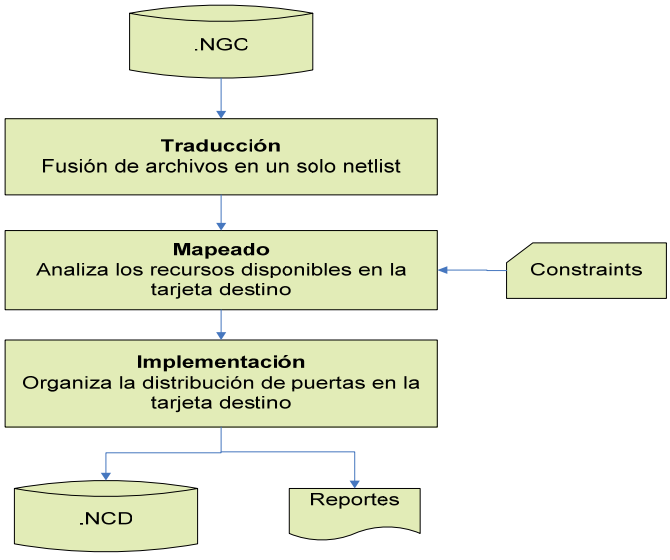

Figura 5.18 *Fases del Proceso de Implementación.*

5.9.1 Traducción

El proceso de traducción de los ficheros netlist obtenidos de la síntesis, consiste en generar una base de datos genérica (ngd), que describe el diseño lógico implementado, reducido a puertas lógicas.

Cada vez que el archivo de constraints sea modificado, el proceso de traducción se deberá ejecutar nuevamente, y así generar la nueva asociación de puertas en el dispositivo.

5.9.2 Mapeado

El proceso de mapeado consiste en asignar los dispositivos descritos en el fichero de extensión ngd, ubicando los bloques lógicos (CLB) y los bloques de entrada y salida (IOB) en los que se implementará la aplicación. El resultado de esta fase es un diseño nativo del circuito (ncd) con los archivos que físicamente representan el diseño asignado sobre los componentes de la FPGA.

5.9.3 Implementación

El proceso de implementación toma el archivo de mapeado (NCD), para aplicar el direccionamiento y fijación del diseño. Como resultado, se reescribe el archivo NCD, el cual será usado para generar un archivo de flujo de bits o bitstream, que será utilizado para generar los archivos de descarga para el hardware programable.

El último paso en el proceso de diseño de un modelo mediante lenguaje de descripción de hardware, es la descarga al dispositivo programable. El entorno ISE posee una aplicación destinada para tal fin denominada iMPACT, en la que se puede realizar la configuración de los distintos parámetros de configuración del dispositivo programable.

En este documento no se profundizará en las distintas opciones de programación con las que cuenta el iMPACT, pero en el [Pulido, 2007], se adjunta un manual donde se describen las diferentes posibilidades que brinda el iMPACT.

6 DISEÑO DE UN SISTEMA EMBEBIDO EN FPGA

En este capítulo se presenta el entorno de programación Embedded Development Kit (EDK) de Xilinx, utilizado como herramienta de diseño para la implementación de sistemas embebidos en FPGA.

Un sistema embebido se puede definir como la combinación de elementos hardware – software y en algunos casos componentes mecánicos, diseñados para cumplir una función específica. Pero esta definición no estaría completa, si no se menciona que el término embebido indica que el dispositivo desarrollado está implantado o construido sobre otro sistema ya existente.

Con el auge de los lenguajes de descripción de hardware, nace una tendencia que apunta a la construcción de dispositivos electrónicos implementados sobre estructuras programables de bajo costo, esto con el objetivo de aumentar la productividad, disminuyendo los periodos de diseño y producción de los dispositivos.

La implementación de componentes IP Core descritos en lenguaje de hardware (VHDL o Verilog) sobre una FPGA, permite desarrollar completamente la estructura de un sistema embebido. Como ya se había mencionado en capítulos anteriores, existen procesadores soft-core (MicroBlaze, PicoBlaze), que conforman la base del sistema embebido permitiendo controlar e interactuar con el sistema.

A continuación se describen los pasos a seguir para implementar un sistema embebido sobre FPGA, basado en el procesador MicroBlaze de Xilinx, teniendo en cuenta por separado los proyectos de hardware y software que permiten la puesta en marcha del sistema.

6.1 INTRODUCCIÓN AL ENTORNO EDK DE XILINX

El entorno de desarrollo EDK está compuesto por una serie de aplicaciones que permiten la configuración hardware, la programación software y el desarrollo de periféricos, para ser implementados en conjunto como un sistema embebido.

La creación y diseño del proyecto de hardware es realizada mediante la herramienta Xilinx Platform Studio (XPS), mientras que el de desarrollo para software es realizado en la herramienta Software Development Kit (SDK).

Sin embargo el entorno EDK por si sólo no es suficiente para desarrollar y cargar un proyecto sobre la FPGA. Todas las tareas de implementación, mapeado y descarga sobre

la FPGA, deben ser realizadas sobre el entorno ISE de Xilinx. De esta manera es necesario instalar el ISE antes de comenzar a trabajar sobre el EDK, pues de lo contrario no será posible la implementación del diseño.

La plataforma XPS incluye una serie de IP Cores, con las cuales el usuario puede construir un sistema embebido sobre la FPGA. Una de éstas Cores incluidas es el procesador embebido MicroBlaze, que es la base del sistema y al cual se conectan las demás IP Core.

Utilizando las herramientas tanto del ISE como del EDK, el usuario puede crear sus propias IP Core y conectarlas al MicroBlaze. De esta manera es posible crear nuevos periféricos de acuerdo a las especificaciones definidas por el usuario.

6.2 CONSTRUCCIÓN DEL PROYECTO DE HARDWARE

Como ya se había mencionado anteriormente la herramienta utilizada para el desarrollo del proyecto de hardware es el XPS. Una vez iniciado el XPS se ejecuta un asistente que permite al usuario elegir las opciones de configuración para el sistema.

En primera instancia aparece una ventana indicando el tipo de proyecto que se quiere implementar, es decir, si el usuario desea utilizar un esquema predefinido sobre el cual desarrollar el proyecto, o por el contrario un proyecto en blanco para que sea totalmente configurado por el usuario (*Fig. 6.1)*.

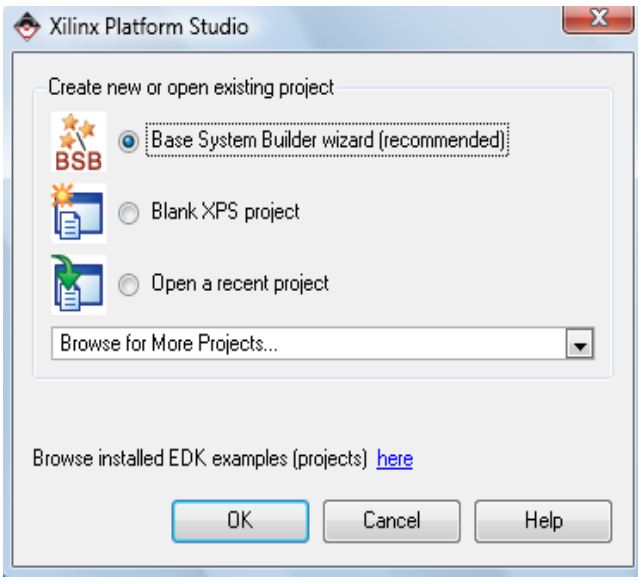

Figura 6.1 *Ventana de Inicio XPS.*

En este caso se hará referencia al desarrollo de un proyecto de hardware, utilizando el Base System Builder, que es el asistente de configuración que permite utilizar esquemas de desarrollo predefinidos.

Después de haber seleccionado el asistente de configuración se inicia una serie de pasos que permiten ajustar el sistema a los requerimientos del usuario. El primer paso es asignar un nombre al proyecto, y al mismo tiempo indicar al asistente cual será la ubicación del proyecto. Si no se asigna un nombre al proyecto, por defecto XPS lo definirá como system (*Fig. 6.2).*

Es muy importante en este paso definir cual será la carpeta en la que estarán ubicados los repositorios (ficheros fuente), siendo recomendado asignar una nueva carpeta para que éstos archivos no sean modificados accidentalmente desde otros proyectos.

En el caso de no asignar una carpeta específica para los repositorios del proyecto, se utilizará una carpeta definida por defecto en la que se archivarán todos los repositorios que hayan sido empleados por el XPS.

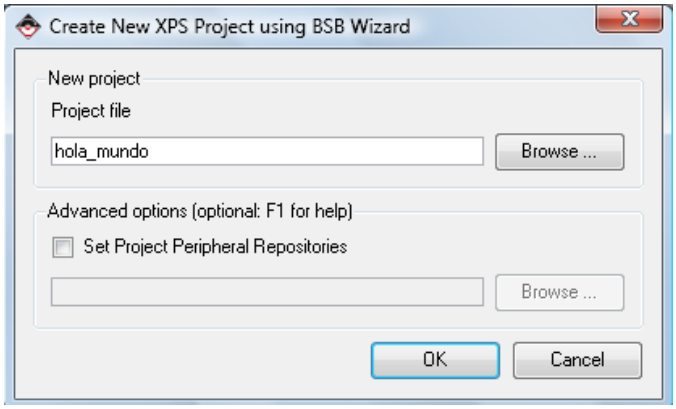

Figura 6.2 *Creación del Directorio del Proyecto.*

Una vez creado el directorio del proyecto, el asistente preguntará si el proyecto se generará a partir de un nuevo diseño o si se utilizará un diseño construido en una sesión anterior (*Fig 6.3).* En este caso se hará énfasis en la opción del desarrollo de un nuevo diseño.

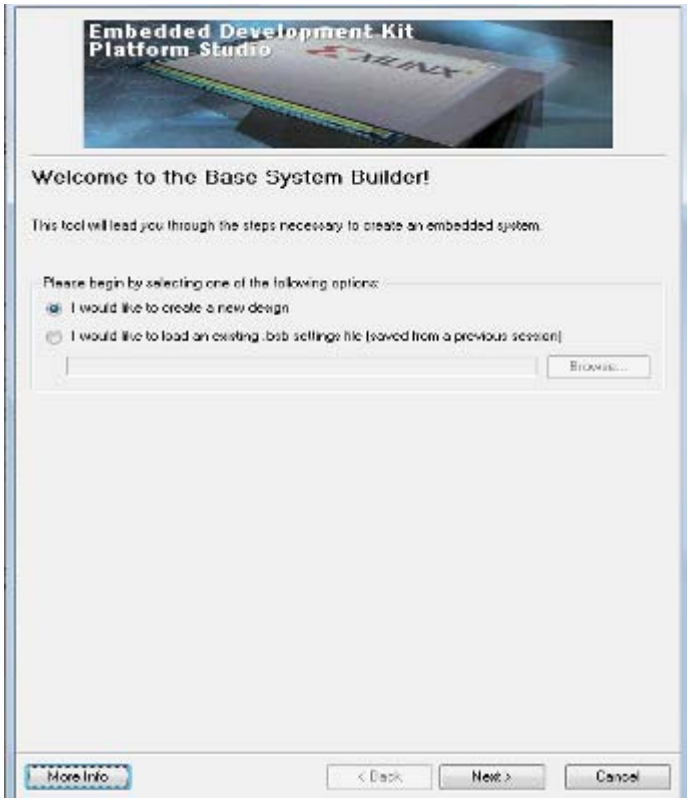

Figura 6.3 *Creación de un Nuevo Diseño.*

En la siguiente ventana el usuario puede escoger entre dos opciones, dependiendo del tipo de proyecto que realice (*Fig.6.4)*. Si el proyecto está basado en una tarjeta comercial o kit de desarrollo, podrá seleccionar en el menú desplegable el tipo de dispositivo empleado.

Pero si el proyecto se realiza sobre una tarjeta de desarrollo propio, es necesario seleccionar la opción de custom board, para así definir las características que posee la tarjeta.

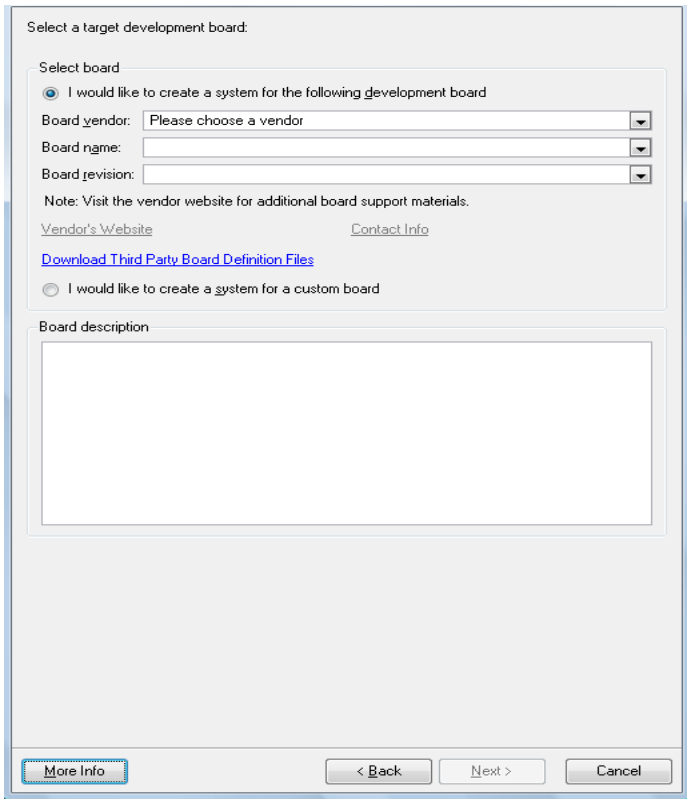

Figura 6.4 *Selección del Tipo de Tarjeta Empleada.*

La diferencia entre las dos opciones radica en que el XPS trae plantillas de proyecto predefinidas para las tarjetas comerciales, de esta manera el proyecto generado traerá las IP Core necesarias para que todos los módulos hardware incluidos en la tarjeta estén operativos.

En cambio para una tarjeta propia habrá que especificar que componentes posee la tarjeta, y definir las IP Core requeridas para que el hardware añadido pueda funcionar correctamente.

El desarrollo de este proyecto se efectuó sobre una tarjeta de desarrollo propio, denominada EOS, por esta razón se explicarán los pasos para construir un sistema de este tipo.

En la siguiente ventana el usuario puede seleccionar el tipo de FPGA empleado en el proyecto. Es muy importante conocer las características específicas del componente utilizado, es decir la arquitectura, el dispositivo, el empaquetamiento y el grado de velocidad (*Fig. 6.5)*.

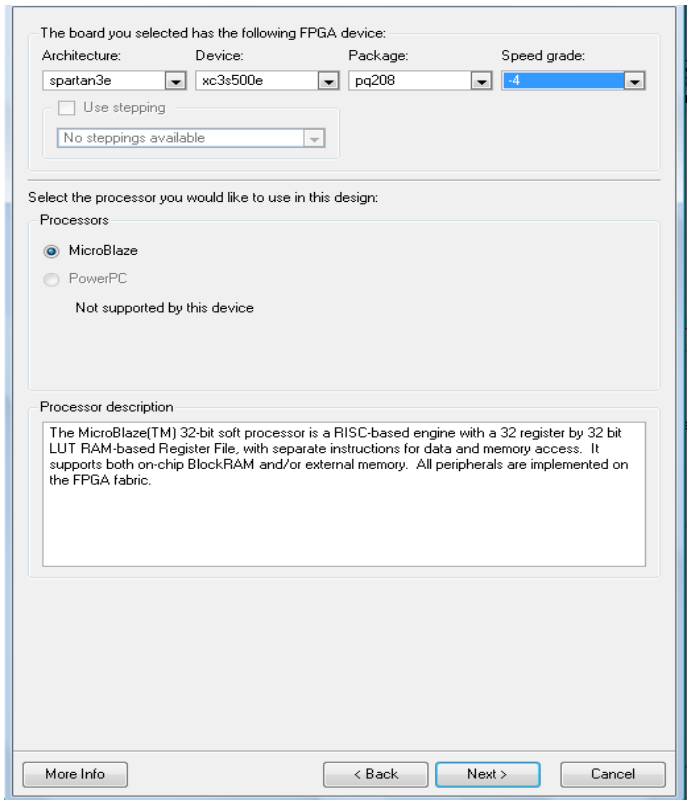

Figura 6.5 *Especificación del Tipo de FPGA Utilizado.*

Además en esta ventana, el usuario podrá seleccionar el procesador embebido que tendrá el sistema. Xilinx incluye en el EDK los Cores de dos procesadores, el MicroBlaze y el PowerPc. Dependiendo de las características de la FPGA utilizada, estará habilitada la opción para seleccionar cualquiera de ellos.

En este proyecto se emplea una FPGA Spartan 3, XC3S500E, con empaquetamiento PQ208 y grado de velocidad -4. Para esta FPGA sólo está habilitado el uso del procesador MicroBlaze (*Fig. 6.5)*.

Si se desconocen las características de la FPGA utilizada, es posible conocerlas observando los datos especificados sobre el CI de la FPGA (*Fig. 6.6)*.

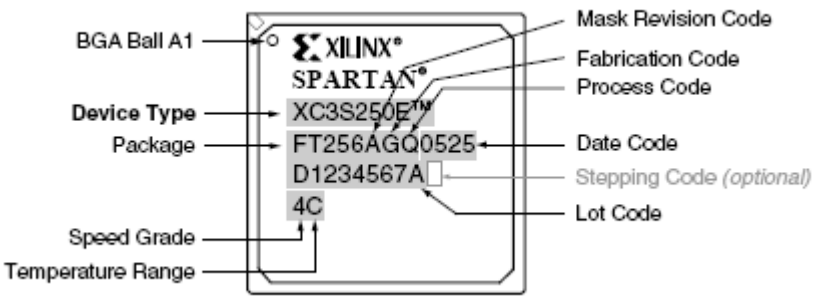

Figura 6.6 *Identificación del Empaquetamiento de la FPGA.*

Una vez definido el tipo de FPGA utilizada y de haber seleccionado el MicroBlaze, es necesario realizar la configuración de este último.

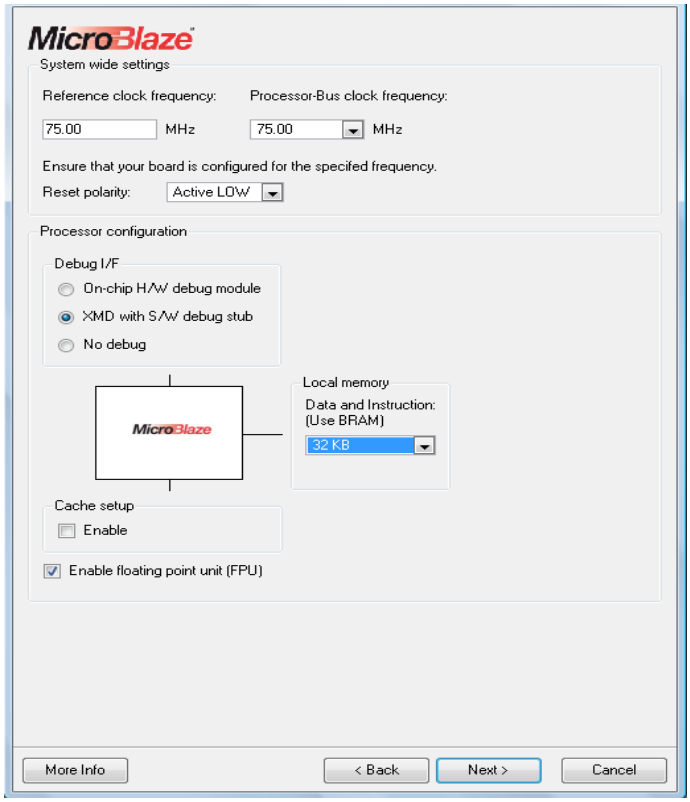

Figura 6.7 *Definición de Parámetros del Microblaze.*

La ventana de configuración del MicroBlaze (*Fig. 6.7)* está dividida en dos partes. La primera de ellas es la configuración del sistema, en donde se puede seleccionar la frecuencia de reloj a la cual estará referido el sistema y a su vez, seleccionar la frecuencia de trabajo a la que opera el bus del procesador.

Existe la posibilidad de definir la polaridad para el reset del sistema. Es necesario tener esto en cuenta a la hora de ajustar las referencias de voltaje del dispositivo, para evitar que la configuración efectuada no corresponda con la dispocisión del hardware.

En la segunda parte de la ventana se encuentra un panel de configuración que permite ajustar el tipo de depuración que el usuario prefiera. MicroBlaze posee dos tipos de depuración, uno realizado directamente desde el hardware (On Chip) y otro ejecutable en software desde el XPS (XMD). En este caso se ha elegido el XMD, pues resulta mucho más cómodo y fácil de ejecutar por el soporte en software que brinda el XPS.

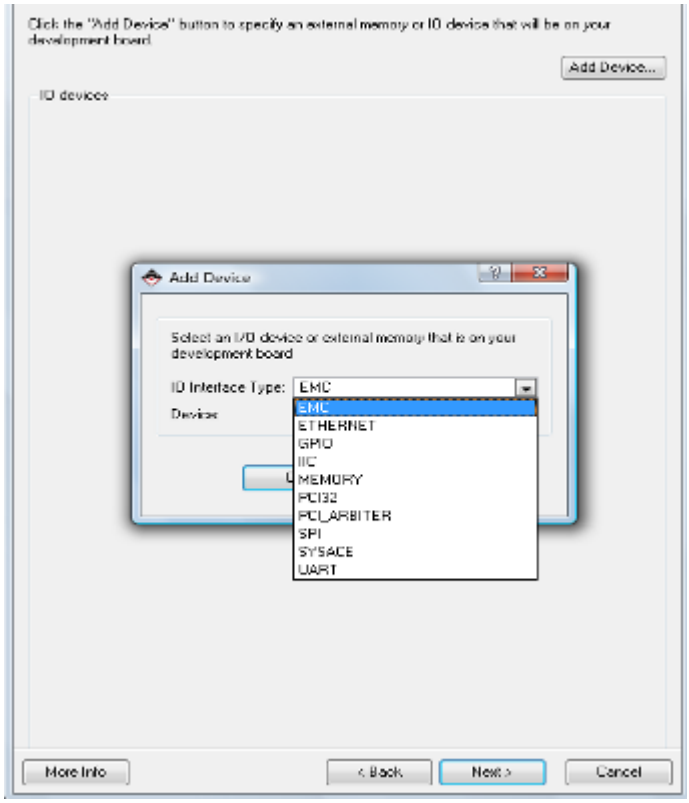

Figura 6.8 *Selección de IP Cores.*

También se debe seleccionar en este punto la capacidad de la memoria local, para el almacenamiento de datos e instrucciones. Por defecto el MicroBlaze está ajustado para una capacidad de 32 kb, pero este valor es ajustable dependiendo del tipo de distribución en memoria que el usuario realice.

Si la aplicación desarrollada requiere del manejo numérico en coma flotante, es posible dotar al MicroBlaze con una unidad de coma flotante para tal fin, al igual que de una memoria caché para incrementar el rendimiento del sistema.

Una vez definidos los parámetros del MicroBlaze, se continúa con la configuración general del sistema, siendo necesario en este punto definir las IP Core que se implementarán en el sistema (*Fig. 6.8)*. Cabe resaltar que el asistente presenta las IP Core básicas o genéricas para cualquier sistema, como controladores para memorias y protocolos de comunicación. Las IP Core desarrolladas por el usuario no se podrán agregar en este punto.

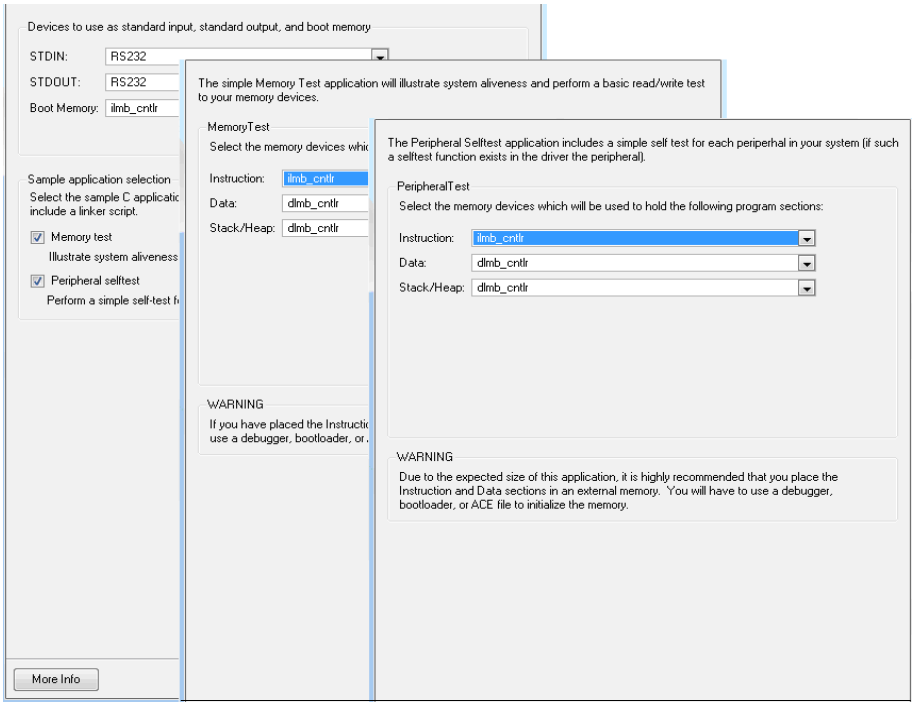

Figura 6.9 *Configuración de la Memoria y la Depuración.*

El siguiente paso es definir cuales serán los dispositivos empleados como salidas y entradas estándar, siendo generalmente utilizado para este fin el puerto R2S232.
De igual modo se consulta al usuario, si se quiere realizar un test a la memoria y a los periféricos agregados. Si se selecciona la realización del test, será necesario especificar la distribución en memoria del dispositivo (*Fig. 6.9).* Si por el contrario no se realiza ningún tipo de test, se pasa a la última etapa del proceso de configuración.

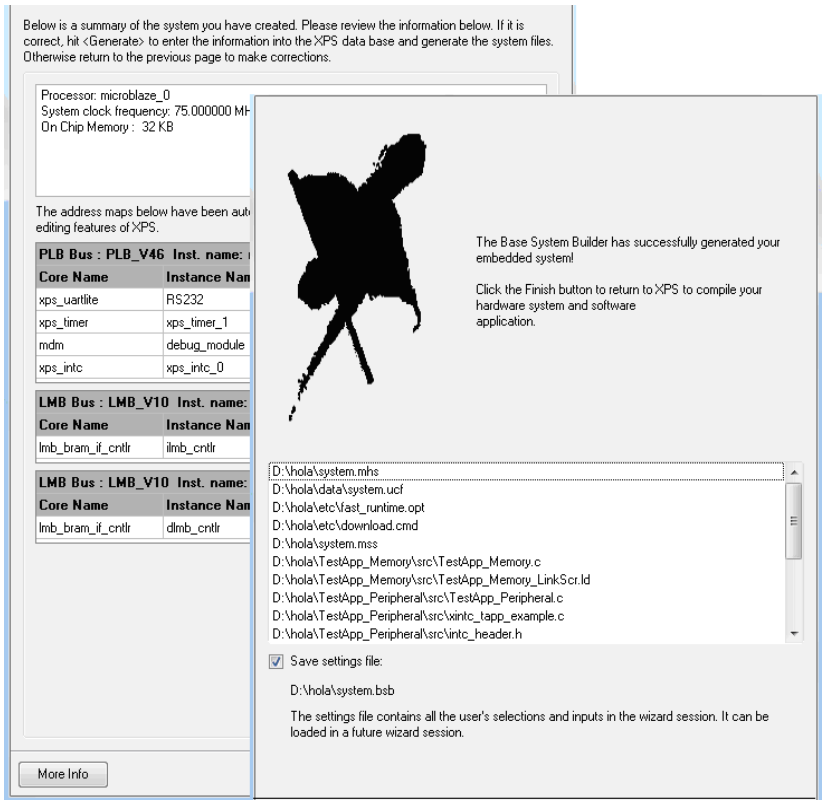

Figura 6.10 *Reporte Final del Diseño Creado.*

Por último aparecerá una ventana con el reporte del proceso de configuración efectuado, si se esta de acuerdo, se avanza indicando al asistente para que genere el sistema que ha sido especificado. De esta manera aparecerá una ventana indicando que se han creado correctamente los archivos de definición del sistema (*Fig. 6.10)* finalizando así el proceso de creación del sistema base.

6.2.1 Interfaz del XPS

La interfaz gráfica del XPS se encuentra divida en tres partes que son (*Fig. 6.11)*: Panel de información, Ventana de estructura del sistema y la ventana de informes. Cada una de estas partes será descrita a continuación:

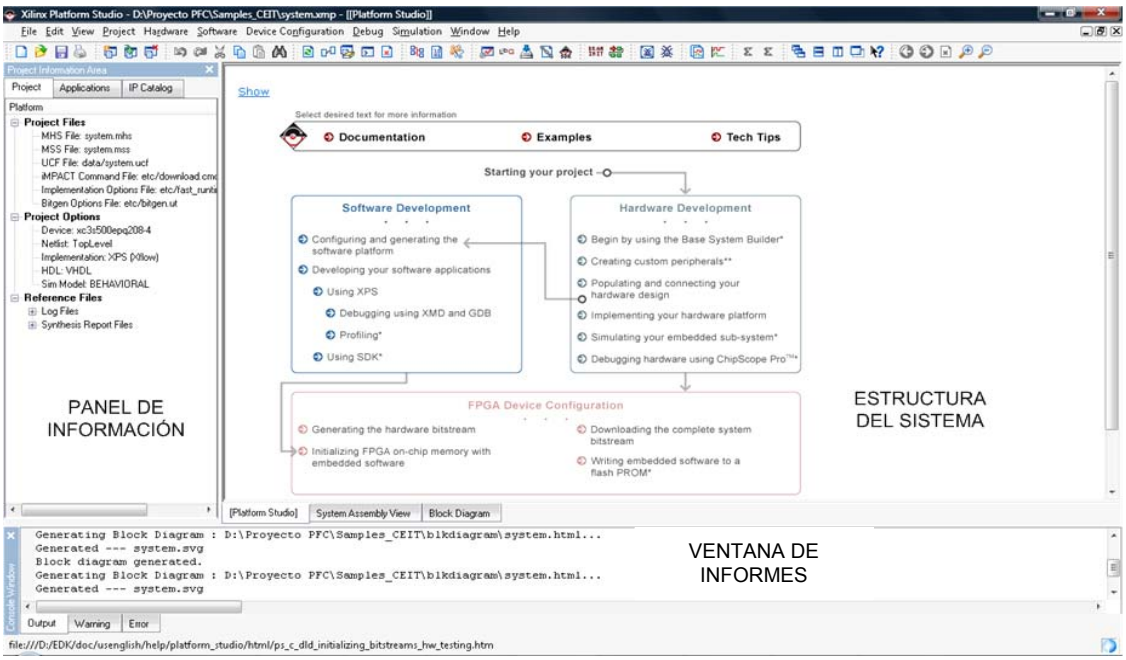

Figura 6.11 *Interfaz Gráfica del XPS.*

- Panel de Información: se encuentra al lado izquierdo del espacio de trabajo. Posee tres pestañas que permiten acceder a la estructura de archivos del proyecto, a las aplicaciones software implementadas en el MicroBlaze y al catálogo de periféricos disponibles.

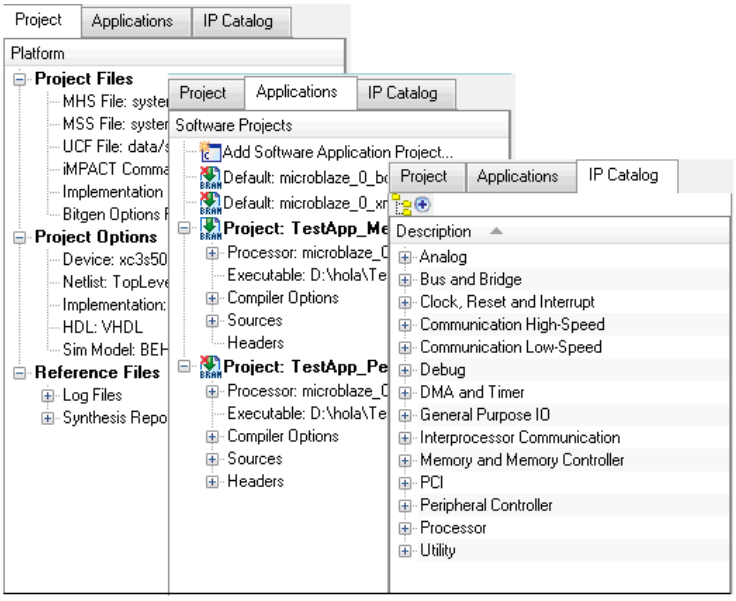

Figura 6.12 *Panel de Información.*

- Ventana de Estructura del Sistema: esta ventana está ubicada al lado derecho de la pantalla. En ella se presenta toda la información concerniente a la configuración y distribución de los componentes del sistema. Está dividida en tres pestañas: Configuración de Buses, Puertos y Direccionamiento.

  Desde cada una de estas pestañas es posible modificar la configuración de cada uno de los periféricos añadidos al sistema, y del mismo modo es posible establecer los parámetros de configuración del MicroBlaze como se verá mas adelante.

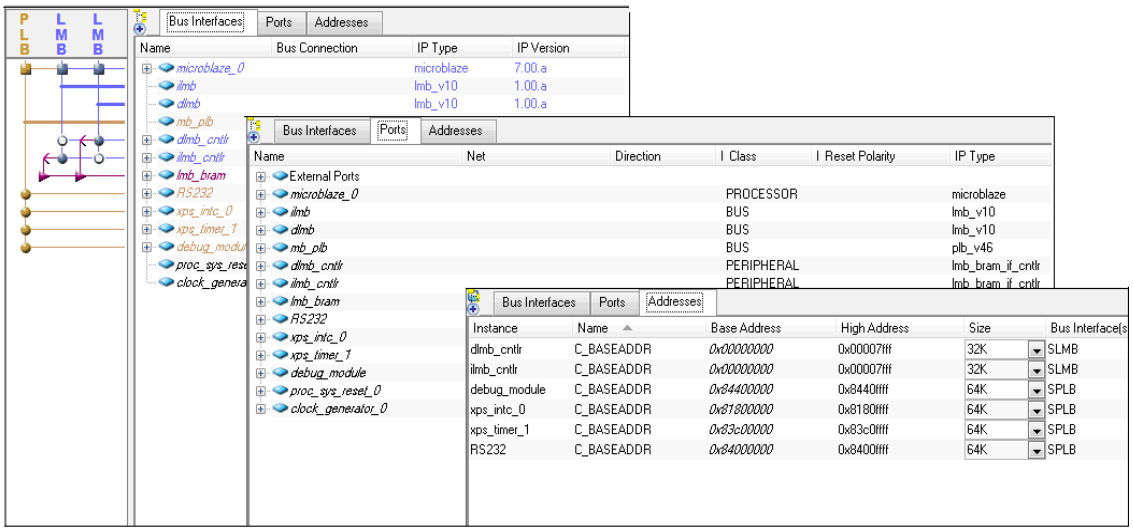

Figura 6.13 *Estructura del Sistema.*

- Ventana de Informes: Esta ventana se encuentra en la parte inferior de la interfaz del XPS. En esta ventana se presentan los informes de la ejecución de cualquier proceso realizado en el XPS, además se especifica la ubicación de cualquier error encontrado durante la compilación del proyecto.

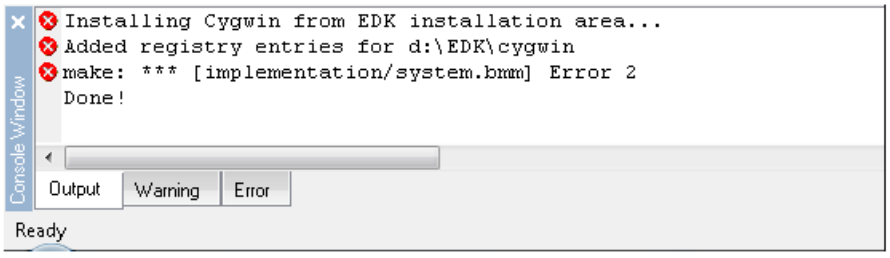

Figura 6.14 *Ventana de Reportes.*

6.2.2 Archivos de Configuración del Sistema

Desde el panel de información es posible acceder a la estructura de archivos que conforman la estructura del proyecto realizado.

Existen tres ficheros en los que se registran los parámetros introducidos por el usuario, tanto de la configuración del proyecto de hardware como del proyecto de software (*Fig. 6.15)*. Estos ficheros son: Archivo de configuración de hardware, archivo de descripción de software y el archivo de restricciones (constraints).

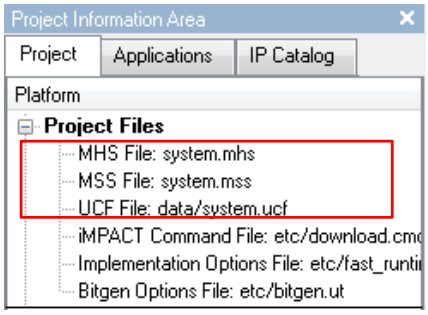

Figura 6.15 *Archivos de Configuración del Sistema.*

- Archivo de Configuración de Hardware: Este archivo con extensión mhs (Microprocessor Hardware Specification) contiene los parámetros definidos para los puertos del sistema (*Fig. 6.16)*. Cada uno de los puertos debe ser especificado con un nombre, tipo (entrada o salida) y el número de bits que componen el puerto.

  En este archivo se especifican las opciones de configuración del MicroBlaze, describiendo los buses de comunicación y los parámetros de funcionamiento del dispositivo.

  Además en este archivo se realiza el instanciamiento de los periféricos del sistema, así como la configuración de estos mismos, por ejemplo se describen los puertos utilizados y del mismo modo las localidades de memoria en las que estará alojado el componente. Es posible además definir las opciones de hardware de un periférico a través del archivo mpd (microprocessor peripheral description) y a partir de este actualizar los datos registrados en el archivo mhs.

```
 14   PARAMETER VERSION = 2.1.0
 15
 16   PORT fpga_0_RS232_RX_pin = fpga_0_RS232_RX, DIR = I
 17   PORT fpga_0_RS232_TX_pin = fpga_0_RS232_TX, DIR = O
 18   PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 75000000
 19   PORT sys_rst_pin = sys_rst_s, DIR = I, RST_POLARITY = 0, SIGIS = RST
 20
 21  BEGIN microblaze
 22   PARAMETER HW_VER = 7.00.a
 23   PARAMETER INSTANCE = microblaze_0
 24   PARAMETER C_INTERCONNECT = 1
 25   PARAMETER C_USE_FPU = 1
 26   PARAMETER C_DEBUG_ENABLED = 0
 27   PARAMETER C_AREA_OPTIMIZED = 1
 28   BUS_INTERFACE DLMB = dlmb
 29   BUS_INTERFACE ILMB = ilmb
 30   BUS_INTERFACE DPLB = mb_plb
 31   BUS_INTERFACE IPLB = mb_plb
 32   PORT RESET = mb_reset
 33   PORT Interrupt = Interrupt
 34  END
 35
 36  BEGIN plb_v46
 37   PARAMETER INSTANCE = mb_plb
 38   PARAMETER HW_VER = 1.00.a
 39   PORT PLB_Clk = sys_clk_s
 40   PORT SYS_Rst = sys_bus_reset
 41  END
```

Figura 6.16 *Archivo de Configuración de Hardware.*

- Archivo de Descripción de Software: Este archivo con extensión mss (Microprocessor Software Specification) contiene las opciones de compilación del proyecto de software (*Fig. 6.17)*. Incluye la especificación del modo de compilación del código, asignación de librerías a periféricos, especificación del método de solución de errores, y otras opciones del compilador.

  Este archivo está relacionado directamente con el código de descripción de hardware del dispositivo implementado, pues debe especificarse el controlador para cada uno de los periféricos instanciados. De igual forma, si en el proyecto se utilizan recursos hardware específicos, por ejemplo un multiplicador, es en este archivo donde se debe indicar al compilador que utilice este recurso hardware.

```
 1
 2   PARAMETER VERSION = 2.2.0
 3
 4
 5  BEGIN OS
 6   PARAMETER OS_NAME = standalone
 7   PARAMETER OS_VER = 1.00.a
 8   PARAMETER PROC_INSTANCE = microblaze_0
 9   PARAMETER STDIN = RS232
10   PARAMETER STDOUT = RS232
11  END
12
13
14  BEGIN PROCESSOR
15   PARAMETER DRIVER_NAME = cpu
16   PARAMETER DRIVER_VER = 1.11.a
17   PARAMETER HW_INSTANCE = microblaze_0
18   PARAMETER COMPILER = mb-gcc
19   PARAMETER ARCHIVER = mb-ar
20   PARAMETER XMDSTUB_PERIPHERAL = debug_module
21  END
22
23
24  BEGIN DRIVER
25   PARAMETER DRIVER_NAME = bram
26   PARAMETER DRIVER_VER = 1.00.a
27   PARAMETER HW_INSTANCE = dlmb_cntlr
28  END
```

Figura 6.17 *Archivo de Descripción de Software.*

- Archivo de Restricciones: Este archivo con extensión ucf (User Constraint File) contiene la asignación de los pines de entrada y salida que serán utilizados en el proyecto realizado. Del mismo modo en este archivo de deben definir los parámetros de configuración de los pines de la FPGA, es decir, los voltajes de operación y el tipo de activación (pull up o pull down) que se utilizará.

```
 1  ##############################################################################
 2  ## This system.ucf file is generated by Base System Builder based on the
 3  ## settings in the selected Xilinx Board Definition file. Please add other
 4  ## user constraints to this file based on customer design specifications.
 5  ##############################################################################
 6
 7  # Net sys_clk_pin LOC=;
 8  # Net sys_rst_pin LOC=;
 9  ## System level constraints
10  Net sys_clk_pin TNM_NET = sys_clk_pin;
11  TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 13333 ps;
12  Net sys_rst_pin TIG;
13
14  ## IO Devices constraints
15
16  #### Module RS232 constraints
17
18  # Net fpga_0_RS232_RX_pin LOC=;
19  # Net fpga_0_RS232_TX_pin LOC=;
20
21
22
```

Figura 6.18 *Archivo de Constraints.*

6.3 CREACIÓN Y ADICIÓN DE NUEVOS PERIFÉRICOS

Una de las principales características que posee el EDK, es la posibilidad de crear y agregar nuevos periféricos al sistema implementado. Se pueden considerar tres fases en el proceso de desarrollo de un nuevo periférico: Creación del nuevo periférico, adición del periférico y finalmente la configuración del bus de comunicaciones utilizado para conectar el periférico con el MicroBlaze.

6.3.1 Creación de un Nuevo Periférico

El XPS cuenta con una asistente para la creación de y adición de nuevos periféricos al sistema (*Fig. 6.19)*. Es de resaltar que este asistente no tiene como función la compilación de código, de esta manera los componentes que se quieran agregar deben ser previamente compilados y probados en el ISE.

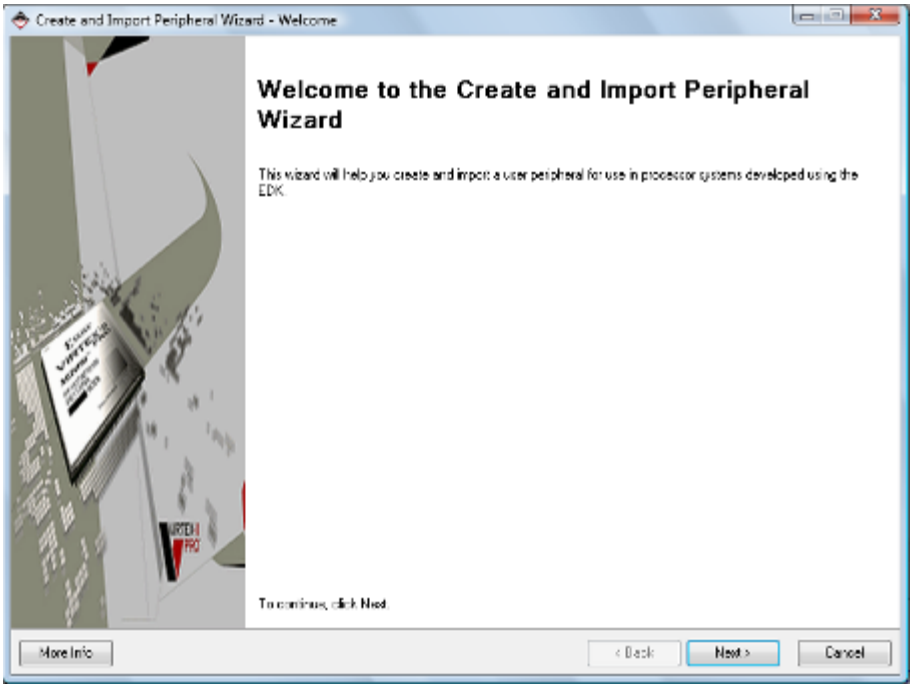

Figura 6.19 *Asistente para la Creación de Nuevos Periféricos.*

Una vez iniciado el asistente para la creación de nuevos periféricos, el primer paso será determinar si se quiere crear el modelo de un nuevo periférico, o si por el contrario se trabajará con el modelo de un componente previamente elaborado (*Fig. 6.20)*.

En este caso, se elegirá la opción de crear el modelo de un nuevo componente, teniendo en cuenta que de esta manera no se creará el periférico definitivo, tan sólo se describirán los archivos de referencia para crear el nuevo dispositivo.

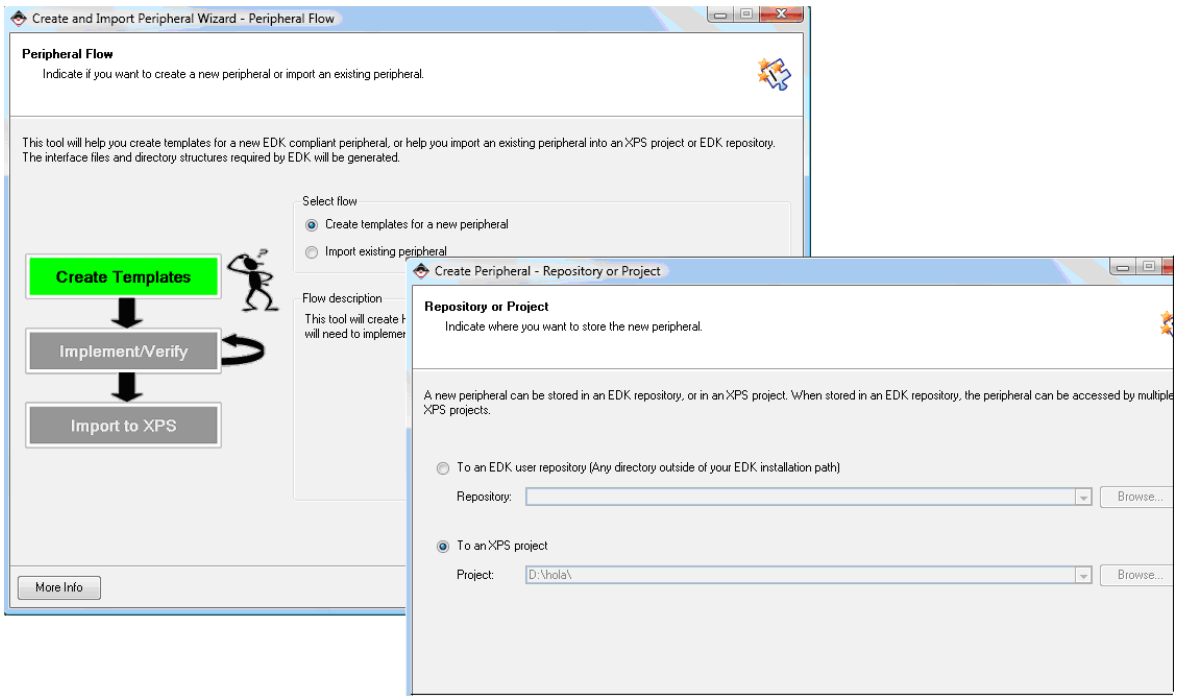

Figura 6.20 *Creación del Modelo para un Nuevo Periférico.*

En esta instancia también se debe definir el directorio en el que será creado el nuevo periférico. Existen dos opciones: escoger como directorio destino la carpeta de repositorios general del XPS, permitiendo de esta manera que el componente creado esté disponible para cualquier proyecto realizado en el XPS.

La otra opción es escoger como destino, el archivo de repositorios local del proyecto. De esta manera el periférico creado sólo estará disponible para el proyecto actual, y si en algún momento fuese necesario utilizarlo desde otro proyecto, se deberá ejecutar nuevamente el asistente.

En la siguiente ventana es necesario definir el nombre con el que será identificado el nuevo dispositivo (*Fig. 6.21)*. Es posible al mismo tiempo colocar el número de la versión del hardware que se está implementando, esto con el fin de organizar las diferentes pruebas o evoluciones que se hagan en el desarrollo de un dispositivo.

El nombre asignado al componente, debe corresponder con el nombre asignado a la entidad del módulo de mayor jerarquía, a partir del cual se creará el dispositivo. De no respetarse esta convención, no será posible habilitar el funcionamiento del dispositivo, pues su direccionamiento será incorrecto.

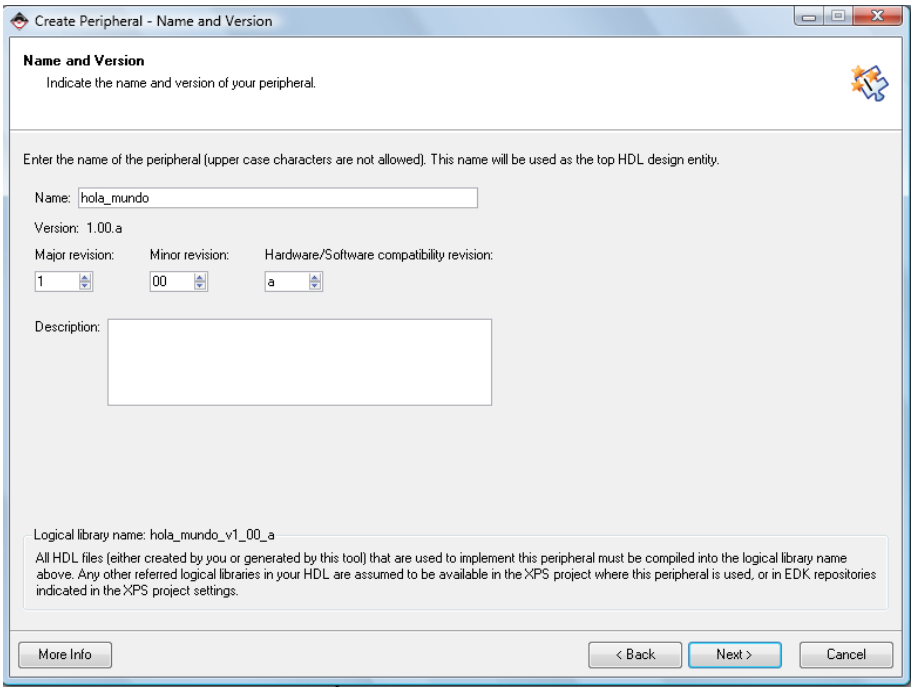

Figura 6.21 *Identificación del Nombre y la Versión del Hardware.*

El siguiente paso es la selección del bus de comunicaciones que se usará para conectar el periférico con el MicroBlaze. El XPS proporciona tres tipos de buses de comunicación disponibles en el protocolo de comunicaciones CoreConnect de IBM, como ya se había mencionado anteriormente. Estos buses son: Processor Local Bus (PLB), On Chip Peripheral Bus (OPB) y Fast Simplex Bus (FSL).

Dependiendo del tipo de aplicación desarrollada, se escogerá uno de los tres buses mencionados, para establecer una interfaz de comunicación entre los dispositivos del sistema. En este caso se empleará una interfaz de comunicación basada en el bus FSL, que ofrece una mayor velocidad de operación.

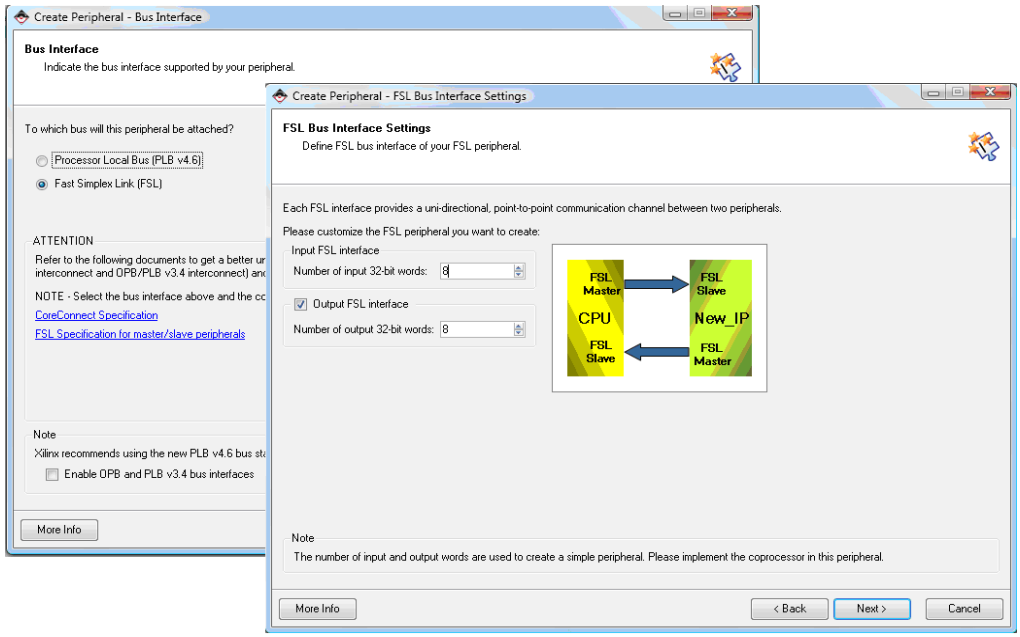

Figura 6.22 *Elección del Bus FSL.*

Una vez seleccionado el bus FSL como el protocolo de comunicación a utilizar, es necesario definir el número de canales de comunicación (entrada - salida) que serán empleados por el dispositivo.

El MicroBlaze permite agregar hasta 16 canales de comunicación FSL, organizados en parejas (8 de entrada y 8 de salida), recordando que este estos canales son unidireccionales.

Otro elemento a tener en cuenta, es la organización punto a punto (maestro - esclavo) que utiliza el bus FSL. De esta manera es necesario tener clara la organización que se utilizará para definir cada canal de comunicación, es decir, si se define un canal de comunicación desde el puerto de salida Maestro del MicroBlaze, este deberá conectarse al puerto de entrada Esclavo del periférico y viceversa (*Fig. 6.23).*

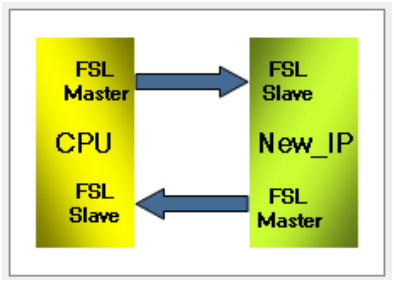

Figura 6.23 *Configuración Maestro – Esclavo del Bus FSL.*

Finalmente el asistente de configuración consultará si se quieren crear plantillas de ejemplo para llevar a cabo la configuración del bus FSL. Del mismo modo se crearán los archivos tanto de descripción de hardware (VHDL o Verilog) y los ficheros en los que se implementarán los controladores del dispositivo (*Fig. 6.24)*.

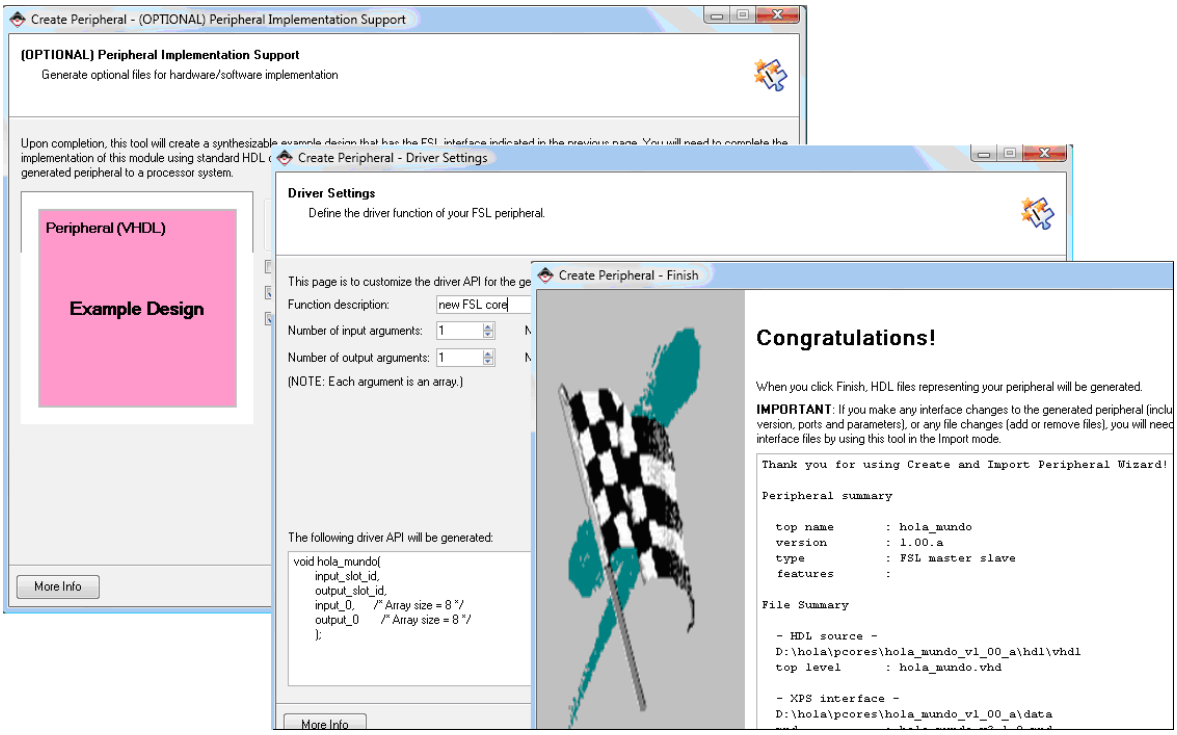

Figura 6.24 *Creación de Ejemplos y Finalización del Proceso.*

Los archivos generados por el asistente deberán ser modificados por el usuario, realizando el instanciamiento de componentes y agregando el código de los controladores del dispositivo. Estos dos procesos se verán más adelante.

Por último se presenta una hoja de informe con todos los parámetros de configuración especificados e indicando que la creación de los archivos que describen el dispositivo ha sido satisfactoria.

6.3.2 Adición de un Nuevo Periférico

El proceso para la adición de un nuevo componente, inicia con la ejecución del asistente para creación y adición de nuevos periféricos (*Fig. 6.25)*. Para este caso se elegirá la opción de importar un dispositivo ya existente y se definirá el directorio de destino para el

periférico que se adjuntará. Es necesario tener en cuenta las sugerencias realizadas en el apartado anterior para establecer el directorio de destino del nuevo componente.

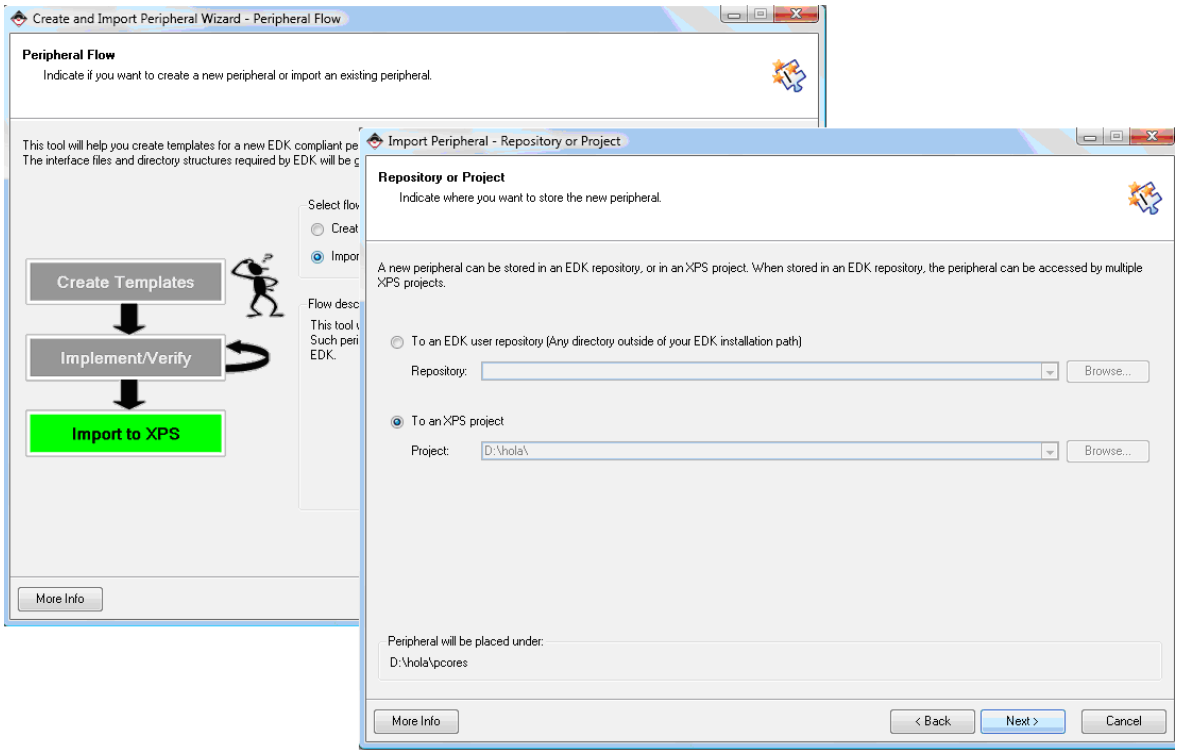

Figura 6.25 *Adición de un Nuevo Periférico.*

Del mismo modo es necesario agregar el nombre que identificará al dispositivo dentro del proyecto. Como ya se había visto en la *Fig. 6.21,* se debe especificar el nombre de la identidad del módulo de mayor jerarquía, pues a partir de este el XPS realizará la descripción del nuevo componente.

En la siguiente ventana se deberá especificar al asistente, cuales son los archivos fuente que conforman el dispositivo. Existen tres tipos de archivo fuente (*Fig. 6.26)*:

- Archivos de descripción de hardware: son los ficheros desarrollados en VHDL o Verilog, en los que está descrito el comportamiento como tal del dispositivo.

- Archivos Netlist: estos ficheros de extensión ngc, edif o ngd, son el resultado del proceso de síntesis de un componente. Este fichero posee la traducción de un diseño creado en lenguaje de descripción de hardware, a una tecnología de implementación específica.

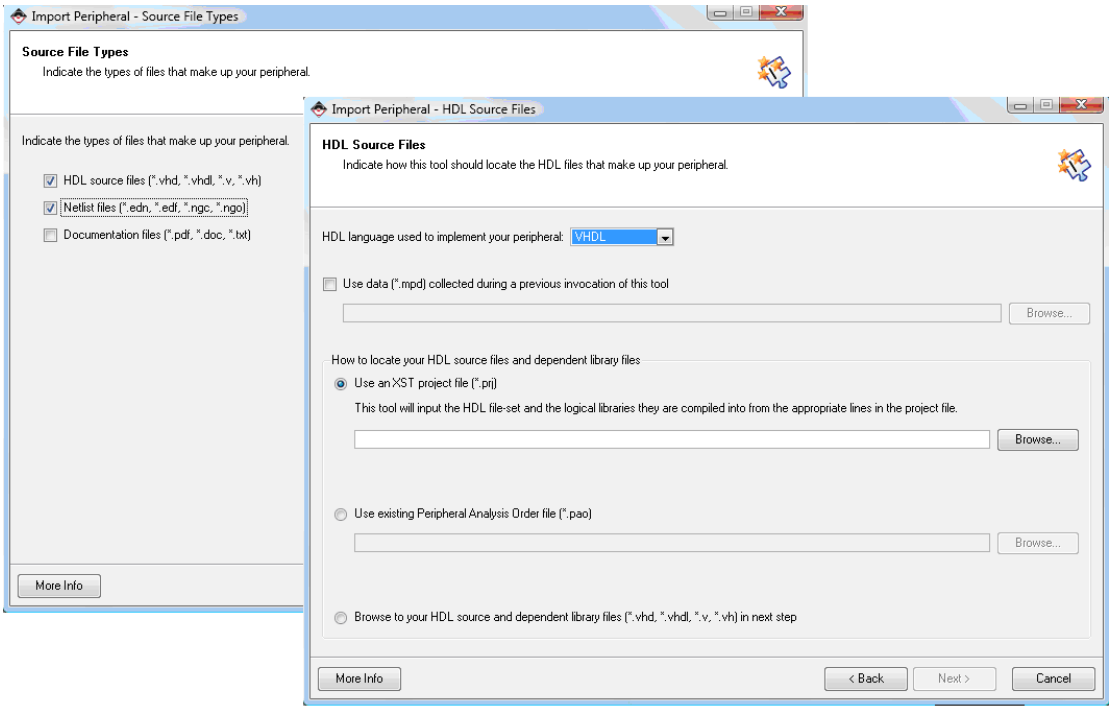

Figura 6.26 *Especificación de Archivos Fuente.*

De esta manera a partir de un Netlist, es posible realizar el proceso de implementación del dispositivo, por ejemplo, en el caso de utilizar un modelo obtenido a partir del CoreGenerator será necesario utilizar el netlist de dicho modelo, esto debido a la estructura de código cerrado que maneja esta aplicación.

- Archivos de Documentación: es posible adjuntar al dispositivo creado, ficheros de documentación que permitan a cualquier usuario conocer información detallada acerca del componente. Esta documentación puede estar en formato pdf, doc o txt.

Una vez seleccionados los archivos fuente que se utilizarán, el asistente solicita especificar los archivos que describen la configuración del componente que será agregado. Estos archivos pueden haber sido generados por el propio usuario, o también pueden ser el resultado de la implementación de una versión anterior del mismo componente. Existen tres archivos mediante los cuales se puede indicar al XPS las características de configuración del nuevo componente (*Fig. 6.26)*:

- Microprocessor Peripheral Description (mpd): en este archivo se especifican los puertos e interfaces de comunicación utilizados en el componente. Los parámetros especificados en este archivo serán configurables desde la interfaz gráfica del XPS y a partir de este generar el mhs descrito anteriormente.

- Peripheral Analysis Order (pao): en este archivo se describe el orden jerárquico en el que deben ser leídas las librerías o archivos fuente, que conforman el dispositivo. Este archivo no es obligatorio para la generación del proyecto, pero es útil para mantenerlo organizado.

- Archivo de Descripción de Proyecto (prj): es posible agregar el archivo de descripción completa de un proyecto anterior, y a partir de este generar el nuevo componente. Esta opción es recomendable si se tiene la certeza que la configuración del proyecto anterior a partir del cual se trabaja, es totalmente compatible con el nuevo dispositivo.

En el caso especial de no haber seleccionado ni el archivo de descripción de proyecto, ni el archivo .pao, es necesario para continuar con la configuración del dispositivo, elegir la opción de búsqueda manual de archivos fuente HDL.

En la siguiente ventana, el asistente permite seleccionar manualmente los archivos fuente HDL que conforman el proyecto. Es necesario ingresar los archivos HDL, respetando la jerarquía de los módulos, pues en este orden serán analizados por el XPS. (*Fig. 6.27).*

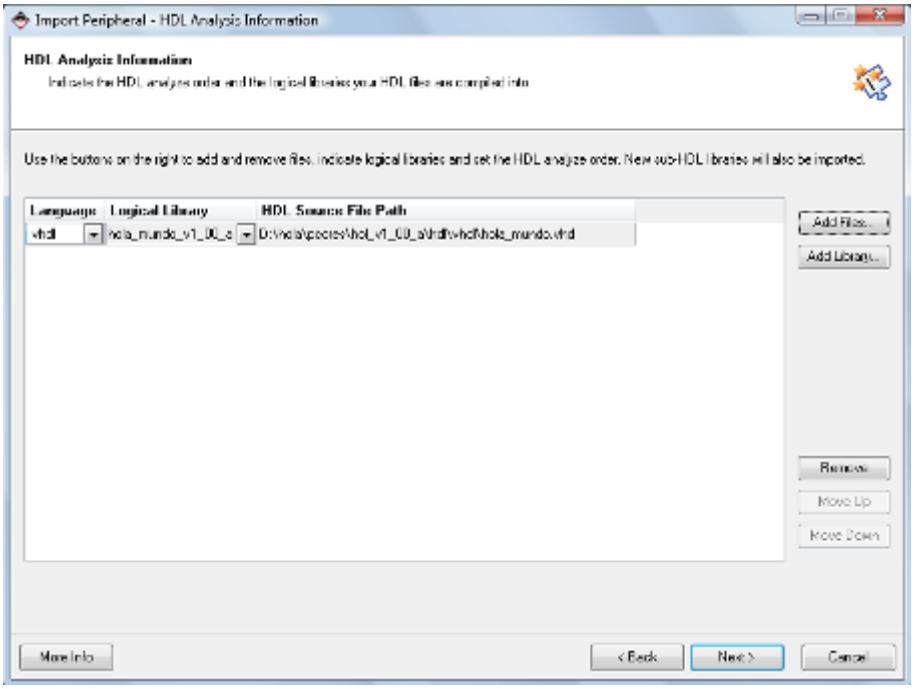

Figura 6.27 *Adición de Archivos Fuente HDL.*

En la siguiente ventana se deben seleccionar las interfaces de comunicación con las que cuenta el nuevo dispositivo, en este caso se ha definido como interfaz de comunicación el bus FSL. Será necesario especificar que se utilizarán tanto los puertos Maestros como los Esclavos para que sean habilitados en la configuración del proyecto (*Fig. 6.28).*

Es de resaltar que en el archivo con extensión mpd especificado previamente, deben estar referenciados los puertos y conexiones del bus FSL que se usarán.

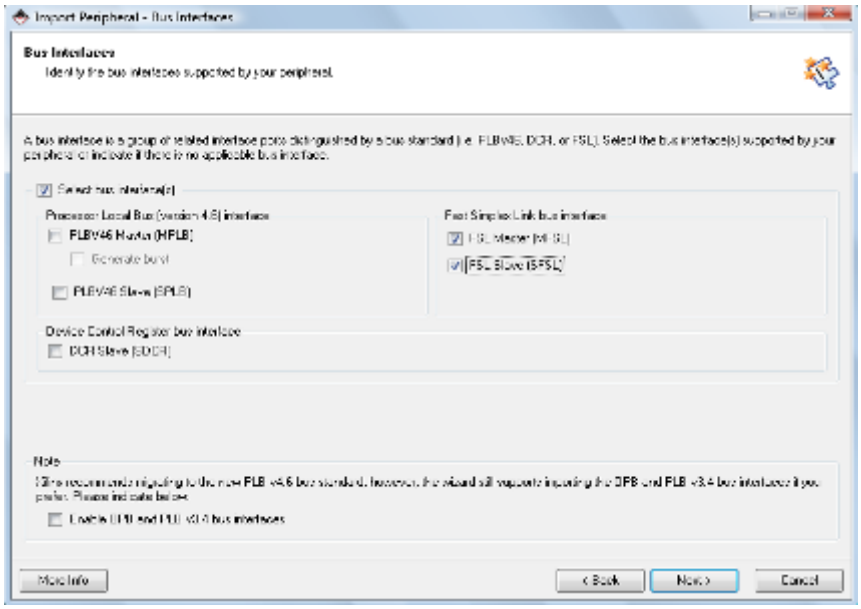

Figura 6.28 *Especificación de los Buses de Comunicación.*

Si la definición de los puertos se ha realizado correctamente, el asistente presentará un reporte con la configuración de puertos y conexiones que se ha establecido tanto para los puertos Maestros como para los Esclavos (*Fig. 6.29).*

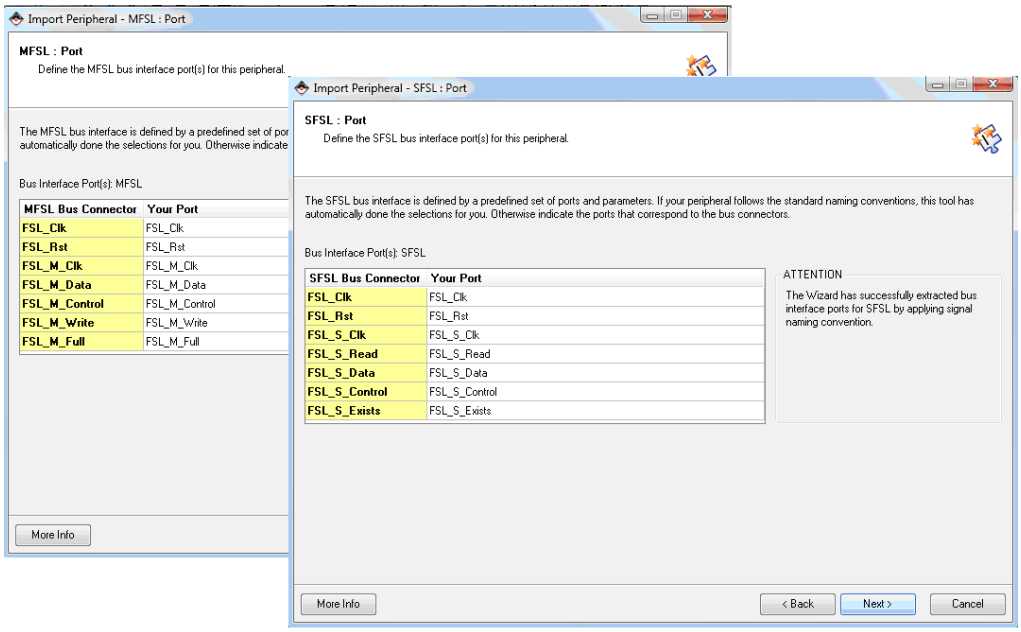

Figura 6.29 *Reporte de Configuración del Bus FSL.*

Finalmente será necesario indicar al asistente la ubicación de los archivos netlist, que previamente se han señalado como archivos fuente para el proyecto (si no se han escogido como fuentes, esta paso será omitido).

Si todo el proceso se ha cumplido satisfactoriamente, se mostrará una ventana de reporte indicando los archivos y librerías que se han creado. Del mismo modo se indican las características definidas para el bus de comunicaciones seleccionado (*Fig. 6.30)*.

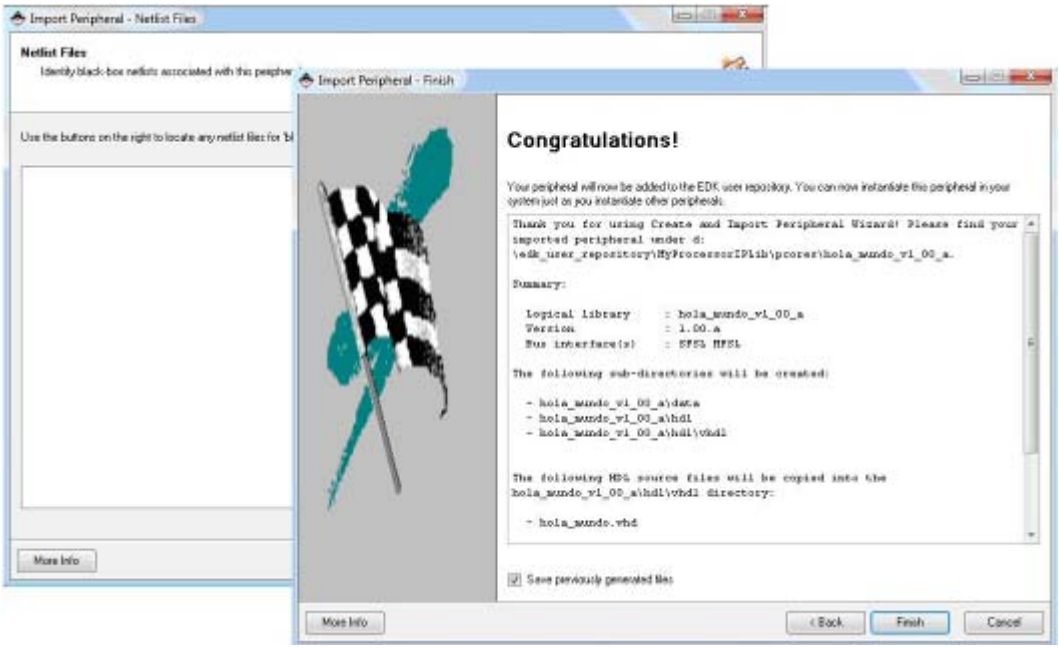

Figura 6.30 *Finalización y Reporte de Datos de la Configuración.*

Después de concluir el proceso de adición de un nuevo componente, es necesario actualizar el directorio de repositorios del proyecto, esto con el fin de habilitar la visualización del nuevo componente en el catálogo de IP Cores disponibles.

Para actualizar el directorio de repositorios, basta con seleccionar en el menú del XPS, las opciones de configuración de proyecto. Allí se selecciona la opción de actualización de repositorios del usuario para que así se modifique el catálogo de IP Cores.

Ahora basta con seleccionar la IP Core generada y arrastrarla hacia la ventana de estructura del sistema (*Fig. 6.31)*. De esta manera quedará añadida la nueva Core, pero aún no está conectada a ningún bus de comunicaciones, por lo que será necesario realizar la conexión y configuración de los buses respectivos, que han sido especificados en los parámetros internos del dispositivo agregado.

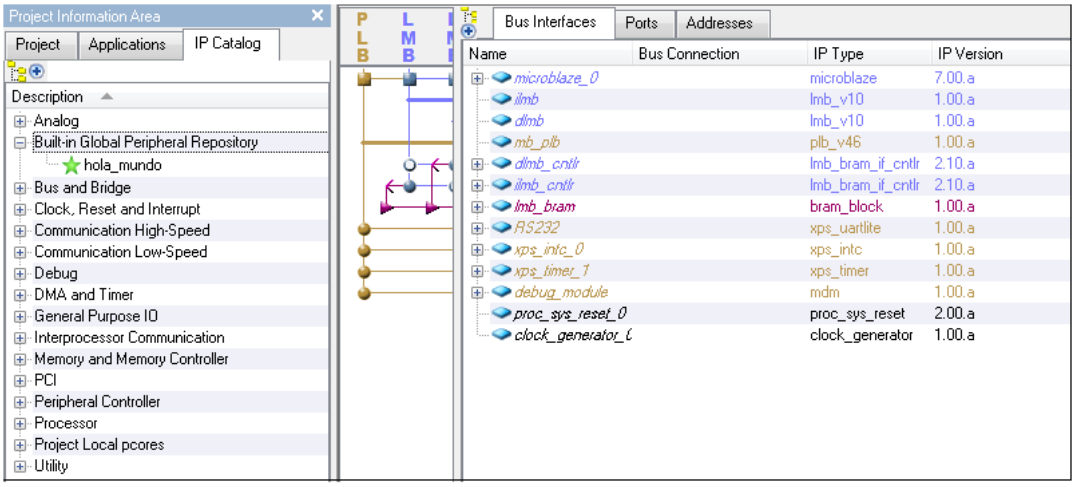

Figura 6.31 *Adición de una Nueva IP Core.*

6.3.3 Configuración del Bus FSL

Una vez agregada la nueva Core al sistema, es necesario configurar los buses de comunicación que conectan a esta con el MicroBlaze.

Desde la ventana de estructura del sistema es posible ingresar a la configuración interna del MicroBlaze, en donde hay una pestaña dedicada a la configuración de los buses del sistema. En este caso debido a que en el proyecto únicamente se emplea el bus FSL, aparecerá tan solo la opción de elegir el número de conexiones FSL que se utilizarán como se aprecia en la *Fig. 6.32.*

Como ya se había mencionado antes, el MicroBlaze permite utilizar hasta ocho parejas de conexiones FSL (Maestro - Esclavo), de esta manera la elección necesariamente se realizará en parejas y no en conexiones individuales. A cada pareja de conexiones del bus FSL se le asigna un identificador numérico (ID), que permite el direccionamiento de datos por el canal específico que desee el usuario (*Fig. 6.33)*. De esta forma la primer pareja de canales FSL será identificada como MFSL0 y SFSL0 (maestro - esclavo) respectivamente.

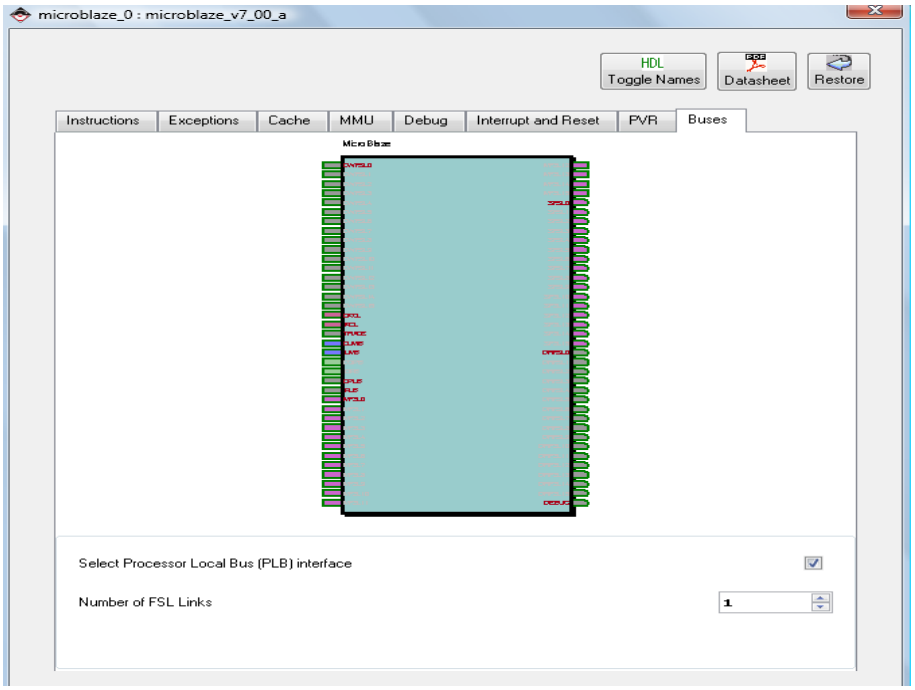

Figura 6.32 *Configuración del Bus FSL en el MicroBlaze.*

Una vez habilitados los canales FSL en el MicroBlaze, es necesario agregar en el proyecto la IP Core de los buses FSL empleados. Por cada canal FSL habilitado habrá que agregar una nueva Core, independientemente que sean maestros o esclavos.

En el panel de información, desplegando la pestaña del catálogo de IP Cores, se encuentra un conjunto de Cores denominadas “Bus and Bridges”. En este conjunto se encuentran las Cores de todas las interfaces de comunicación incluidas en el estándar CoreConnect. De allí se selecciona la Core del bus FSL y se arrastra hacia la ventana de estructura del sistema, para que quede conectada como un nuevo dispositivo del proyecto actual.

Después de haber agregado los buses FSL requeridos, en la ventana de estructura del sistema se despliega la pestaña de configuración de buses, en donde se muestran todas las interfaces de comunicación conectadas al sistema.

Es necesario asignar a cada puerto FSL del MicroBlaze y a los puertos de la Core que se ha diseñado (hola_mundo), uno de los canales FSL agregados. Para realizar esta asignación se debe tener en cuenta la conexión punto a punto (maestro - esclavo) que utiliza el bus FSL y que se había mencionado anteriormente (*Fig. 6.23)*.

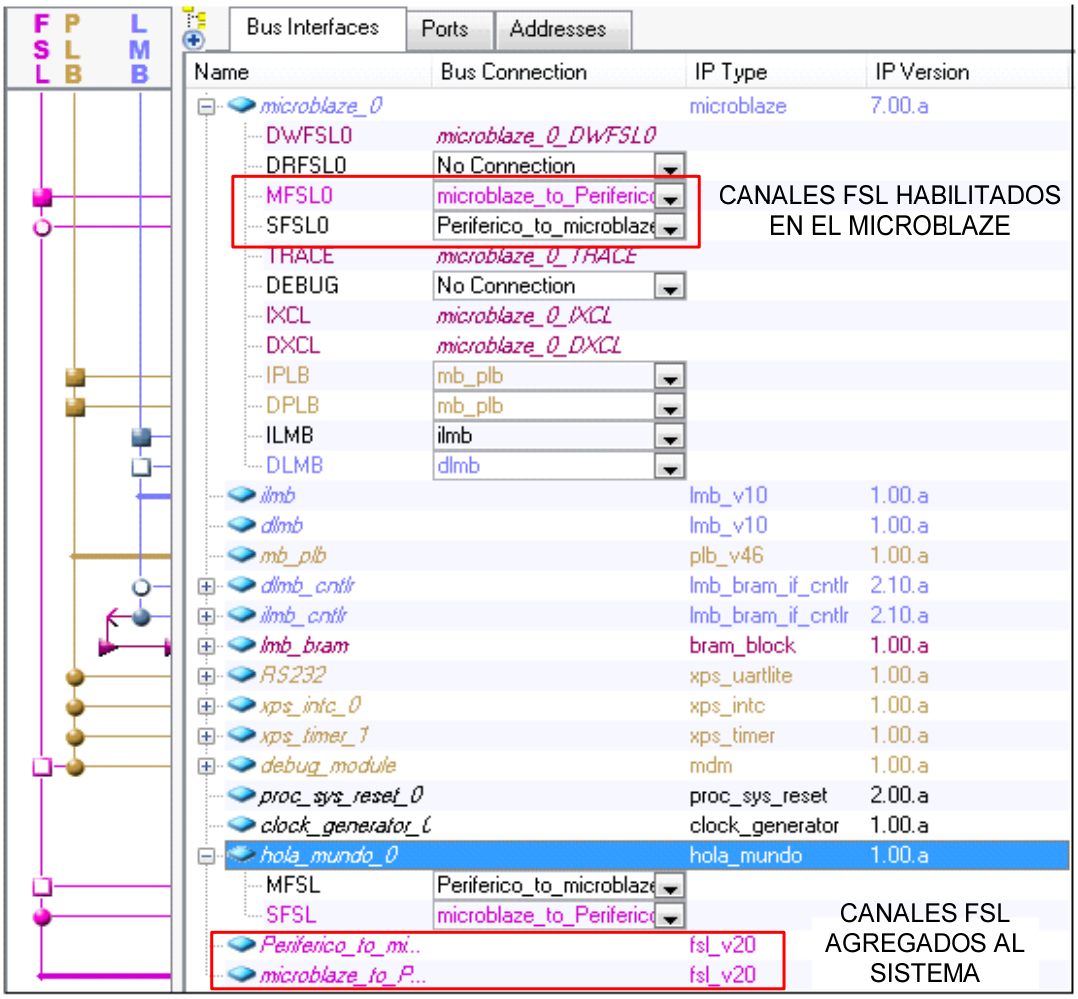

Figura 6.33 *Conexión de Buses FSL.*

Para finalizar la configuración del bus FSL, en la ventana de estructura del sistema se despliega la pestaña de configuración de puertos. Seleccionando la Core del dispositivo creado (hola_mundo), se visualizarán todos los puertos de comunicación habilitados.

Los puertos correspondientes a los canales FSL aparecerán conectados por defecto, pues en los archivos mpd y mhs generados durante la creación del dispositivo, ya se habían establecido estas conexiones. Tan sólo será necesario conectar las señales de sincronismo (reloj y reset) que gobiernan el sistema (*Fig. 6.34)*.

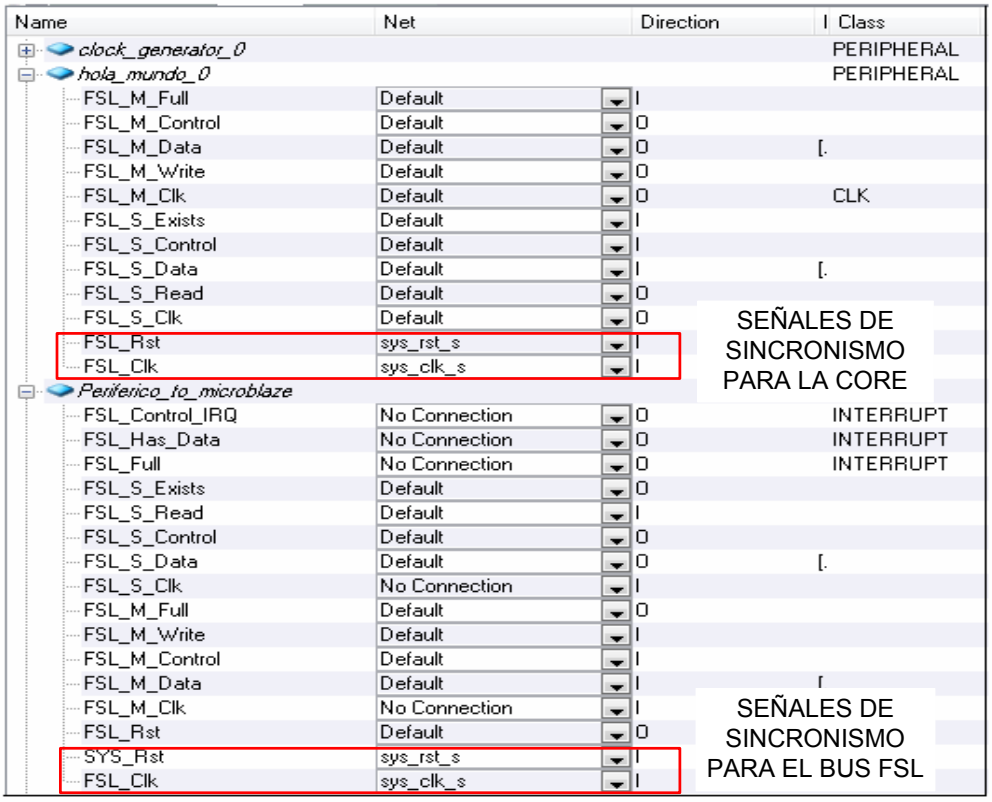

Figura 6.34 *Configuración de Puertos FSL.*

6.4 COMPILACIÓN DEL PROYECTO DE HARDWARE Y GENERACIÓN DEL BITSTREAM

Después de concluir la configuración del MicroBlaze y habiendo definido los parámetros de funcionamiento de las IP Core y sus respectivos buses de comunicación, es posible realizar la compilación del proyecto de hardware.

Para iniciar la compilación se elige en el menú del XPS la opción de crear el archivo de implementación de hardware o bitstream. El proceso de compilación puede tardar varios minutos dependiendo de la velocidad del computador en que se lleve a cabo, por esta razón es recomendable realizar toda la configuración del proyecto de hardware antes de la compilación, pues de esta manera se evita el gasto innecesario de tiempo en sucesivas etapas de compilación.

Una vez compilado el proyecto de hardware y habiendo creado el archivo bitstream, si no se efectúa ningún cambio en el proyecto de hardware, se guardarán los datos de la compilación y no se tendrá que ejecutar de nuevo cuando se esté realizando la compilación del software.

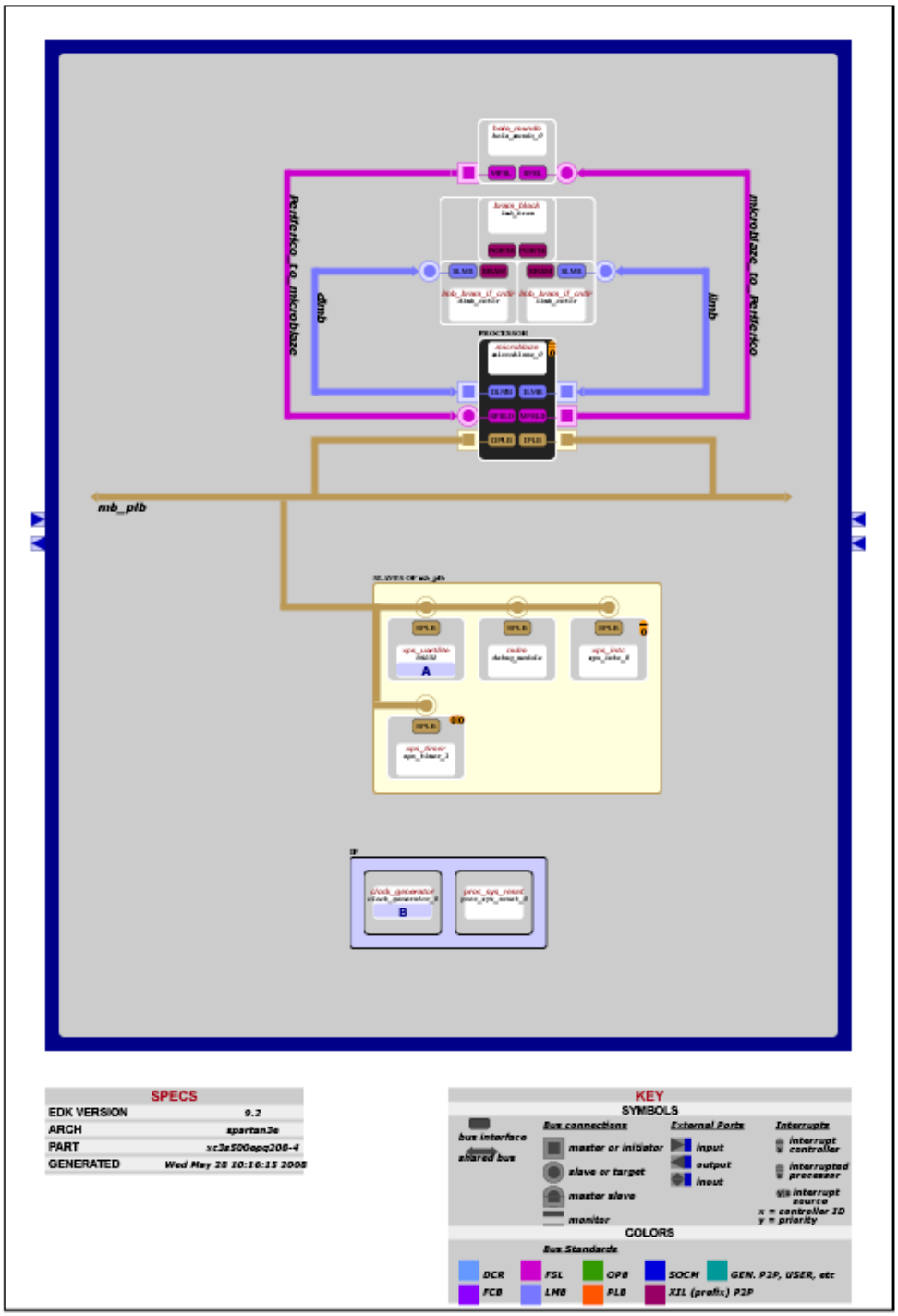

Figura 6.35 *Diagrama de Bloques del Sistema Implementado.*

De esta manera las modificaciones que se realicen en el proyecto de software no requerirán de una nueva compilación del hardware, es decir que el proceso de compilación para cada uno de los proyectos es independiente.

Si la compilación del hardware se ha desarrollado satisfactoriamente, se puede generar el diagrama de bloques del sistema, en el que se especifican las IP Core agregadas al proyecto y sus respectivos buses de comunicación como se ve en la *Fig. 6.35*.

6.5 CONSTRUCCIÓN DEL PROYECTO DE SOFTWARE

A continuación se presentan de manera general, los pasos a seguir para elaborar un proyecto de software sobre el MicroBlaze. Para conocer con más profundidad este proceso, revisar [Pulido, 2007] adjunto a este proyecto.

6.5.1 Configuración de los Parámetros Software del MicroBlaze

El primer paso en la construcción de una nueva aplicación software para un sistema basado en el MicroBlaze, es la configuración de los parámetros de software. En el menú del XPS, se selecciona la opción denominada Software Platform Settings.

Se abrirá una nueva ventana con cuatro opciones en la parte izquierda como se ve en la *Fig. 6.36*. La primera opción, Software Platform, permite modificar algunos parámetros del diseño, así como qué librerías se quieren incluir al proyecto (xilnet, xilmfs, xilfile) y si se quiere utilizar sistema operativo o no (xilkernel, standalone).

Será necesario modificar el parámetro CORE_CLOCK_FREQ_HZ para ajustarlo al valor de la frecuencia de reloj a la cual operará el sistema. en este caso se utilizará una fuente de reloj externa, así que el valor ingresado debe coincidir con la frecuencia de operación de dicha fuente. También se debe seleccionar en el parámetro xmdstub_peripheral la opción RS232. De esta manera se indica al sistema, que para depurar el proyecto se utilizará el puerto serie.

En la siguiente opción, OS and Libraries, se pueden modificar los parámetros del sistema operativo que se vaya a utilizar. Para el caso, no se utilizará ningún sistema operativo, así que sólo se debe indicar que periférico se quiere utilizar para la entrada y salida estándar (stdin y stdout), en este caso será el puerto RS232 .

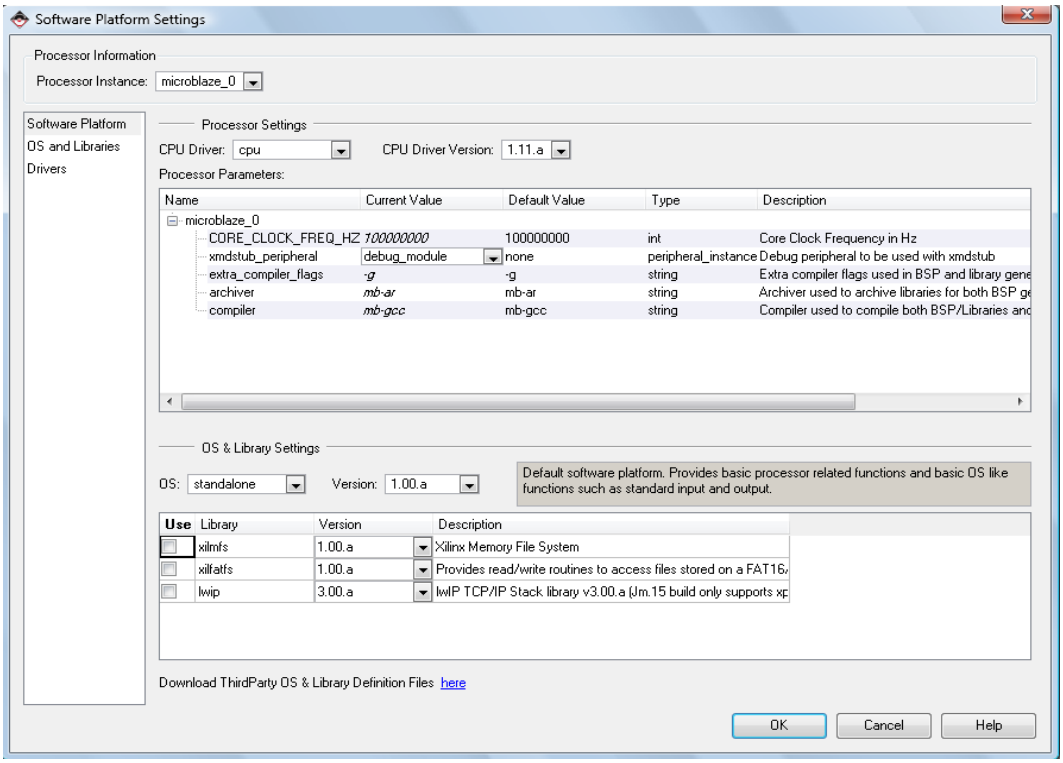

Figura 6.36 *Configuración de los Parámetros Software del MicroBlaze.*

En la opción de Drivers no es necesario realizar ninguna modificación, ya que se usarán los controladores por defecto que proporciona el XPS. Es muy importante conocer que versión de los controladores se ha utilizado en el proyecto, pues es muy frecuente encontrar conflicto entre versiones, dependiendo de la versión del XPS que se utilice para compilar el proyecto.

6.5.2 Creación de una Aplicación Software

Para crear una nueva aplicación de software, se elige la opción de agregar un nuevo proyecto de software, que aparece en la pestaña de aplicaciones del panel de información (*Fig. 6.37*).

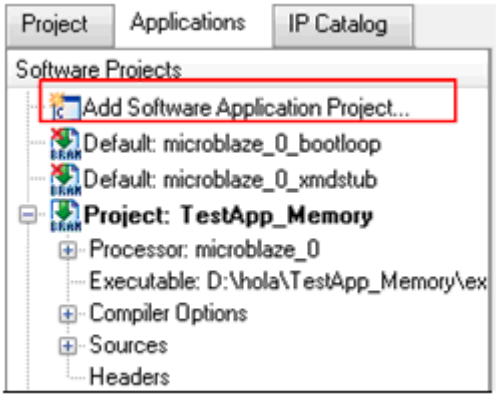

Figura 6.37 *Ventana para Agregar un Nuevo Proyecto de Software.*

Una vez hecho esto, aparecerá una ventana en donde se debe especificar el nombre de la nueva aplicación de software que se va agregar. Al mismo tiempo se debe seleccionar el procesador sobre el cual se desea que se ejecute la aplicación, esto en el caso de haber agregado más de un procesador MicroBlaze en un mismo proyecto de hardware (*Fig. 6.38*).

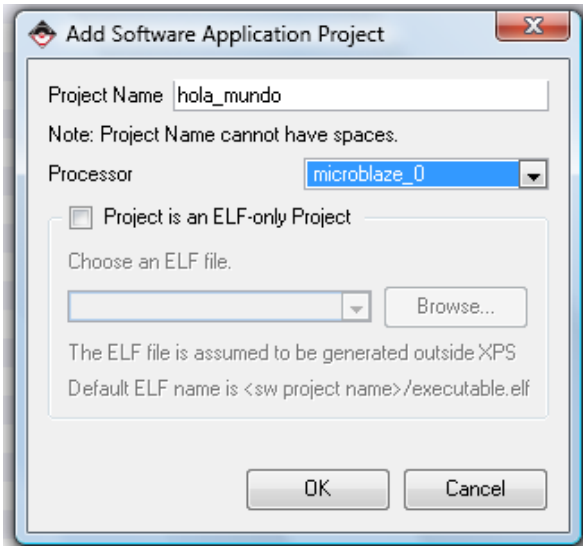

Figura 6.38 *Identificación de la Nueva Aplicación*

De esta manera se ha generado la estructura básica del proyecto de software, pero ahora es necesario agregar las fuentes de código en las que se programarán las distintas funciones que desempeñará el sistema.

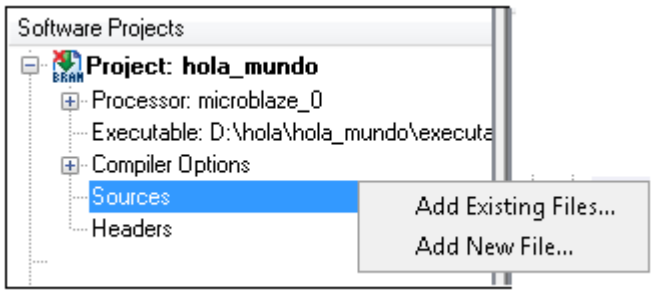

Figura 6.39 *Adición de Códigos Fuente al Proyecto de Software*

Existen dos tipos de archivos fuente: sources y headers (*Fig. 6.39*).

- Sources: archivos de extensión .c, que contiene la definición de las funciones y el programa que se ejecutará en el MicroBlaze.

- Headers: archivos de extensión .h, en donde se realiza la declaración de las funciones que serán utilizadas en el programa.

A continuación se presenta un sencillo ejemplo, en el que se ha programado la escritura de un mensaje a través del puerto serie:

```
1
2   #include "xutil.h"
3   #include "xparameters.h"
4   #include "stdio.h"
5   #include "xbasic_types.h"
6   #include "xgpio.h"
7   #include "math.h"
8
9
10
11  int main (void) {
12
13  xil_printf ("Este es un programa de prueba \r\n");
14
15  xil_printf ("Hola Mundo \r\n");
16
17  }
18
```

Figura 6.40 *Programa de Prueba para el MicroBlaze.*

Como se puede observar en la *Fig. 6.40,* se utiliza lenguaje µclinux para programar el dispositivo. Este lenguaje es muy similar al lenguaje c estándar, pero algunas funciones

han sido optimizadas para tener un mejor funcionamiento sobre procesadores de bajo nivel. Por ejemplo, la función utilizada para imprimir caracteres por el puerto serie denominada printf en el lenguaje c estándar, en µclinux es identificada como xil_printf.

En la *Fig. 6.41,* se observa el resultado de la ejecución del programa de prueba descrito anteriormente, en donde los caracteres generados por la FPGA son leídos en un computador a través del hyperterminal.

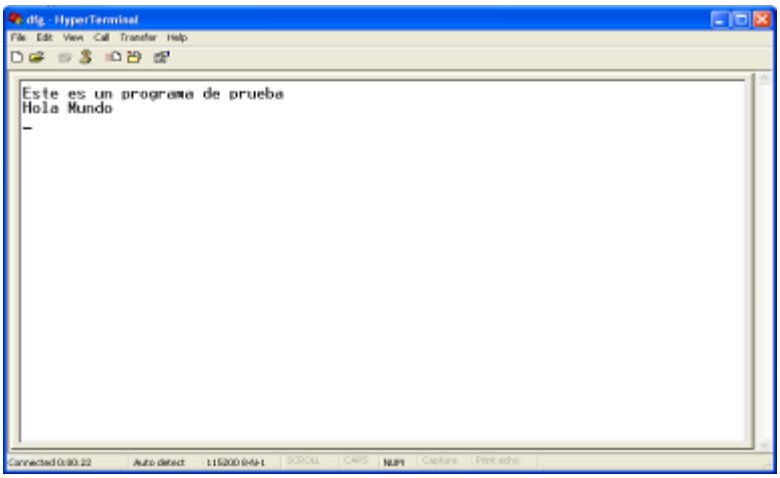

Figura 6.41 *Ejecución del Programa de Prueba*

7 CONSTRUCCIÓN DE HARDWARE EMBEBIDO PARA LA TARJETA EOS

En este capitulo se presentan los aspectos más importantes en el desarrollo de nuevos periféricos para la tarjeta EOS. Con esto se pretende aplicar de manera práctica, los conceptos de diseño de hardware embebido vistos en los capítulos anteriores.

7.1 DESCRIPCIÓN DE LA TARJETA EOS

La tarjeta EOS puede definirse como un dispositivo electrónico para la adquisición de datos y el control de interfaces robóticas que requieran de una alta velocidad de respuesta y flexibilidad para la optimización de los modelos de control allí implementados.

Esta tarjeta, fue desarrollada en su totalidad por el CEIT y surgió como una solución a los inconvenientes tanto operativos como económicos que representaba la utilización de tarjetas de tipo comercial en el desarrollo de aplicaciones con dispositivos hápticos.

La EOS está basada en una FPGA Spartan XC3S500E de Xilinx, en la cual se implementa el procesador embebido MicroBlaze, así como también los controladores para los periféricos con que cuenta la tarjeta.

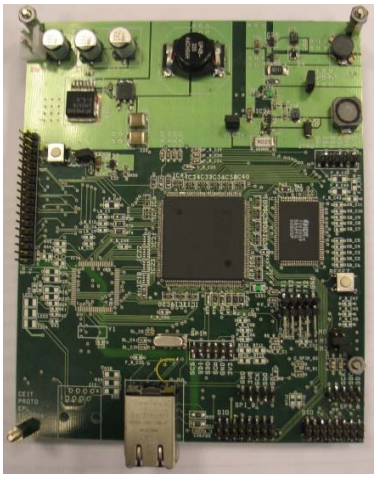

Figura 7.1 *Tarjeta EOS Desarrollada por el CEIT*

Como ya se mencionó, la EOS posee un conjunto de periféricos o módulos externos que la hacen más completa que el kit de desarrollo proporcionado por Xilinx, ajustándose mejor a las necesidades que existen en el Departamento de Mecánica Aplicada del CEIT.

Los periféricos con que cuenta la tarjeta EOS son:

- 6 entradas de encoder diferenciales.
- 8 entradas/salidas digitales por defecto
- Módulos de 48 entradas/salidas digitales.
- Módulos de 2 salidas analógicas ±5V.
- Módulos de 6 salidas analógicas ±10V.
- Módulos de 8 entradas analógicas ±10V.
- Ethernet.
- Puerto serie RS232.

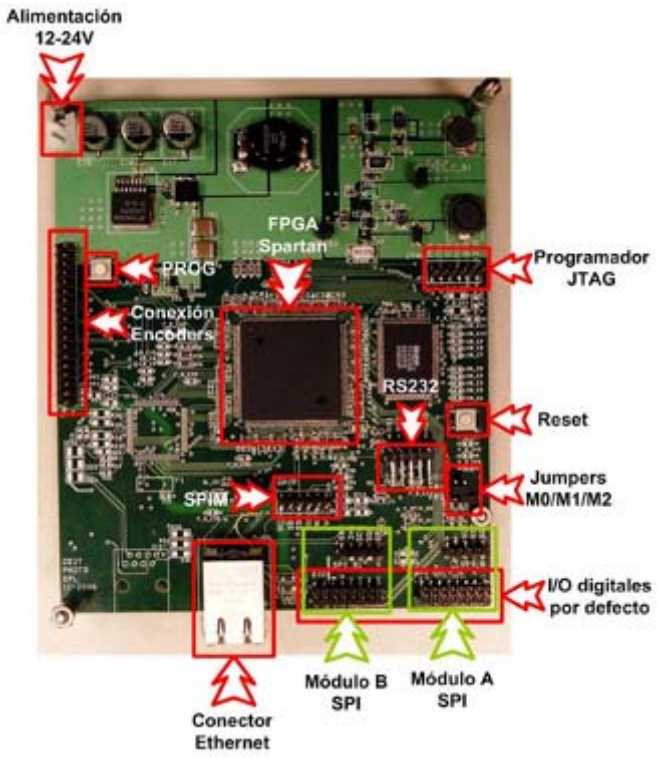

Figura 7.2 *Periféricos de la Tarjeta EOS*

Para conocer a profundidad las características que posee la tarjeta EOS remitirse a [Pulido, 2007], en donde se describen cada uno de los periféricos de la tarjeta.

7.2 DESARROLLO DE HARDWARE EMBEBIDO PARA LA TARJETA EOS

Una de las principales ventajas de utilizar tecnología de hardware reconfigurable (FPGA), es la flexibilidad que se dispone para realizar cambios y mejoras al sistema desarrollado, sin que ello implique la introducción de variaciones en la estructura física del dispositivo.

Como ya se había mencionado, la tarjeta EOS es un dispositivo de desarrollo propio del CEIT y que se encuentra aún en fase de evolución y mejora. De esta forma y como parte de la optimización del hardware embebido de la tarjeta EOS, se encontró la necesidad de crear una nueva IP Core para el cálculo de funciones trigonométricas (seno, coseno), con el propósito de incrementar el rendimiento en la estimación de los parámetros cinemáticos, necesarios en el control de dispositivos hápticos.

7.2.1 Diseño del Modelo de una IP Core para la Tarjeta EOS

El primer paso en el proceso de diseño de una nueva IP Core, es la elección del editor de código sintetizable que se usará para elaborar el modelo de hardware.

Como se vio en el apartado 5.1 existen diferentes opciones para generar código sintetizable. En este caso se ha optado por utilizar el CoreGenerator y así implementar un modelo prediseñado adaptable a los requerimientos del sistema planteado.

El CoreGenerator posee en su catálogo de IP Cores prediseñadas, un módulo para el cálculo de funciones trigonométricas (seno y coseno), que puede configurarse para ser utilizado como coprocesador aritmético, mejorando así el rendimiento del MicroBlaze (*Fig. 7.3*).

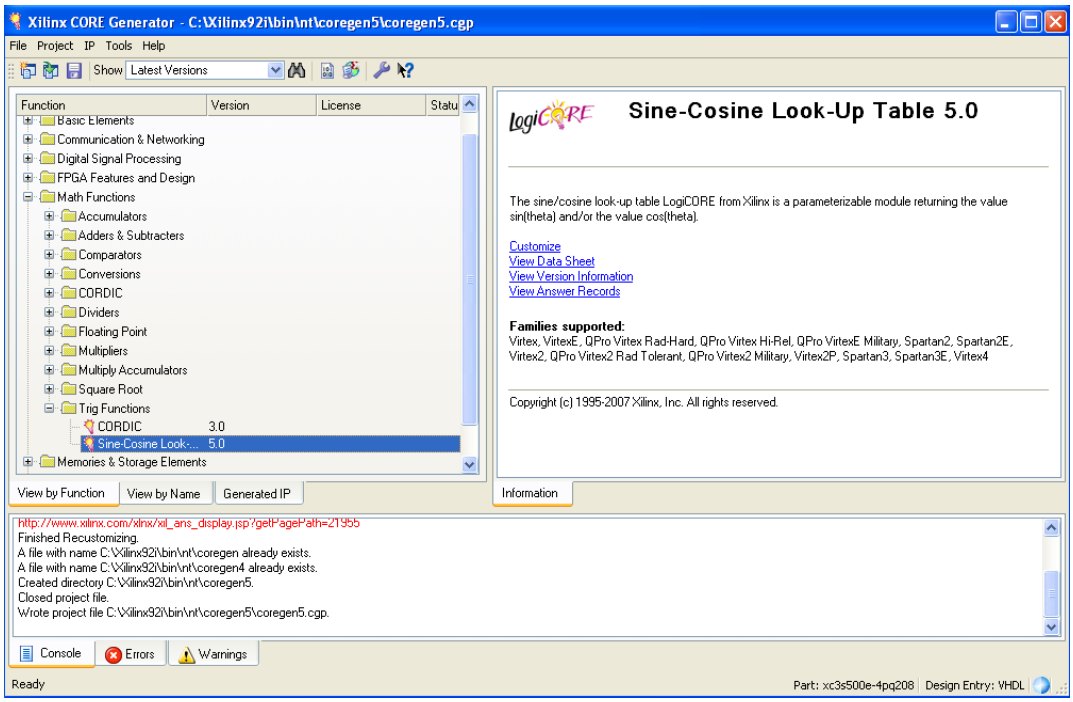

Figura 7.3 *Generación del Modelo en el CoreGenerator*

Una vez elegida la IP Core que se va a implementar, se inicia el proceso de configuración de los parámetros de funcionamiento del nuevo dispositivo.

En primera instancia aparecerá una ventana en donde se define el nombre con el cual se identificará la Core generada (*Fig. 7.4)*. Se debe tener en cuenta que el nombre asignado será al mismo tiempo, el nombre de la entidad del dispositivo, con el cual se efectuará el instanciamiento desde módulos de mayor jerarquía.

Del mismo modo en esta ventana es necesario definir la longitud de las tramas de datos que se manejarán tanto en la entrada como en las salidas del dispositivo. En este caso se utilizarán los valores máximos permitidos tanto para las entradas como para las salidas, es decir, 10 y 32 bits respectivamente.

Se debe aclarar que la nueva IP Core, estará conectada al MicroBlaze a través del bus FSL; este bus utiliza por defecto canales de 32 bits, haciendo necesario ajustar los parámetros de lectura de datos en las entradas de la Core, para obtener la información enviada en los 10 bits especificados anteriormente.

Además será necesario especificar en esta ventana, el tipo de función que se va a implementar, para el caso se elegirán las funciones seno y coseno.

Después se debe precisar la configuración de signos para las funciones calculadas (se dejarán los valores por defecto) y el tipo de memoria empleado (Bloque RAM, Memoria

Distribuida) para almacenar la tabla de valores precalculados (LUT) que será incluida en el componente. Para el caso se seleccionará la memoria distribuida, pues aunque se reduce la resolución de cálculo, el consumo de espacio en memoria también es menor.

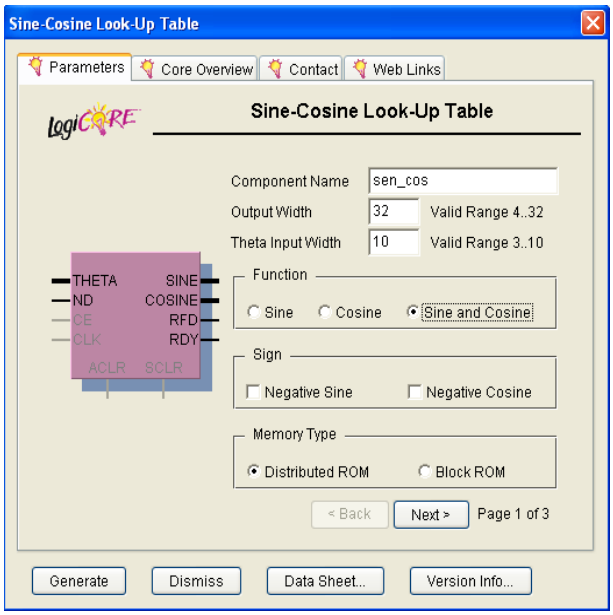

Figura 7.4 *Definición de Parámetros para la IP Core.*

Debido a que las salidas pueden obtenerse con signo positivo o negativo, la representación de estas se realiza en complemento a dos, siendo necesario especificar el tipo de simetría empleado para la conversión a dicho formato. En este caso se ha seleccionado la salida no simétrica como se ve en la *Fig. 7.5.*

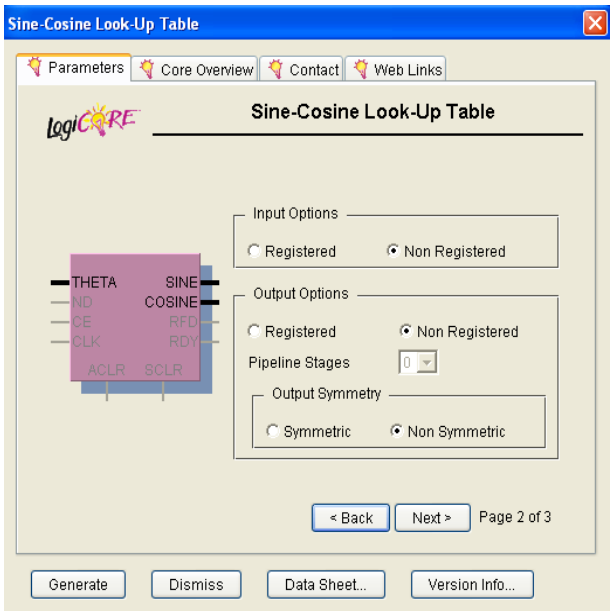

Figura 7.5 *Configuración del Sincronismo de las Entradas y Salidas.*

Finalmente para completar la configuración de la IP Core, es necesario definir el protocolo de acceso (handshaking) utilizado para interactuar con el dispositivo.

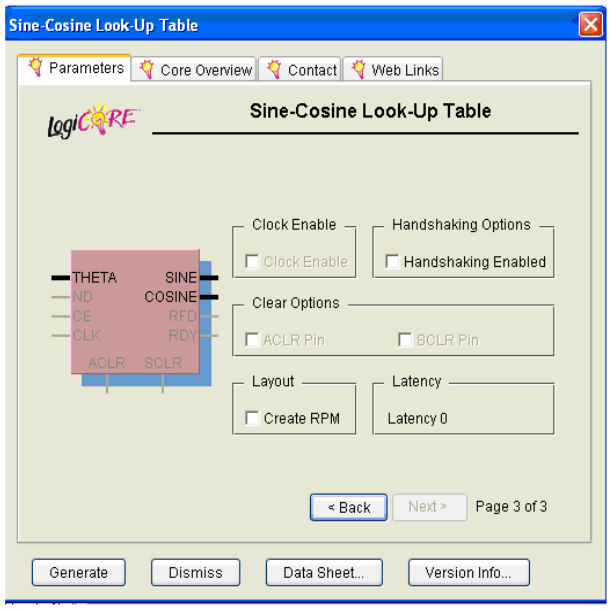

Figura 7.6 *Configuración del Protocolo de Acceso.*

En esta aplicación no se utilizará el protocolo de acceso, pues el componente responderá automáticamente a los valores presentes en la entrada de datos (*Fig. 7.6)*. Las demás opciones presentes en esta ventana no se modificarán utilizando así los valores por defecto asignados por el sistema.

7.2.3 Creación del Nuevo Periférico

Una vez generada la IP Core para el cálculo de seno y coseno, es necesario crear un módulo superior en el cual se implemente el protocolo de comunicación con el bus FSL y además se realice el instanciamiento de la IP Core previamente desarrollada. Este módulo IP Core en adelante se mencionará como IPSC (**IP C**ore **S**eno y **C**oseno).

El primer paso es crear la plantilla para un nuevo periférico, utilizando el asistente de configuración que posee el XPS y que anteriormente se presentó en el apartado 6.3.1.

En el asistente de configuración se debe especificar la interfaz de comunicación que se empleará para conectar la IPSC con el MicroBlaze; en este caso será el bus FSL como ya se había mencionado.

En la definición de parámetros del bus FSL que se lleva a cabo en el asistente de configuración, se establece el número de canales FSL que se habilitarán. Para la IPSC será necesario tener dos canales FSL habilitados: un canal correspondiente a la entrada de datos provenientes del puerto salida (maestro) del MicroBlaze, y otro canal que será compartido por las salidas de la IPSC y que se conecta al puerto de entrada esclavo del MicroBlaze *Fig. 7.7.*

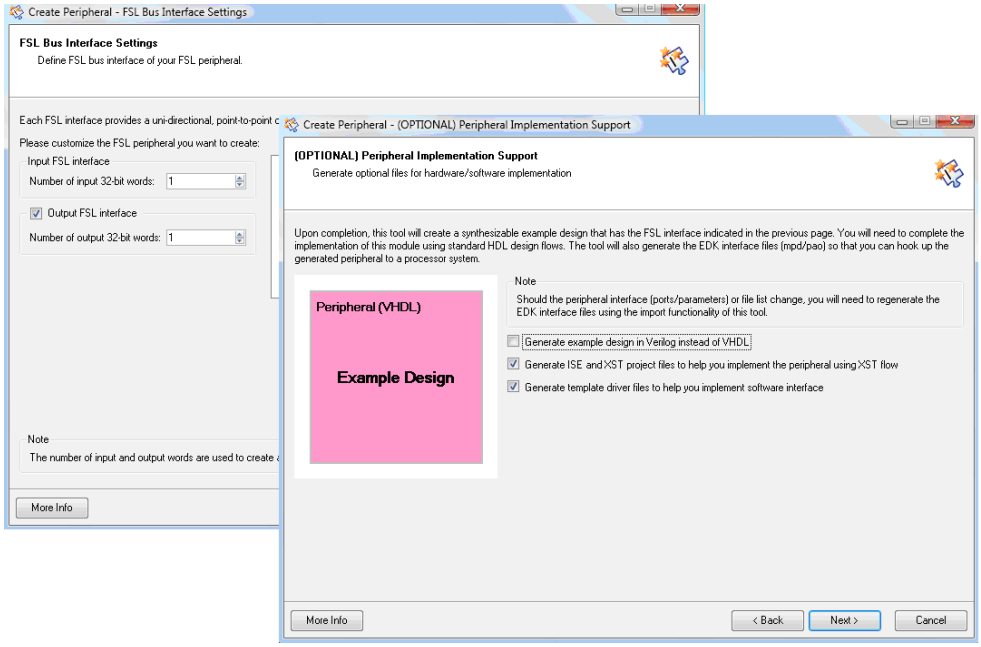

Figura 7.7 *Adición de Canales FSL.*

En la ventana de soporte para la implementación del periférico, se seleccionan las opciones que permiten generar una plantilla general para la implementación del protocolo de comunicación a través del bus FSL como se ve en la *Fig. 7.7.*

Del mismo modo se selecciona la opción para generar plantillas de ejemplo, que se tomarán como guía para construir los controladores del dispositivo.

7.2.4 Descripción del Código VHDL de la IP Core Implementada

Como ya se había mencionado en el apartado 5.3, un módulo de hardware descrito en lenguaje VHDL consta de dos partes fundamentales: la entidad y la arquitectura.

De esta forma el primer paso para iniciar la construcción del módulo para el cálculo de seno y coseno (IPSC), es la definición de puertos (entrada - salida) en la entidad del componente.

La IPSC no utiliza pines físicos de la FPGA, pues está conectada directamente con el MicroBlaze a través de canales FSL. Por consiguiente en la definición de puertos sólo se tendrán como parámetros de configuración, las señales de datos y control de los canales FSL que han sido definidas previamente por el asistente para la configuración de nuevos periféricos, como se vio en el apartado anterior.

Esta sección de código no deberá ser editada, pues el asistente de configuración asigna los valores estándar de funcionamiento, para los puertos del protocolo FSL.

```
entity seno_coseno is

port
   (
      -- DO NOT EDIT BELOW THIS LINE ---------------------
      -- Bus protocol ports, do not add or delete.
      FSL_Clk          : in  std_logic;
      FSL_Rst          : in  std_logic;
      FSL_S_Clk        : out std_logic;
      FSL_S_Read       : out std_logic;
      FSL_S_Data       : in  std_logic_vector(31 downto 0 );
      FSL_S_Control    : in  std_logic;
      FSL_S_Exists     : in  std_logic;
      FSL_M_Clk        : out std_logic;
      FSL_M_Write      : out std_logic;
      FSL_M_Data       : out std_logic_vector(31 downto 0 );
      FSL_M_Control    : out std_logic;
      FSL_M_Full       : in  std_logic
      -- DO NOT EDIT ABOVE THIS LINE ---------------------
   );
```

Una vez establecida la entidad de la IPSC, es necesario construir la arquitectura del dispositivo.

La arquitectura está dividida en dos partes: en la primera, se realiza el instanciamiento de los componentes que serán utilizados en la estructura del componente, siendo en este caso, el módulo obtenido a partir del CoreGenerator como se vio en el apartado 7.2.1.

Además en esta primera parte, se realiza la definición de señales y constantes, que serán utilizadas dentro de la estructura del programa general. Como ya se había mencionado, la IPSC no utilizará pines externos de la FPGA, por esta razón las variables de datos utilizadas por la IPSC deberán ser definidas como señales.

```
architecture Behavioral of seno_coseno  is

component sen_cos
   port (
   THETA: IN std_logic_VECTOR(9 downto 0);
   SINE: OUT std_logic_VECTOR(31 downto 0);
   COSINE: OUT std_logic_VECTOR(31 downto 0));
end component;

   -- Total number of input data.
   constant NUMBER_OF_INPUT_WORDS  : natural := 1;

   -- Seno_coseno FSL states
   type STATE_TYPE is (Idle, Read_Inputs, Write_Outputs);

   signal state        : STATE_TYPE;

   -- Data to send to the master FSL
   signal data          :  std_logic_vector (31 downto 0);
   signal data2          :  std_logic_vector (31 downto 0);

   -- Counters to store the number inputs read & outputs written

   signal nr_of_writes : natural range 0 to 1;

   -- Señales de datos
   signal THETA        : std_logic_vector (9 downto 0 );
   signal SINE         : std_logic_vector (31 downto 0 );
   signal COSINE       : std_logic_vector (31 downto 0 );
```

En la segunda parte de la arquitectura se realiza el mapeo de los componentes instanciados y además se definen cada uno de los estados de operación que tendrá la IPSC.

```
begin

   U1 : sen_cos
      port map (
         THETA => THETA,
         SINE => SINE,
         COSINE => COSINE);

   FSL_S_Read  <= FSL_S_Exists   when state = Read_Inputs   else '0';
   FSL_M_Write <= not FSL_M_Full when state = Write_Outputs else '0';
   FSL_M_Data  <= data;
```

El proceso que se lleva a cabo en la arquitectura se encuentra sincronizado con el reloj del sistema, es decir, cada vez que se reciba un flanco ascendente de reloj se entrará en un nuevo ciclo de operación.

Se han definido tres estados de operación para el dispositivo: Modo de Espera (Idle), Modo de Lectura (Read_Inputs) y Modo de Escritura (Write_Outputs).

Cada vez que se inicie el sistema se igualarán a cero las señales de entrada y salida de datos y además se activará el modo de espera.

```
The_accelerator : process (FSL_Clk) is
begin  -- process The_SW_accelerator

 if FSL_Clk'event and FSL_Clk = '1' then      -- Rising clock edge
   if FSL_Rst = '1' then                  -- Synchronous reset (active high)
     -- CAUTION: make sure your reset polarity is consistent with the
     -- system reset polarity
     state        <= Idle;
     data         <= (others => '0');

     THETA        <= (others => '0');
```

- Modo de Espera: cada vez que se inicie un nuevo ciclo de operación y esté activo el modo de espera, se verificará la existencia de un nuevo dato en el bus de entrada esclavo FSL. Si existe un nuevo dato en el bus, el bit correspondiente al puerto esclavo (FSL_S_Exists) tomará el valor '1' y el sistema cambiará al estado de lectura de entradas, de lo contrario, finalizará el proceso en el ciclo de operación actual.

```
     else
       case state is
         when Idle =>
           if (FSL_S_Exists = '1') then
             state       <= Read_Inputs;

             data        <= (others => '0');

           end if;
```

- Modo de Lectura: cuando se inicia un ciclo de operación en el modo de lectura, se verifica el tipo de dato que se ha recibido.

  El protocolo FSL distingue entre dos tipos de datos: datos de control y datos generales. De esta forma si el dato recibido es de control, el bit correspondiente (FSL_S_Control) tomará el valor '1' y se procede a asignar como valores de entrada (THETA), los 10 primeros bits del bus FSL esclavo. Esta asignación se

debe realizar, pues en la configuración del CoreGenerator se especificó que se tendría una resolución de 10 bits a la entrada de la Core (ver apartado 7.2.1).

Una vez leídos los datos del bus FSL el sistema volverá al estado de espera, finalizando así el ciclo de operación actual.

Si el dato leído es de tipo general, el bit FSL_S_Control tomará el valor '0' y se activará el modo de escritura. Al mismo tiempo se cargará en la salida (FSL_M_Data) el valor correspondiente al coseno del ángulo THETA.

Como se puede observar, se está escribiendo un dato antes de haber iniciado un nuevo ciclo en el modo de escritura, esto se hace con el fin de aprovechar el ciclo de operación actual, ahorrando tiempo en la ejecución global del proceso.

```
when Read_Inputs =>
  if (FSL_S_Exists = '1') then
    if ( FSL_S_Control = '1') then
      |      THETA <= std_logic_vector(  unsigned(FSL_S_Data (9 downto 0)));
             state <= Idle;
    else
    data <=COSINE;
    state        <= Write_Outputs;
        nr_of_writes <= 1;
    end if;
end if;
```

- Modo de Escritura: cuando se inicia un nuevo ciclo de operación en el modo de escritura se verifica el número de datos que faltan por escribir en el bus FSL. Si no falta ningún dato por escribir, se activará el modo de espera finalizando la operación del ciclo actual.

  Por el contrario si hace falta algún dato por escribir, se verificará que el bus de escritura (FSL_M_Data) esté disponible para añadir un nuevo bloque de datos. Si el bit FSL_M_Full posee el valor '0' el bus está disponible y se asignará un nuevo valor a la salida.

  Si el bit FSL_M_Full está en '1', el bus de escritura no está disponible para enviar un nuevo dato. De esta manera se finaliza el ciclo de operación actual para esperar que el bus esté disponible y así enviar el dato que se ha calculado.

  Una vez enviados los datos del seno y coseno, se volverá al estado de espera, iniciando de esta manera un nuevo ciclo de la máquina de estados implementada en el sistema.

```
          when Write_Outputs =>
            if (nr_of_writes = 0) then
              state <= Idle;
            else
              if (FSL_M_Full = '0') then
                    data <=SINE;

                nr_of_writes <= nr_of_writes - 1;

              end if;
            end if;


         end case;
       end if;
     end if;
   end process The_accelerator;



end Behavioral;
```

7.2.5 Adición de la Nueva IP Core a la Tarjeta EOS

Para agregar la IPSC a la estructura del proyecto de hardware de la tarjeta EOS es necesario ejecutar el asistente para configuración de nuevos periféricos. Este proceso se debe llevar a cabo como se describió en el apartado 6.3.2.

De esta manera se le indicará al asistente, que se va agregar un componente de hardware ya existente (*Fig. 7.8*). Además será necesario especificar el tipo de archivos fuente que se utilizarán en la implementación del componente, en este caso se agregarán las fuentes VHDL diseñadas y también el fichero netlist (ngc) que se obtuvo en el CoreGenerator cuando se creó el módulo de cálculo para seno y coseno.

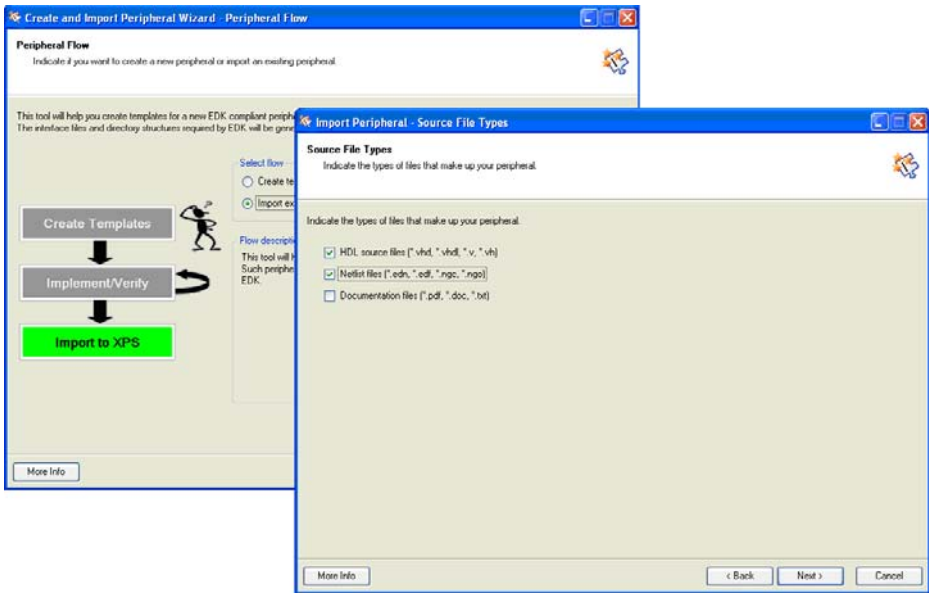

Figura 7.8 *Inicio del Asistente para Configuración de Periféricos.*

Una vez especificado el tipo de códigos fuente que se utilizarán para describir el componente, es necesario adjuntar al proyecto los ficheros especificados. No se debe olvidar que la adición de los códigos fuente debe realizarse en orden jerárquico, pues esta es la manera en que el XPS interpreta el código fuente adjuntado (*Fig. 7.9*).

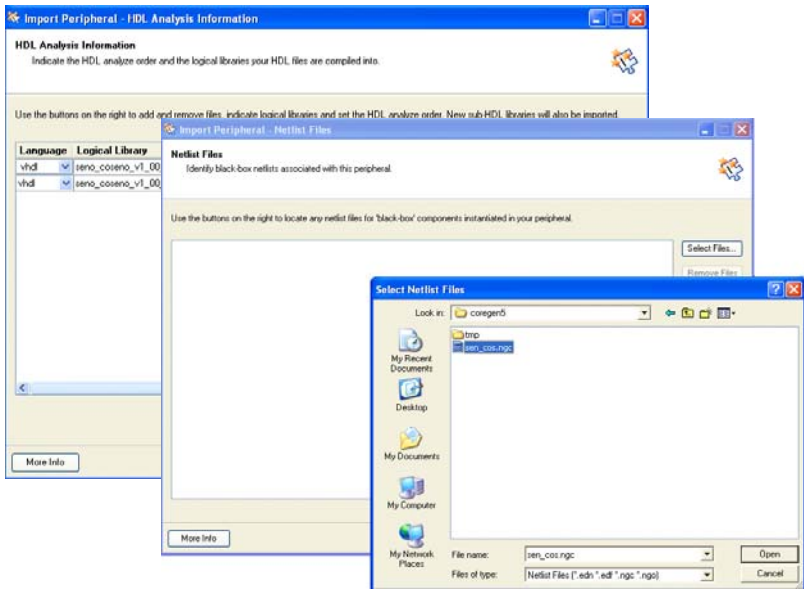

Figura 7.9 *Adición de Códigos Fuente y Archivos Netlist.*

Si los ficheros VHDL han sido agregados satisfactoriamente, el asistente presentará una ventana de informe en donde se especifican los puertos encontrados y el mapeo que se ha efectuado en el módulo principal.

Después será necesario adjuntar el archivo netlist (ngc), generado por el CoreGenerator y que contiene la descripción hardware de la Core utilizada para implementar el módulo de cálculo de seno y coseno.

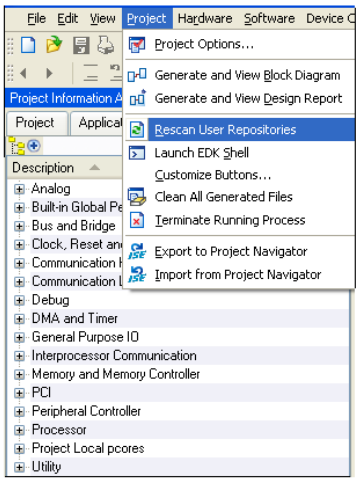

Figura 7.10 *Actualización de los Repositorios del Proyecto.*

Una vez finalizado el asistente para configuración de nuevos periféricos, se debe actualizar la carpeta de repositorios del proyecto (*Fig. 7.10*), para que la IPSC aparezca en el catalogo de IP Cores disponibles.

Para agregar la IPSC a la estructura del proyecto, tan sólo será necesario seleccionarla en el listado de IP Cores que aparece en el panel de información y arrastrarla hasta la ventana de estructura del sistema. Del mismo modo será necesario agregar las IP Core de los dos canales FSL que se usarán para conducir el flujo de datos entre el MicroBlaze y la IPSC (*Fig. 7.11*).

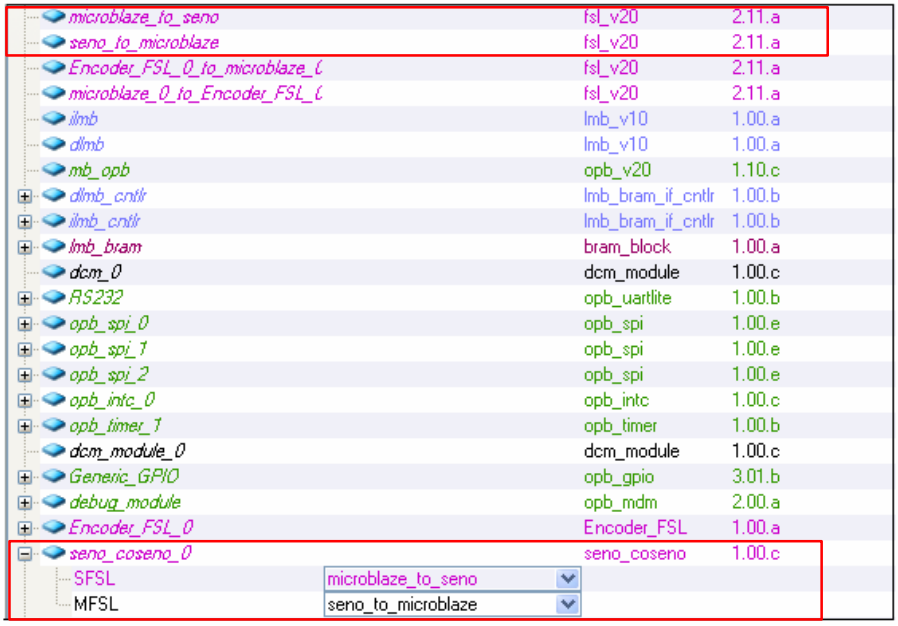

Figura 7.11 *Configuración de de Puertos y Canales FSL.*

Tomando como referencia los pasos descritos en el apartado 6.3.3, es posible llevar a cabo la configuración de los buses FSL, asignando los puertos y las señales de sincronismo necesarias para habilitar la interacción entre el MicroBlaze y la IPSC.

Una vez terminada la configuración de los canales FSL es posible comenzar la compilación del proyecto de hardware. Si este proceso culmina satisfactoriamente, en la ventana de informes se indicará que el archivo bitstream utilizado para programar la FPGA ha sido creado. De esta manera la implementación del proyecto de hardware queda terminada y será necesario desarrollar un proyecto de software para controlar los dispositivos hardware que se han adherido al sistema.

7.3 DESARROLLO DE CONTROLADORES PARA LA TARJETA EOS

Los controladores desarrollados para manejar las distintas aplicaciones en el MicroBlaze, contienen dos tipos de archivos: Header (.h) y Sources (.c).

En el header se realiza la declaración de las funciones que se utilizarán en la función general definida en el archivo source. Para el caso se ha declarado la función ‘cos_sin’, que permite comunicar el MicroBlaze con la IPSC, teniendo como dato de entrada el ángulo en radianes ‘rad’ y además un vector puntero en el que se almacenarán los datos de salida obtenidos (seno y coseno).

```
#include "fsl.h"
#include "xparameters.h"


void cos_sin(float rad, float* out);
```

El archivo source contiene la definición de las funciones y el programa general que se va a ejecutar. En este caso el archivo source se ha dividido en cuatro partes para comprender mejor su funcionamiento: Asignación de Macros, Definición de Variables, Lectura/Escritura de Datos y Conversión de Formato.

- Asignación de Macros: el XPS permite a través de uso de funciones Macro, obtener un fácil acceso a las interfaces FSL del MicroBlaze. Para habilitar el funcionamiento de los macros, es necesario incluir en el programa la librería 'fsl.h' en donde se incluyen dichas funciones.

  En este caso se utilizaron tres funciones macro, para leer y escribir sobre el bus FSL:

  cputfsl (val, id): este macro permite ingresar al bus FSL un bloque datos de control. 'Val' corresponde al valor que se quiere introducir al bus, mientras que 'id' es el identificador del canal FSL utilizado.

  putfsl (val, id): este macro permite ingresar al bus FSL un bloque datos generales. 'Val' corresponde al valor que se quiere introducir al bus, mientras que 'id' es el identificador del canal FSL utilizado.

  getfsl (val, id): este macro permite leer datos del bus FSL. 'Val' corresponde a la variable en que se almacenan los datos leídos, mientras que 'id' es el identificador del canal FSL utilizado.

  En esta instancia también es necesario definir el 'id' de los canales FSL que se utilizarán. Como se había mencionado en el apartado 6.3.3, el protocolo FSL utiliza canales organizados en parejas (maestro - esclavo).

  A cada pareja se le asigna un 'id' numérico, para identificar cual pareja se usará para transmitir en determinado instante. En este caso se utilizaron los canales MFSL1 y SFLS1, por eso es necesario asignar al 'id' el valor de '1'.

```
#include "fsl.h"
#include "xparameters.h"

#define BUS_FSL     1
#define write_c_into_fsl(val, id)  cputfsl(val, id)
#define write_into_fsl(val, id)    putfsl(val, id)
#define read_from_fsl(val,id)      getfsl(val, id)
```

- Definición de Variables: en esta sección se crean las variables que se utilizarán en el programa. En este caso es importante aclarar el valor asignado a la variable 'rad'. Esta variable será la el dato (ángulo) que se envíe a la IPSC; pero es necesario ajustar el formato de este valor, pues la IPSC calcula los valores de salida, a partir de una tabla de valores preasignados (LUT).

  Para el caso el valor de 'rad' se calcula de la siguiente forma:

$$rad = \frac{(\phi * 2^n)}{2\pi}$$

  En donde n corresponde al número de bits definidos para la entrada de la IPSC, asignándose en este caso el valor de 10 como se vio en el apartado 7.2.1. Además es el ángulo en radianes que se utilizará como entrada.

```
void cos_sin(float rad, float* out)
{
int sin_cos[2], theta;
float pi,tmp;
pi= 3.141592f;
rad = (rad*1024.0f)/(2.0f*pi);
theta= rad;
tmp=rad-theta;
theta=0.5f+tmp+theta;
```

- Lectura/Escritura de Datos: en esta sección del programa se ejecutan los macros para la lectura y escritura en el bus FSL. Es necesario utilizar el mismo orden que se definió en el código VHDL (apartado 7.2.4), para leer y escribir en el bus, pues de lo contrario se obtendrán como respuesta valores erróneos.

```
      write_c_into_fsl(theta, BUS_FSL);
      write_into_fsl(1, BUS_FSL);
      read_from_fsl(sin_cos[0], BUS_FSL);
      read_from_fsl(sin_cos[1], BUS_FSL);
```

- Conversión de Formato: los valores obtenidos como respuesta en la IPSC, se encuentran en formato de complemento a dos, siendo necesario convertirlos a formato decimal para poder utilizarlos en el programa general.

  Para realizar la conversión de formato se empleó la siguiente ecuación:

$$C_2{}^N = 2^n - N$$

  En donde n corresponde al número de bits utilizados (32) y N es el número en complemento a dos que se quiere convertir.

  Además de esto es necesario dividir la respuesta en 2^{n-1}, para normalizarla según los parámetros de simetría definidos en el CoreGenerator en el apartado 7.2.1.

```
   if (sin_cos[0] >0x7FFFFFFF){

       out[0]= (float) -( 0x100000000 - sin_cos[0])/2147483648.0f;
       }

  else{
       out[0]=(float) sin_cos[0]/2147483648.0f;

       }

  if (sin_cos[1] >0x7FFFFFFF){

       out[1]=(float) -(0x100000000 - sin_cos[1])/2147483648.0f;
       }
  else{
       out[1]= (float)sin_cos[1]/2147483648.0f;

       }
}
```

8 MODELOS DE CONTACTO

Cuando se produce una colisión se calcula la fuerza normal de contacto entre dos objetos virtuales en función de la penetración máxima que se ha producido entre ambos objetos, conocido como el método de la penalización.

Si en el contacto entre los dos objetos no se produce deformación plástica, la fuerza que aparece entre ambos se puede escribir como suma de dos componentes, una elástica o conservativa y otra viscosa.

Las fuerzas que puede ejercer el entorno contra el usuario pueden ser de dos tipos, normales o tangenciales. Las fuerzas normales son perpendiculares a la superficie de contacto y son las que realmente hacen sentir el objeto. Las fuerzas tangenciales están formadas por el rozamiento y los potenciales. Si estas fuerzas no existiesen se podrían sentir fuerzas, pero daría la sensación de que se desliza sobre hielo.

El método más extendido entre los sistemas hápticos de impedancia para el cálculo de la fuerza entre dos objetos virtuales es aquel que determina dicha fuerza en función de la penetración que se ha producido entre ambos objetos. A este método se le suele llamar método de la penalización.

Si en el contacto entre los dos objetos no se produce deformación plástica, la fuerza que aparece entre ambos se puede escribir como suma de dos componentes, una elástica o conservativa y otra viscosa.

$$f_e = f_{el}(x) + f_v(x, \dot{x})$$

La fuerza de contacto, f_e , posee dos componentes, f_{el} es la parte elástica o conservativa y f_v es la parte viscosa y el valor x es la penetración entre los objetos.

Un problema común de todos los modelos que se describirán es encontrar un método para calcular la penetración x existente entre los dos objetos virtuales. Este problema tiene diferentes soluciones dependiendo de cómo se encuentren caracterizados los objetos virtuales dentro del ordenado.

8.1 MODELO ELÁSTICO

Si los objetos en contacto son metálicos, la componente viscosa es pequeña frente a la elástica y se puede despreciar.

El modelo elástico más extendido propone restituir una fuerza directamente proporcional a la penetración que se ha producido entre los dos objetos. Este modelo lo describe la siguiente ecuación, donde k_e es la rigidez virtual implementada.

$$f_e = k_e x$$

Los objetos virtuales se sentirán más o menos rígidos dependiendo de la rigidez virtual k_e que se implemente *Fig. 8.1*. No es posible implementar de forma estable cualquier valor de rigidez, por lo que las investigaciones al respecto intentan determinar cuál es el valor teórico crítico.

Acerca de la rigidez limitada de los objetos virtuales, [Massie, 1994] afirma que el hombre aprecia tipos de rigidez mayores de 2000 N/m como una superficie completamente rígida. Por su parte, [Tan, 1994] afirman que el umbral absoluto de rigidez, a partir del cual un usuario no es capaz de distinguir el tipo de rigidez, hay que situarlo entre los 15300 N/m y los 41500 N/m.

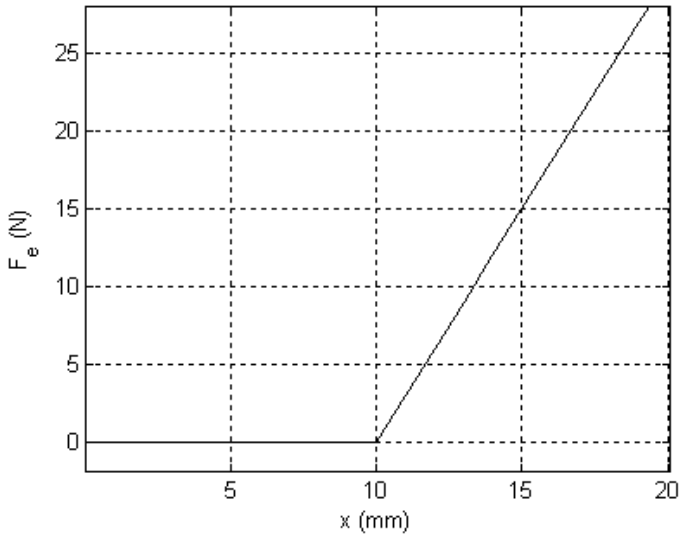

Figura 8.1 *Representación del modelo de fuerza de pared elástica.*

8.1.1 Modelo Elástico Lineal

Consiste en restituir una fuerza directamente proporcional a la penetración que se ha producido entre los dos objetos, siendo la constante de proporcionalidad la rigidez virtual implementada k.

$$f_n = kx$$

No es posible implementar cualquier valor de rigidez de forma estable. Además el modelo solo es valido cuando el objeto que toca el escenario no tiene dimensiones, en caso contrario el modelo es solo aproximado.

8.1.2 Modelo Elástico No Lineal

Hay que tener en cuenta que el modelo anterior sólo es válido cuando el objeto que toca el escenario no tiene dimensiones, es decir, es un punto. Si los dos objetos en contacto tienen dimensiones, el modelo elástico es sólo aproximado.

Parece lógico pensar que la fuerza de contacto debe ser, más bien, proporcional al volumen de intersección entre ambos objetos.

Si los objetos en contacto son esféricos, se ha propuesto un modelo elástico no lineal, mostrado en la siguiente ecuación:

$$f_e = k_e x^{\frac{3}{2}}$$

Este modelo, descrito por ejemplo por [Jhonson, 1985], es coherente con la teoría de Hertz sobre los impactos, publicada en 1882.

Para objetos en contacto de otras formas, e incluso de diversos materiales, se han propuesto otras potencias x^y para ponderar la penetración.

Como, en general, no se conoce la forma de los objetos en colisión tanto el modelo lineal cono el no lineal son aproximados. Si a la mayor simplicidad del modelo lineal se añade el hecho de que muchos dispositivos hápticos, por ejemplo el PHANToM, sólo contemplan la posibilidad de que colisione un punto del usuario, hace que el modelo lineal sea el más empleado hasta la fecha.

8.2 MODELO VISCOELÁSTICO

El modelo viscoelástico lineal añade al modelo elástico lineal una componente viscosa proporcional a la velocidad de aumento de la penetración en el escenario virtual.

Este modelo llamado también Kelvin-Voigt, fue citado en [Goldsmith, 1960] y se presenta a continuación:

$$f_e = k_e x + b_e \dot{x}$$

La constante de amortiguación b añadida beneficia en algunos aspectos la sensación de contacto. Además el amortiguamiento virtual disipa energía y, por tanto, contribuye a la estabilidad del sistema.

El principal inconveniente de este método es la fuerte discontinuidad de la fuerza de contacto con el objeto. La fuerza comienza desde un valor finito en función de de la velocidad inicial con la que la se produce la colisión. Cuando se produce la separación de los objetos aparece una fuerza contraria a la normal creando un fenómeno de adherencia entre objetos.

Este hecho no tiene sentido físico, porque los cuerpos se deben separar sin ningún tipo de oposición a no ser que exista un fenómeno de adherencia entre ambos.

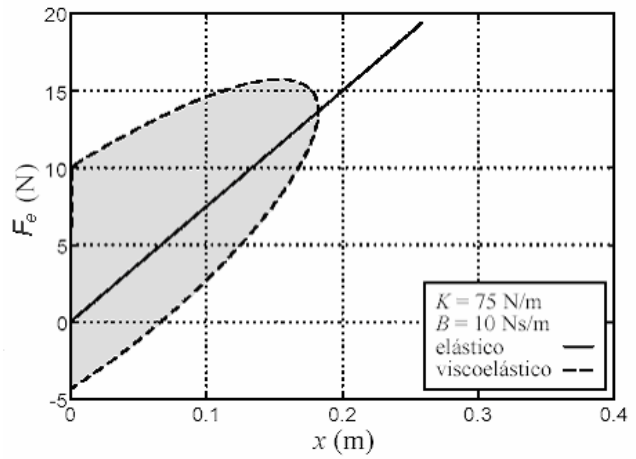

Figura 8.2 *Comparativa de los modelos elástico y viscoelástico* [Gil, J. J, 2003].

La *Fig. 8.2* muestra una comparativa de los modelos elástico y viscoelástico. En ambos casos una pared virtual repele una masa de 5 kg que impacta con una velocidad de 1 m/s. Se representa el módulo de la fuerza en función de la penetración.

Se puede observar la fuerza inicial no nula del modelo viscoelástico. También se observa la aparición de fuerzas negativas, de adherencia o atracción, en los últimos momentos de contacto con la pared viscoelástica.

La energía disipada por el amortiguador del modelo viscoelástico corresponde al área sombreada de la *Fig. 8.2.* En el modelo elástico, el movimiento de penetración y salida restituye fuerzas iguales en cada valor de penetración, la energía que devuelve la pared al objeto es la misma que ha disipado, por lo que el objeto en la simulación rebota con la misma velocidad con la que ha impactado.

Esto no ocurre en el caso de modelo viscoelástico, donde la velocidad de salida es inferior a la de impacto, es decir, su energía cinética ha disminuido.

Otro hecho beneficioso del modelo viscoelástico que se observa en la simulación es que se consigue repeler la masa con menor penetración que con el modelo elástico. Además la fuerza máxima del modelo viscoelástico es inferior a la del modelo elástico. Este hecho también es positivo ya que una medida de seguridad habitual en este tipo de sistemas es limitar hasta un determinado valor la fuerza que se restituye al usuario.

Un inconveniente de este modelo, es que requiere conocer la derivada de la penetración. El cálculo de esta derivada requiere emplear un método de diferenciación que quizá pueda afectar a la estabilidad del sistema.

Además, si la resolución de los sensores de posición no es muy grande, la señal derivada puede llegar a tener una muy baja precisión de cálculo, con lo que se apreciarían efectos no lineales en el sistema.

8.3 MODELO VISCOELÁSTICO NO LINEAL

Si además se añade la penetración a la componente viscosa se obtiene el modelo viscoelástico no lineal [Hunt, 1975].

Para evitar el problema de las discontinuidades en la fuerza del modelo anterior, lo más sencillo es hacer incrementar el coeficiente b_e desde cero en función de la penetración

$$f_e = kx + b_e x\dot{x}$$

De esta manera se evita el problema de las discontinuidades en la fuerza, manteniendo las ventajas del modelo anterior.

La *Fig. 8.3* muestra una comparativa realizada entre los modelos elástico y viscoelástico. También en este caso una pared virtual repele una masa de 5 kg que impacta con una velocidad de 1 m/s. Se representa el módulo de la fuerza en función de la penetración.

Como era de esperar, se comprueba que el modelo viscoelástico no lineal disipa energía, el área sombreada de la *Fig. 8.3*, aunque en menor medida que el modelo viscoelástico anterior lineal. Por este motivo la penetración es similar al modelo elástico. También

se comprueba el hecho de que la fuerza al inicio de la penetración y al final de la misma es igual a cero.

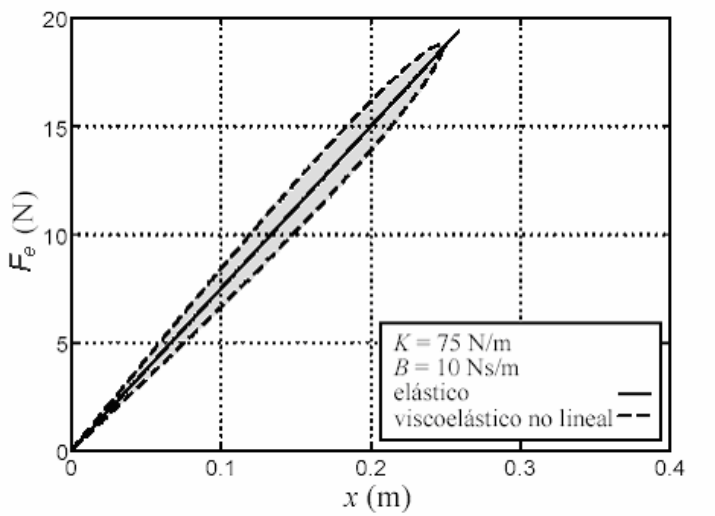

Figura 8.3 *Comparativa elástico vs. viscoelástico no lin*eal [Gil, J. J, 2003].

Otro modelo viscoelástico no lineal propuesto en [Hunt, 1975] consiste en:

$$f_e = kx^n + b_e x^n \dot{x}$$

Sin embargo este otro modelo presenta la dificultad de la elección de las potencias adecuadas para hacer que el sistema sea estable. Este modelo presenta las mismas características que el modelo anterior, además introduce un efecto de incremento de la rigidez conforme aumenta la penetración.

8.4 MODELO SEMIVISCOELÁSTICO

Este modelo, propuesto por [Rosenberg,1993], es igual que el modelo viscoelástico, añadiendo solo la componente del amortiguamiento mientras la penetración aumenta. Si la penetración disminuye sólo se restituye la componente elástica.

$$f_e = \begin{cases} kx + b\dot{x} & \dot{x} > 0 \\ kx & \dot{x} \le 0 \end{cases}$$

Presenta las mismas ventajas del modelo viscoelástico y además no tiene el inconveniente del efecto de atracción al usuario cuando abandona el contacto con el objeto.

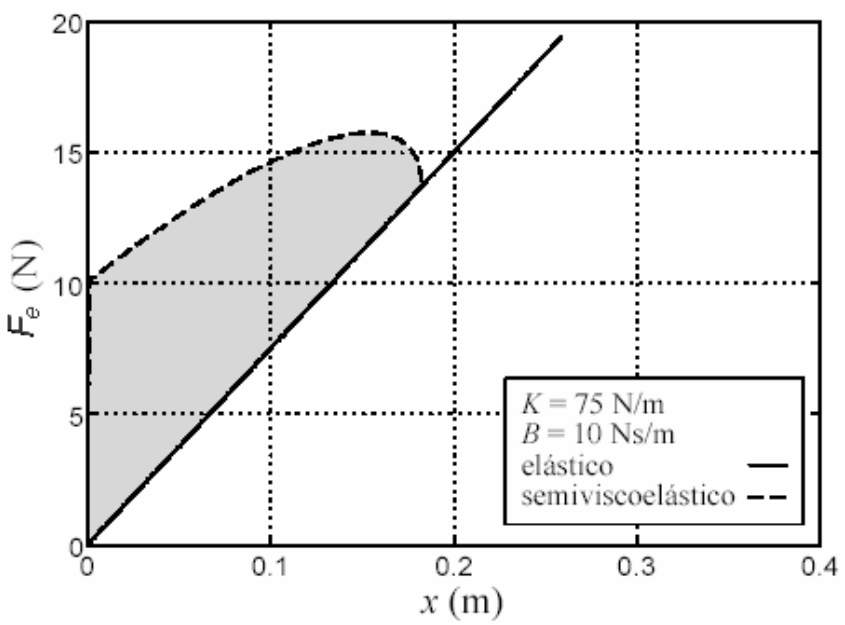

Figura 8.4 *Comparativa de los modelos elástico y semiviscoelástico* [Gil, J. J, 2003].

En la *Fig. 8.4* se presenta una simulación de las mismas características que los anteriores modelos y se observa cómo este modelo no restituye fuerzas negativas.

8.5 OPEN LOOP FORCE PULSES

Más que de un modelo de contacto se trata de una estrategia de control basada en eventos. Para limitar la penetración, se busca cancelar rápidamente el momento y además producir una fuerza característica del tipo de contacto [Hwang, 2004].

Supongamos una masa m que se desplaza a una velocidad inicial v_0 hacia un muro virtual. Aplicando un pulso de fuerza F_{peak} y duración T, se cancelará el momento P.

$$P = \int_0^T F(t)dt$$

con

$$P = mv_0$$

y tomando

$$T = \frac{mv_0}{F_{peak}}$$

donde F_{peak} es la fuerza en el momento de establecimiento del contacto. Este impulso (F_{peak}) neutralizará la velocidad y detendrá la masa en:

$$\Delta x = \frac{\frac{1}{2} m v_0^2}{F_{peak}}$$

Si la masa no se conoce con exactitud se puede estimar examinando su actual desaceleración:

$$\hat{m} = \frac{F_{peak} T}{v(0) - v(T)}$$

donde *v(0)* es la velocidad antes de la penetración y *v(T)* es la velocidad en el instante de muestro posterior a la penetración.

8.6 BRAKING PULSE

Se trata de otra estrategia de control propuesta por [Salcudean, 1996], proponen aumentar la rigidez aparente durante el impacto.

Se basa en el hecho de que un objeto rígido lanzado contra la mayoría de las superficies alcanzara contacto continuo en poco tiempo. Por lo tanto la masa debe detenerse lo mas rápido posible (idealmente en un periodo de muestreo). Este pulso de frenado se puede calcular, siempre que la fuerza resultante no sature el actuador, como:

$$f_{pulse} = m(v_k - v_{k-1}) / T = -m v_{k-1} / T$$

donde f_{pulse} es la fuerza a aplicar en el momento de contacto; v_k, v_{k-1} son respectivamente las velocidades en el instante de muestreo en el que ser produce la colisión y el anterior; y T el período de muestreo.

Esto corresponde a una fuerte viscosidad en el momento de la penetración, y para $X_{k-1} \leq 0$ se puede implementar la fuerza que se restituye al operario f_e como:

$$f_e = -(k_{pulse} + b_e / T)(x_{i-1} - x_{i-2}) \qquad \text{si} \quad x_{i-2} > 0 \quad x_{i-1} \leq 0$$

$$f_e = -k_p x_{i-1} - \frac{b_e}{T}(x_{i-1} - x_{i-2}) \qquad \text{en otro caso}$$

con

$$k_{pulse} + k_v / T = m / T^2$$

Siendo f_e es la fuerza restituida por el modelo para el resto de períodos de muestreo después de establecer el contacto; k_p, la rigidez del muro; b_e, el amortiguamiento del contacto; k_{pulse} el valor necesario para conseguir frenar la masa en el momento del contacto; y el subíndice i señala el actual período de muestreo.

Si se satura el motor el pulso de fuerza se debe limitar en magnitud. Para ello éste se divide en diferentes periodos de muestreo, de forma que la suma total sea la fuerza deseada.

En la *Fig. 8.5* se puede ver la representación de la fuerza frente al tiempo. Se puede comprobar cómo al aplicar el pulso el pico de fuerza es mucho mayor que si no se aplica.

La fuerza que devuelve el mecanismo es mucho mayor, así que la penetración que se producirá será menor. Así se simula el choque y la pérdida instantánea de velocidad con mayor realismo.

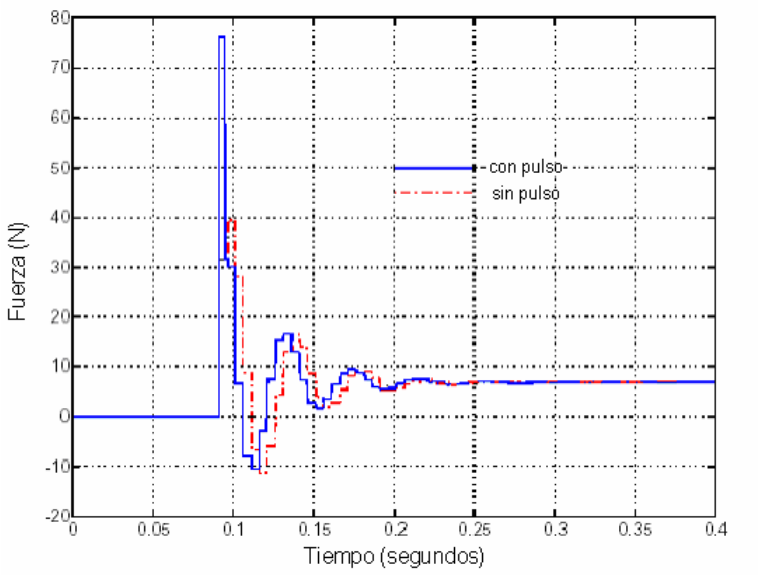

Figura 8.5 *Gráfico de Fuerza vs. Tiempo* [Salcudean, 1996].

En la *Fig. 8.6.* se puede observar claramente cómo la penetración es menor en la simulación realizada en el artículo en el caso de que se dé un pulso de frenado al chocar con la superficie.

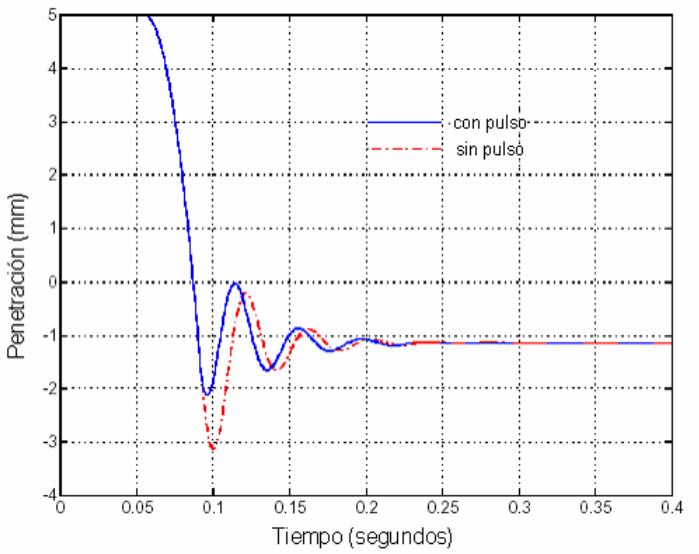

Figura 8.6 *Gráfico de Penetración vs. Tiempo* [Salcudean, 1996].

9 VALIDACIÓN DE LA TARJETA EOS

El proceso de validación de la tarjeta EOS se dividió en tres partes: Análisis del tiempo de ejecución para el cálculo de funciones trigonométricas, análisis del tiempo de ejecución para el control del PHANToM y pruebas utilizando modelos de contacto.

9.1 ANÁLISIS DEL TIEMPO DE EJECUCIÓN PARA EL CÁLCULO DE FUNCIONES TRIGONOMÉTRICAS

En la primera prueba de la validación, se realizaron mediciones de tiempo con el fin de determinar las mejoras en la velocidad de ejecución, que implica el uso de una IP Core para el cálculo de funciones trigonométricas, en comparación con los resultados obtenidos realizando esta misma operación directamente en software.

La prueba realizada consistió en medir el tiempo que emplea el dispositivo para calcular el seno y el coseno de un ángulo. Se tuvieron en cuenta tres casos: Cálculo utilizando la IPSC, cálculo directo en software y cálculo utilizando la unidad de coma flotante que posee el MicroBlaze.

9.1.1 Cálculo de Funciones Trigonométricas Utilizando la IPSC

En esta prueba se utilizan las salidas digitales de la tarjeta EOS, para generar una señal cuadrada que se activa antes de llamar a la IPSC y que se desactiva cuando la Core termina su trabajo. De esta forma es posible medir el tiempo de ejecución de la IPSC.

Para tratar que el tiempo de retardo producido por la activación y desactivación de la salida digital no incida en la medida global, se tomó el tiempo que tarda la tarjeta en calcular veinte veces el seno y el coseno de un ángulo.

Como resultado de la prueba, se obtuvo que el tiempo empleado para calcular los cuarenta valores especificados fue de 268.37 µs, lo que equivale a una frecuencia de 3.73 khz como puede verse en la *Fig. 9.1.*

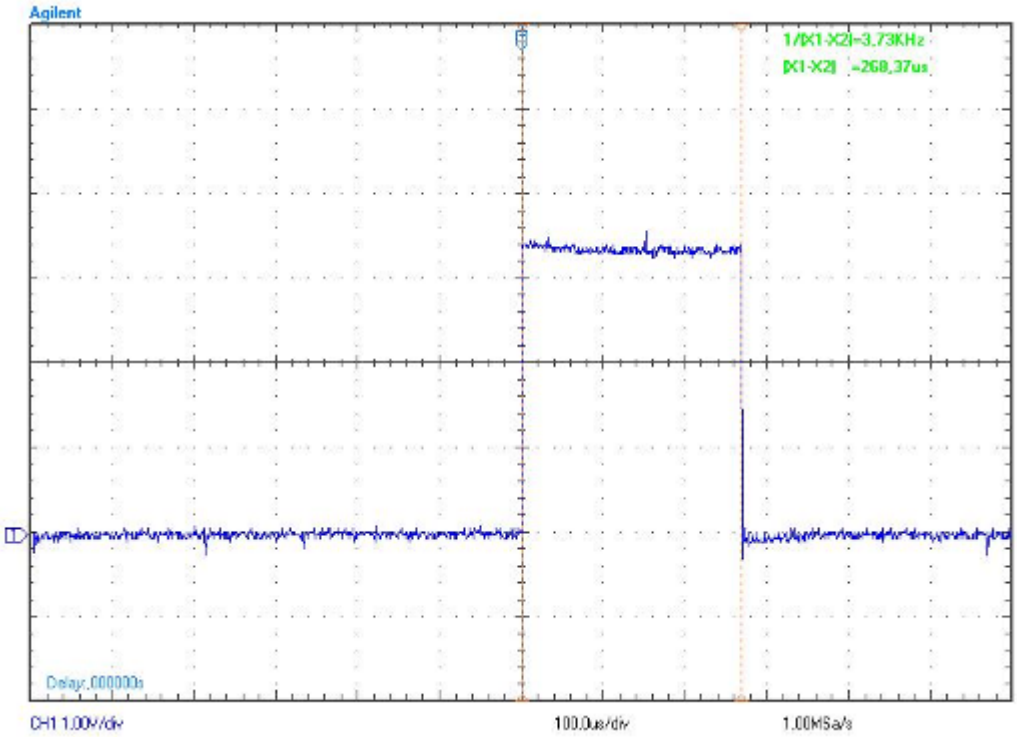

Figura 9.1 *Cálculo de Seno y Coseno en la IPSC.*

9.1.2 Cálculo de Funciones Trigonométricas Directamente en Software

Existe la posibilidad de efectuar los cálculos de seno y coseno de un ángulo, utilizando los recursos de software que posee el MicroBlaze. Específicamente se empleó la librería math.h, que permite realizar cálculos matemáticos directamente sobre el procesador.

Para esta prueba, igualmente se midió el tiempo empleado para calcular el seno y coseno de veinte valores de entrada, obteniendo como resultado un tiempo 11.5 ms, lo que equivale a una frecuencia de 86.96 hz como puede verse en la *Fig. 9.2.*

De esta manera se puede inferir, que esta opción de cálculo es 43 veces más lenta, teniendo en cuenta los valores obtenidos empleando la IPSC como se vio en el apartado anterior.

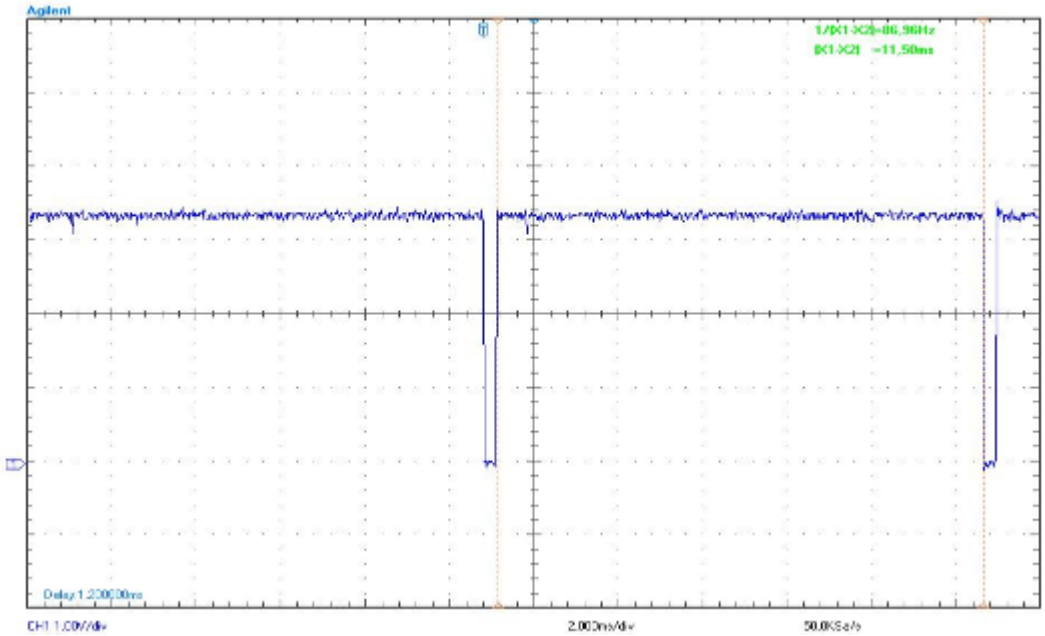

Figura 9.2 *Cálculo de Seno y Coseno en el MicroBlaze.*

9.1.3 Cálculo de Funciones Trigonométricas Utilizando la Unidad de Coma Flotante del MicroBlaze

Existe una tercera opción para realizar los cálculos de seno y coseno en la tarjeta EOS, habilitando la unidad de coma flotante que posee el MicroBlaze. Esta unidad se utiliza como coprocesador aritmético, permitiendo que el MicroBlaze no ocupe sus recursos en éstos procesos.

Nuevamente se realizó una prueba, midiendo el tiempo empleado para calcular veinte valores de seno y coseno de un determinado ángulo, obteniendo como resultado un tiempo de 412.55 µs, lo que equivale a una frecuencia de 2.42 kHz, como puede verse en la *Fig. 9.3.*

Aunque la respuesta en tiempo es menor al valor obtenido en el cálculo directo sobre el MicroBlaze, el tiempo empleado en este caso, es mayor que el valor obtenido utilizando la IPSC.

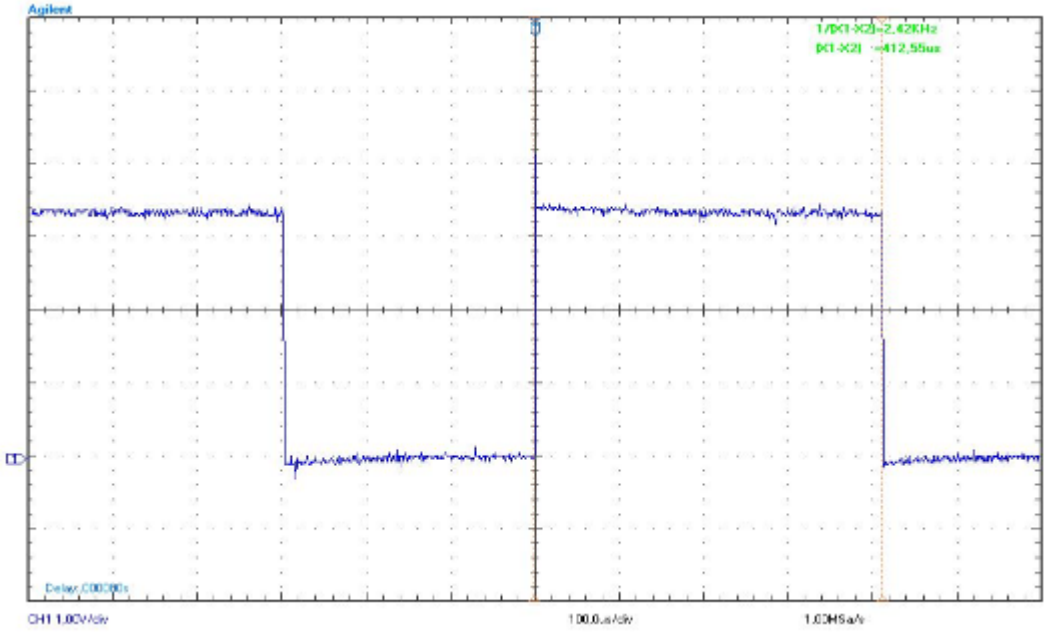

Figura 9.3 *Cálculo de Seno y Coseno Utilizando Unidad de Coma Flotante*

9.2 ANÁLISIS DEL TIEMPO DE EJECUCIÓN PARA EL CONTROL DEL PHANTOM

El dispositivo háptico utilizado para realizar las pruebas de control con la tarjeta EOS, es el Phantom 1.0 Premium *Fig. 9.4*. Este dispositivo posee 3 GdL sensados y con reflexión de fuerza, que permiten al usuario interactuar con ambientes virtuales obteniendo una realimentación táctil.

Figura 9.4 *PHANToM 1.0 Premium*

Para diseñar un sistema de control para este tipo de dispositivo, es necesario resolver el problema cinemático y además establecer los valores de fuerza que serán realimentados al usuario (Jacobiana transpuesta), para estructurar así el lazo de control del sistema.

El análisis del lazo de control propuesto puede analizarse desde dos puntos de vista, aunque al final éstos resultan ser complementarios. Es posible analizar el lazo de control teniendo como variable de entrada del sistema, la posición en coordenadas articulares de cada uno de los grados de libertad del PHANToM.

Teniendo como base las mediciones realizadas con los encoders alojados en cada uno de los grados de libertad de la interfaz, es posible obtener la relación angular (coordenadas articulares) existente entre las articulaciones del robot y la estructura como tal del mismo.

A partir de las coordenadas articulares es posible obtener las coordenadas cartesianas del extremo del robot, utilizando para ello, la Cinemática Directa del dispositivo descrita en [Çavuşoğlu, 2001].

Dependiendo del tipo de sensación que se quiera percibir durante la interacción con el háptico, se debe seleccionar el modelo de contacto a implementar. Éstos modelos de contacto (descritos en el Capítulo 8), transforman a una representación en fuerza (colisiones) la ubicación espacial obtenida de la cinemática directa.

De esta manera se le da una representación física a las colisiones o eventos que se presenten durante la simulación y que han sido pre-programados.

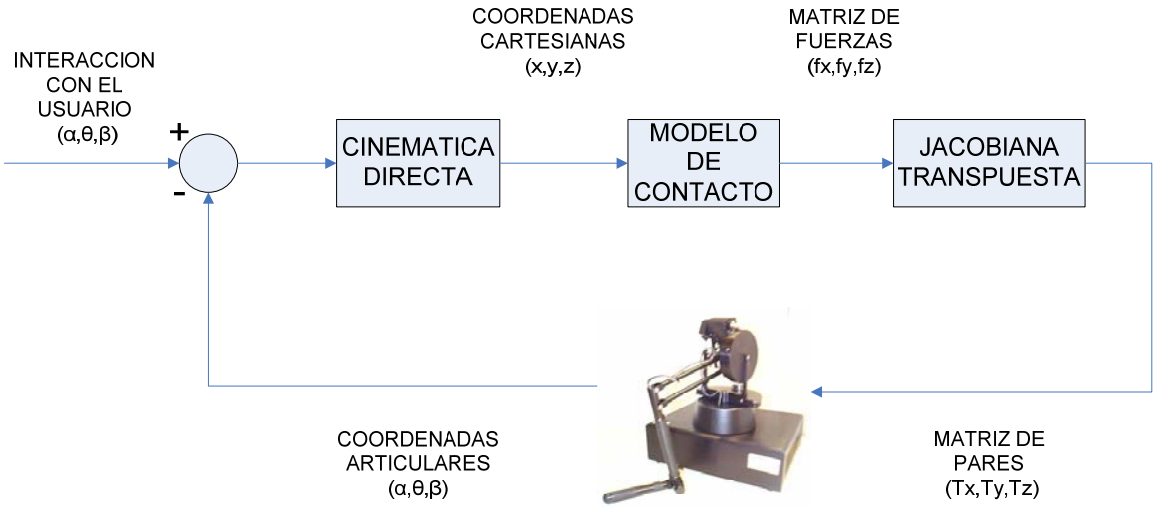

Figura 9.5 *Lazo de Control Implementado en Posición.*

Posteriormente es necesario traducir la fuerza obtenida del modelo de contacto, a un par aplicable a los actuadores, en este caso motores, del dispositivo háptico. Para el cálculo de pares se utiliza la Jacobiana Transpuesta, cuyo algoritmo de cálculo se describe en [Çavuşoğlu, 2001].

La otra forma de analizar el sistema es desde el punto de vista de la fuerza. Es decir, la entrada del sistema será la fuerza resultante entre la diferencia de la fuerza que ejerce el usuario sobre la interfaz y la fuerza que generada por la interfaz misma.

Pero esta fuerza resultante no es interpretada como tal por el sistema, pues no se cuenta con sensores de fuerza que lo permitan. Por esta razón la resultante de fuerza es interpretada como la posición o variación de esta, que adopten cada uno de los grados de libertad de la interfaz robótica.

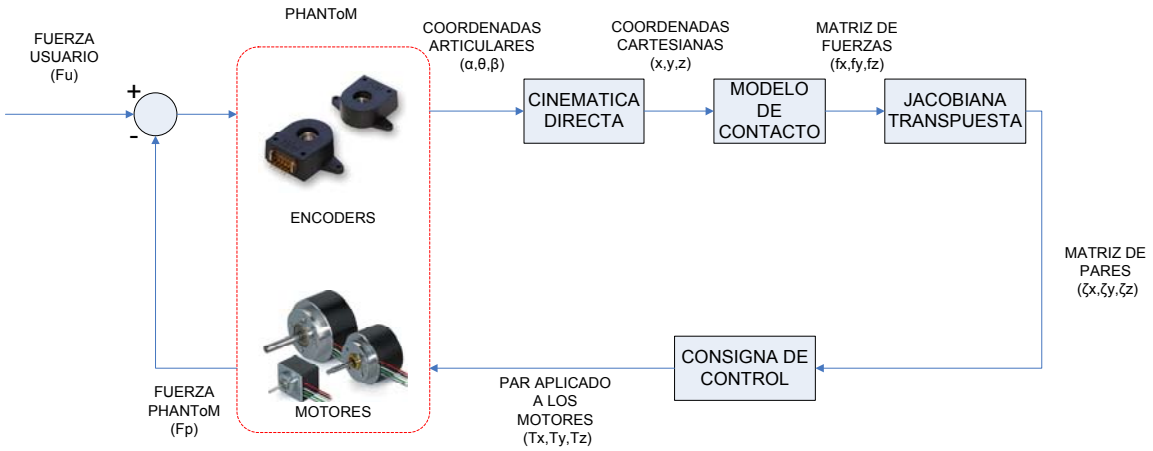

Figura 9.6 *Lazo de Control Implementado en Fuerza.*

Pero al igual que en el método descrito en primera instancia, es necesario utilizar las relaciones cinemáticas de la estructura de la interfaz y calcular la Jacobiana de la misma, para así estimar los pares que deben ser aplicados a los motores cerrando así el lazo de control.

De esta manera queda explicito que no se realiza un control directamente en fuerza sino que se deriva en un control en posición, pues en todo momento existe una relación intrínseca entre ambas variables.

Por esta razón, en adelante se presentan los resultados obtenidos en posición, pues resulta más práctico interpretar estos resultados, sabiendo que implícitos en ellos se encuentra la respuesta en fuerza.

9.2.1 Conversión a Coordenadas Articulares

El primer paso para comenzar a estructurar el lazo de control del dispositivo háptico PHANToM 1.0, es traducir a coordenadas articulares los pulsos de Encoder que son enviados desde cada uno de los tres grados de libertad que posee el dispositivo.

Como fue mencionado en el Capítulo 7, la tarjeta EOS posee un módulo de seis entradas para lectura de Encoders diferenciales, implementado y descrito en hardware reconfigurable VHDL (IP Core) dentro de la FPGA que posee el sistema.

Este modulo retorna el valor de pulsos de Encoder producido cuando alguno de los grados de libertad del PHANToM varía su posición. Pero el valor de pulsos, no puede recibir una interpretación física directa sino que es necesario aplicar una conversión a coordenadas articulares, teniendo en cuenta los parámetros mecánicos que posee la estructura del dispositivo.

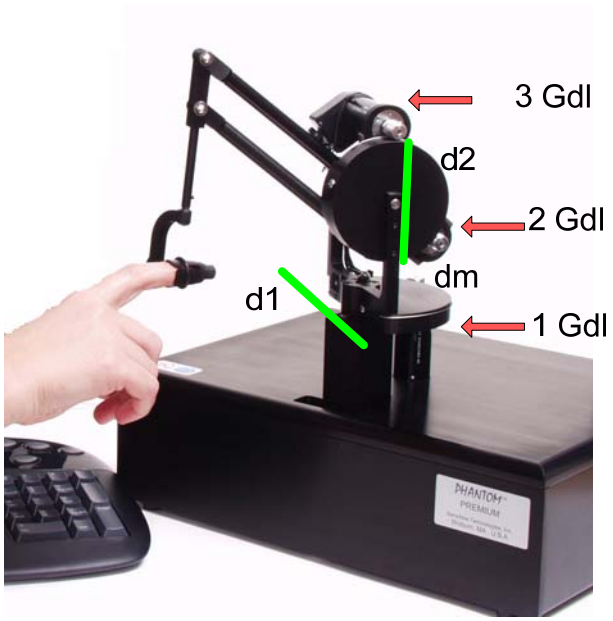

Figura 9.7 *Dimensiones y parámetros del PHANToM 1.0*

De esta manera es posible plantear las ecuaciones para transformar los pulsos de Encoder a coordenadas articulares:

- Primer Gdl:

$$k = \frac{118}{13} = \frac{d_1}{d_m} \qquad (1)$$

$$\theta_1 = \theta_{1e} * k \qquad (2)$$

$$\theta_1 = \theta_{1e} * k * \frac{2\pi}{ppv} \qquad (3)$$

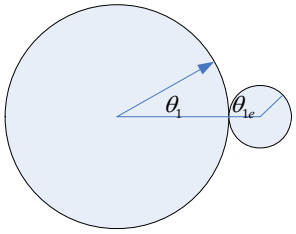

Figura 9.8 *Relación de diámetros entre el motor y la articulación*

Donde:

k Relación de diámetros (Motor y acople mecánico).
ppv Pulsos por vuelta del encoder.
θ_1 Ángulo en radianes descrito por el primer Gdl.
θ_{1e} Ángulo en ppv descrito por el encoder del primer Gdl.

Para el segundo y tercer Gdl, se aplican las mismas relaciones de conversión, teniendo en cuenta los parámetros específicos (diámetro) que poseen.

Cabe resaltar que debido a la dependencia mecánica (transmisión por cable) que presenta el tercer Gdl sobre el segundo, es necesario modificar la ecuación resultante como se muestra a continuación:

$$\theta_3 = \theta_{3e} * k * \frac{2\pi}{ppv} - \theta_2 \qquad (4)$$

La implementación de estas ecuaciones se presenta en el ANEXO 4.

9.2.2 Cinemática Directa

Efectuando el cálculo de la cinemática directa, es posible representar y describir la localización de un objeto en el espacio tridimensional con respecto aun sistema de referencia fijo.

En este caso, se utiliza la cinemática directa del PHANToM, para estimar la posición del efector final, teniendo en cuenta la posición individual de cada uno de los 3 GdL que posee el PHANToM.

Este algoritmo no fue inferido por el autor de este proyecto, pues existe bibliografía tanto del fabricante como en publicaciones científicas en las que se presenta dicho algoritmo; ver [Çavuşoğlu, 2001].

Sin embargo a continuación se presenta una clara descripción de los pasos que se deben seguir para llegar a determinar el algoritmo propuesto, esto con el ánimo de dar al lector una mejor base para el entendimiento de las pruebas realizadas.

En primera instancia es necesario especificar los ejes de referencia a partir de los cuales se realizará todo el análisis cinemática del dispositivo. Para ello se deben seguir las reglas definidas en el algoritmo de Denavit – Hartenberg [Craig, 2004].

El algoritmo de Denavit – Hartenberg dicta una serie de pasos que se deben seguir, para desarrollar de una forma simple y práctica, el problema cinemático de una estructura. Es necesario resaltar que utilizando el algoritmo de Denavit – Hartenberg es posible obtener múltiples soluciones válidas para una misma estructura, pues dependiendo de la asignación de ejes de referencia que se realice, se obtendrá una u otra solución.

Existen dos versiones del algoritmo de Denavit – Hartenberg:

Versión ‘Standart’ y la versión modificada por Craig [Craig, 2004]. Cada una de ellas utiliza convenciones diferentes para la ubicación de los ejes de referencia. No es posible mezclar las soluciones obtenidas con estos algoritmos y por ello siempre se debe aclarar cual de las versiones se ha utilizado, pues de lo contrario se podría estar induciendo a errores interpretativos.

Del mismo modo, se debe aclarar que las ecuaciones que describen el desarrollo cinemático de una estructura, pueden variar dependiendo de la tabla de desarrollo obtenida con Denavit – Hartenberg.

Esto no significa que alguna de las soluciones sea incorrecta, pero si da un patrón para pensar que el desarrollo cinemática realizado, se ha hecho en base a otros ejes de referencia.

A continuación se presenta una solución al problema cinemática del dispositivo háptico PHANToM 1.0, utilizando el algoritmo de Denavit – Hartenberg en su versión ‘Standart’:

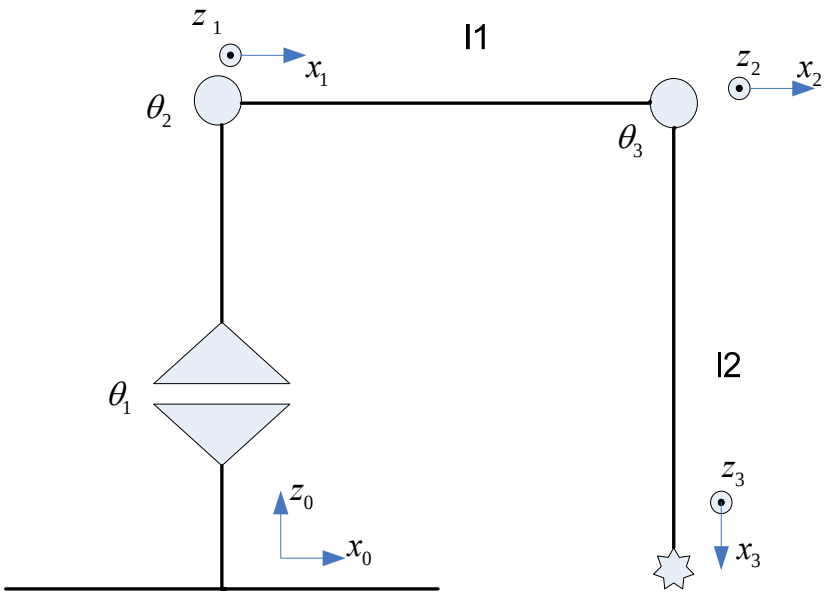

Figura 9.9 *Aplicación del algoritmo de Denavit-Hartenberg*

Habiendo ubicado los ejes de referencia, es posible tabular los parámetros tanto angulares como de longitud con los cuales se construyen las matrices de traslación.

A continuación se presenta la tabla con los datos obtenidos para el anterior diagrama:

	θ_i	d_i	a_i	α_i
1	θ_1	l_2	0	90
2	θ_2	0	l_1	0
3	$\theta_3 - \theta_2 - 90$	0	l_2	0

Utilizando la Robotics Toolbox de Matlab, es posible comprobar que el análisis cinemático planteado es correcto, obteniendo una grafica tridimensional de la estructura robótica, empleando los parámetros consignados en la tabla de resultados del algoritmo de Denavit -Hartenberg.

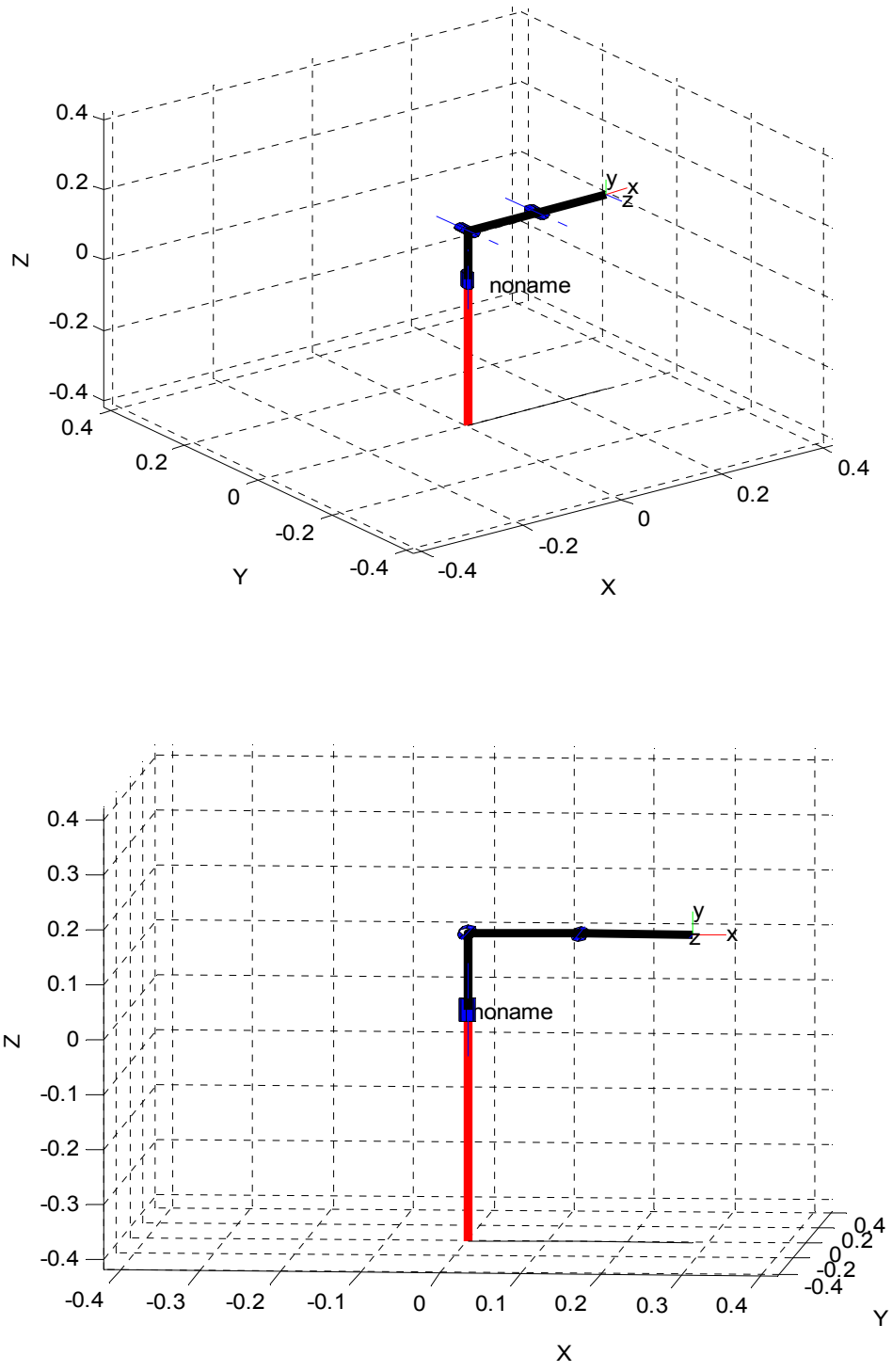

Figura 9.10 *Modelo obtenido con la Robotics Toolbox de Matlab*

Los algoritmos empleados para determinar la gráfica de la estructura robótica a partir de la tabla de valores de Denavit – Hartenberg, se pueden observar en el ANEXO 5.

A partir de los datos obtenidos con la aplicación del algoritmo de Denavit –Hartenberg, se construyen las matrices de traslación, teniendo en cuenta que cada casilla de la tabla corresponde a una matriz de rotación, siendo necesario así realizar el producto entre matrices para finalmente hallar la respectiva matriz de traslación.

De esta manera se obtiene:

$$ {}^{0}T_{1} = \begin{bmatrix} C_1 & -S_1 & 0 & 0 \\ S_1 & C_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}T_{2} = \begin{bmatrix} C_2 & -S_2 & 0 & 0 \\ S_2 & C_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_2 & -S_2 & 0 & l_1 * C_2 \\ S_2 & C_2 & 0 & l_1 * S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}T_{3} = \begin{bmatrix} C_{3-2} & -S_{3-2} & 0 & 0 \\ S_{3-2} & C_{3-2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_{3-2} & -S_{3-2} & 0 & l_2 * C_{3-2} \\ S_{3-2} & C_{3-2} & 0 & l_2 * S_{3-2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}T_{3} = {}^{0}T_{1} * {}^{1}T_{2} * {}^{2}T_{3} $$

$$ {}^{0}T_{3} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

La matriz de traslación del sistema aporta la información tanto de la posición como también de la rotación del elemento final de la estructura, con respecto a la base u origen de coordenadas.

En este caso resulta importante para las pruebas a realizar, conocer las componentes que describen la posición, es decir p_x, p_y, p_z , siendo ésta la solución del problema cinemático propuesto.

De esta forma p_x, p_y, p_z quedan definidos como:

p_x =-cos(t1)*cos(t2)*l2*sin(-t3+t2)+cos(t1)*sin(t2)*l2*cos(-t3+t2)+cos(t1)*l1*cos(t2)

p_y = -sin(t1)*cos(t2)*l2*sin(-t3+t2)+sin(t1)*sin(t2)*l2*cos(-t3+t2)+sin(t1)*l1*cos(t2)

p_z = -sin(t2)*l2*sin(-t3+t2)-cos(t2)*l2*cos(-t3+t2)+sin(t2)*l1+l2

En donde t1, t2 y t3 corresponden a los ángulos de giro de cada articulación y (l1 y l2) son las dimensiones de cada link de la estructura.

Para conocer el algoritmo definitivo, empleado para calcular la cinemática directa del PHANToM referirse al ANEXO 4.

En la *Fig. 9.11.* se observan los valores de tiempo medidos para el cálculo de la cinemática directa del PHANToM. La gráfica azul, representa el tiempo empleado para calcular la cinemática utilizando para ello la IPSC, mientras que la gráfica violeta corresponde al tiempo empleado calculando la cinemática directamente en software.

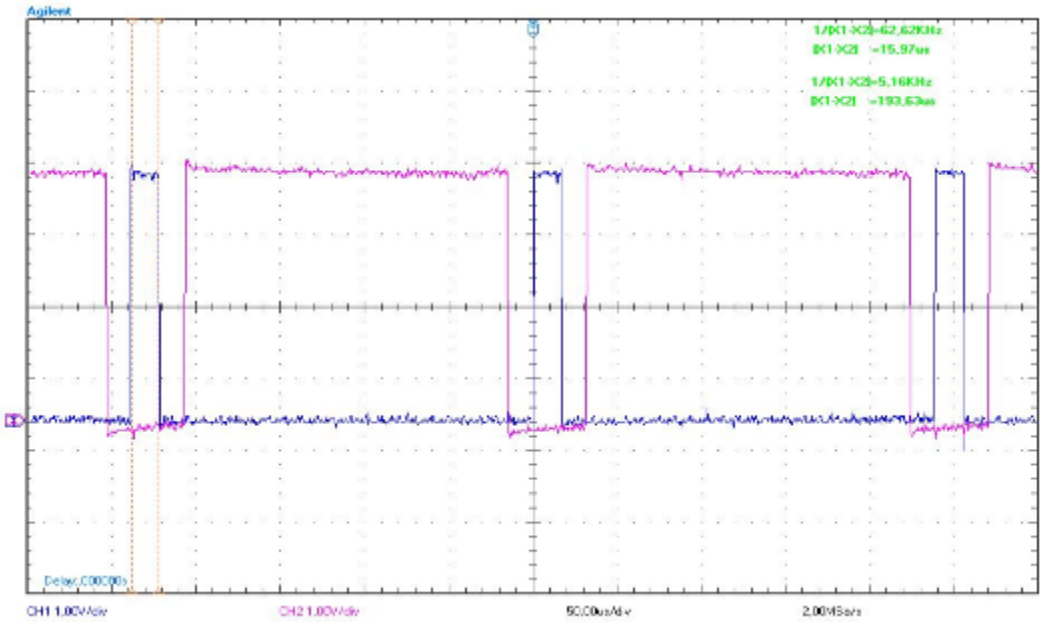

Figura 9.11 *Cálculo de la Cinemática Directa del PHANToM.*

9.2.3 Jacobiana Transpuesta

La Jacobiana de una estructura robótica relaciona las velocidades articulares con las velocidades rotacionales del extremo del robot.

Uno de los métodos más utilizados para determinar la Jacobiana de una estructura, es el método de propagación de velocidades, en el cual se analiza la distribución de las velocidades por las articulaciones del robot. De esta forma los enlaces entre dos articulaciones se suponen rígidos con movimiento descrito por vectores de velocidades lineales y angulares.

El algoritmo general que describe el método de propagación de velocidades [Craig, 2004], se presenta a continuación:

Matriz de velocidades angulares (rotacionales):

$$^{i+1}w_{i+1} = {}^{i+1}R_i \, {}^{i}w_i + \theta_{i+1} \, {}^{i+1}z_{i+1}$$, de donde:

$$\theta_{i+1} \, {}^{i+1}z_{i+1} = {}^{i+1}\begin{bmatrix} 0 \\ 0 \\ \theta_{i+1} \end{bmatrix}$$

Matriz de velocidades lineales (prismáticas):

$$^{i}v_{i+1} = {}^{i}v_i + {}^{i}w_i \times {}^{i}P_{i+1}$$

De esta manera es posible calcular la Jacobiana como:

$$\begin{bmatrix} v \\ w \end{bmatrix} = J_q$$

R_i Matriz de rotación de la articulación i

w_i Velocidad angular de la articulación i

v_i Velocidad lineal de la articulación i

z_i Componente z de la posición de la articulación i

Como se puede apreciar, es necesario utilizar la cinemática directa calculada previamente, pues en el cálculo de la Jacobiana se emplean las matrices de rotación (incluida en la matriz traslacional) calculadas en el análisis cinemático.

Por medio de la jacobiana transpuesta, es posible calcular el valor de los pares que deben ser aplicados a los motores, para reproducir las colisiones del objeto virtual, a partir de las fuerzas estimadas a través de los modelos de contacto vistos anteriormente en el Capítulo 8.

$$\tau_{xyz} = J^T F_{xyz}$$

Para conocer el algoritmo empleado para determinar los pares del PHANToM referirse al ANEXO 4.

Del mismo modo como se comentó en el anterior apartado, el algoritmo empleado para el cálculo de la Jacobiana Transpuesta, se encuentra publicado en [Çavuşoğlu, 2001].

En la *Fig. 9.12.* se observan los valores de tiempo medidos para calcular el valor de los pares. La gráfica azul, representa el tiempo empleado utilizando IPSC, mientras que la gráfica violeta corresponde al tiempo empleado calculando los pares directamente en software.

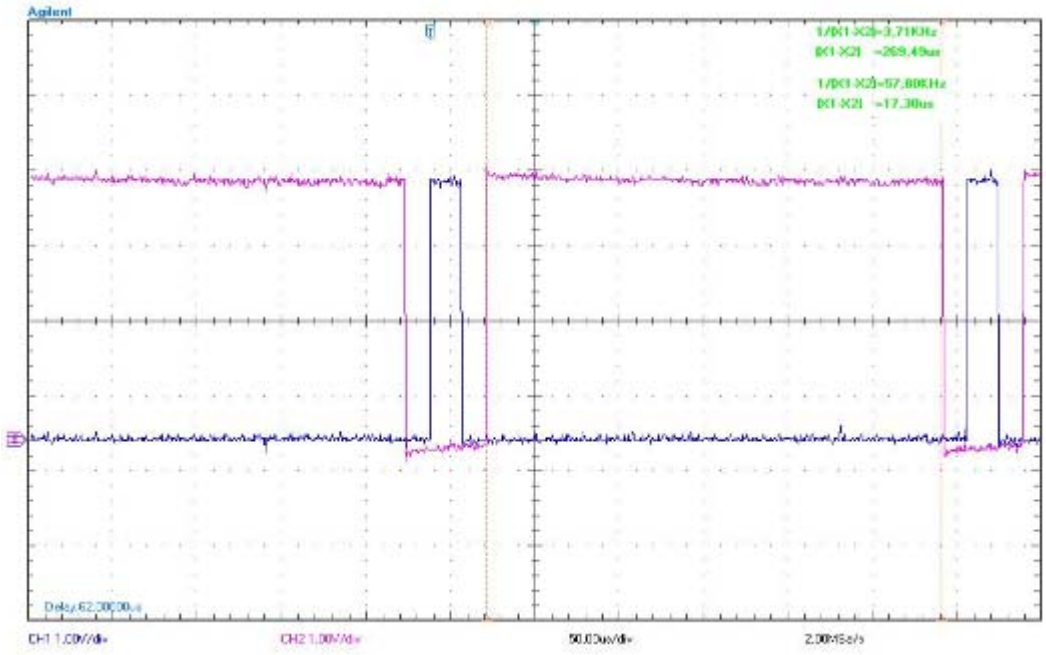

Figura 9.12 *Cálculo de Pares Utilizando la Jacobiana Transpuesta.*

9.2.4 Análisis del Tiempo de Ejecución del Lazo de Control

Para determinar la eficiencia de la tarjeta EOS, en la implementación del lazo de control de un dispositivo háptico, se realizó la medida del tiempo de ejecución del modelo de control para el PHANToM 1.0. Se realizaron dos pruebas: una empleando la IPSC y la otra realizando todos los cáculos directamente en el MicroBlaze.

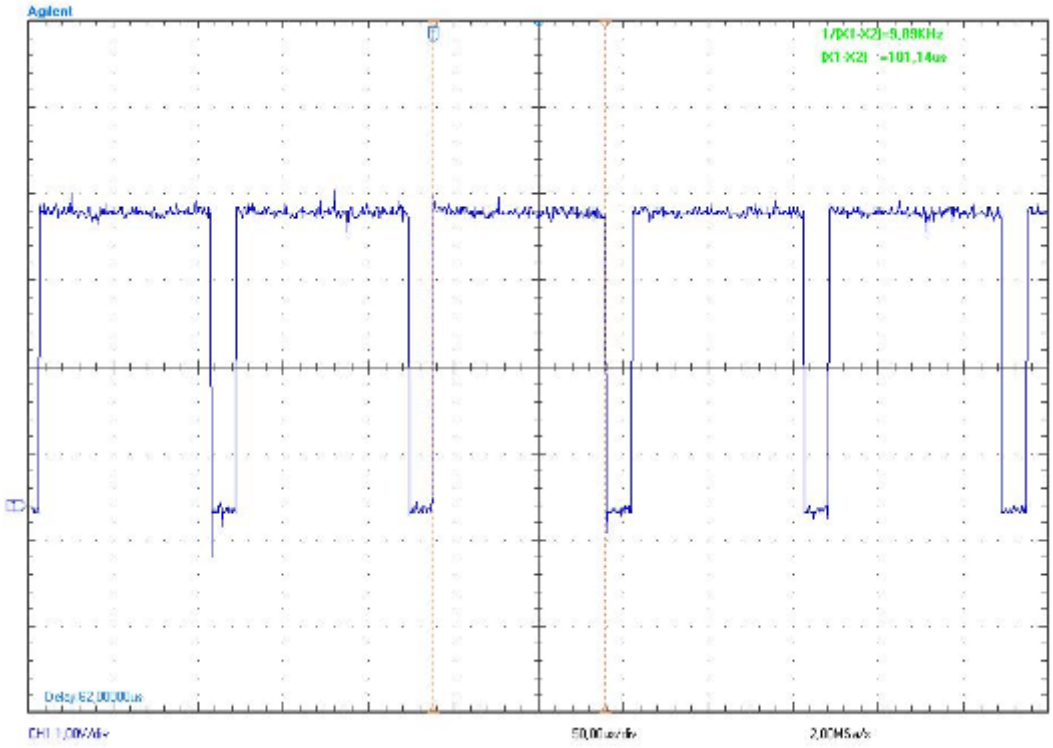

Figura 9.13 *Tiempo Empleado en el Lazo de Control Utilizando la IPSC.*

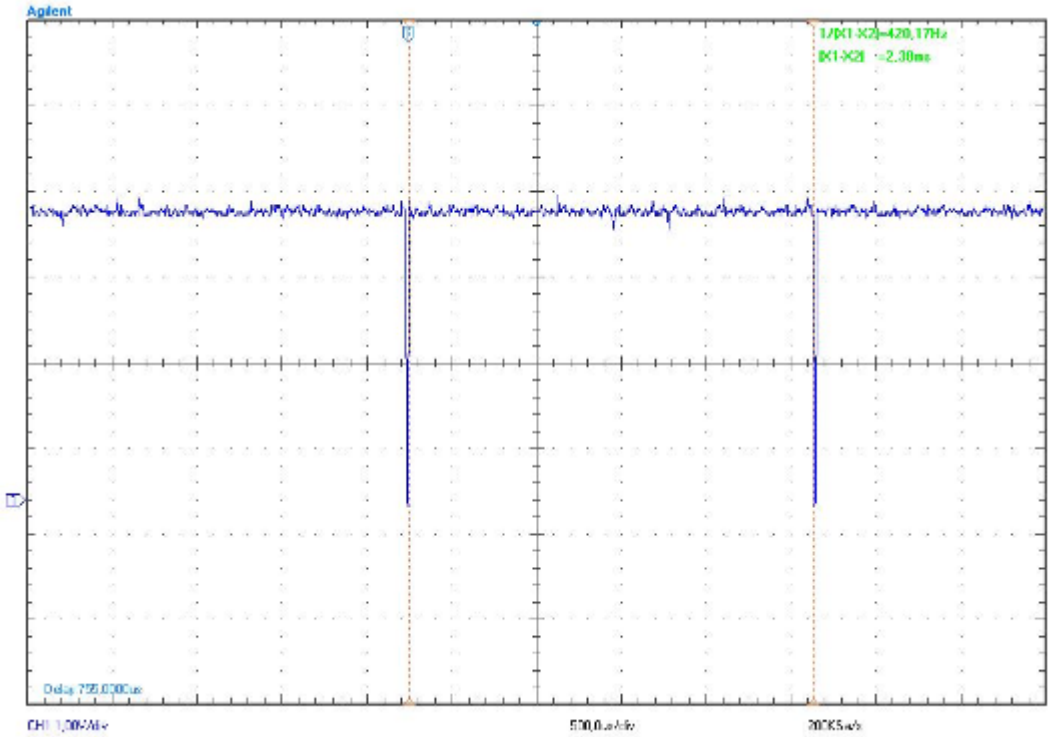

Figura 9.14 *Tiempo Empleado en el Lazo de Control Implementado en Software.*

Es de aclarar que en las dos pruebas realizadas se empleó el modelo viscoelástico, para determinar las fuerzas de colisión que se presentan durante la sesión háptica.

A partir de los resultados obtenidos, se estableció que utilizando la IPSC, el lazo de control completo tarda en su ejecución 101.14 µs, lo que equivale a una frecuencia de 9.89 kHz. Del mismo modo se obtuvo un tiempo 2.30 ms, equivalente a una frecuencia de 420.17 Hz, realizando todos los cálculos del lazo de control directamente sobre el MicroBlaze.

9.3 PRUEBAS UTILIZANDO MODELOS DE CONTACTO

Utilizando los modelos de contacto descritos en el capítulo 8, se realizaron pruebas para determinar el comportamiento del Phantom, durante una sesión háptica programada en la tarjeta EOS.

La experiencia realizada consistió en definir una pared virtual en el eje ‘z,’ a 15 cm de la base del Phantom. Utilizando los modelos elástico y viscoelástico, se comprobó el seguimiento en posición que realiza el sistema, cuando se colisiona contra la pared definida.

Además en las pruebas se variaron los parámetros específicos de cada modelo de contacto (rigidez y viscosidad) y así observar el efecto que cada uno de ellos produce en la respuesta obtenida.

9.3.1 Modelo Elástico

A continuación se presentan los resultados de las pruebas realizadas, empleando el modelo elástico:

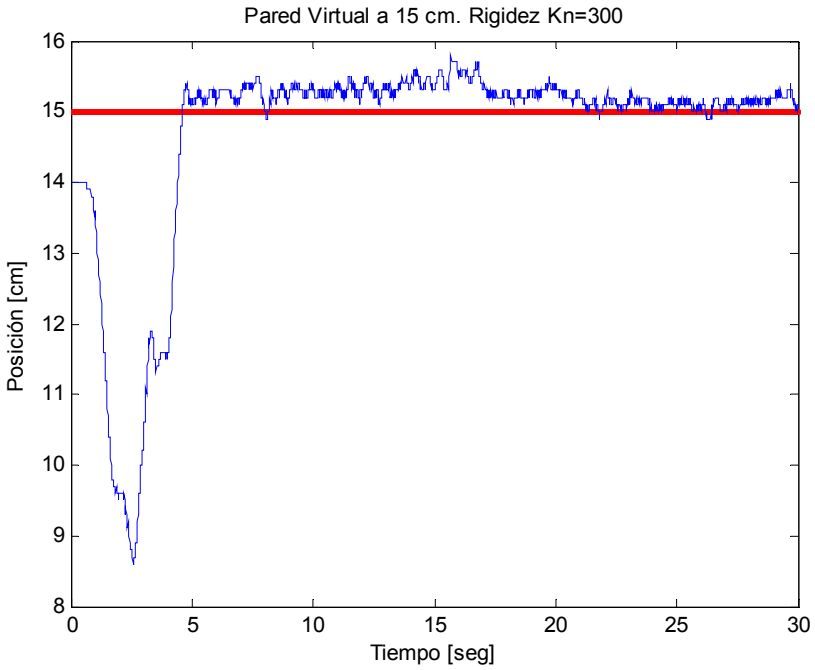

Figura 9.15 *Prueba 1 Modelo Elástico.*

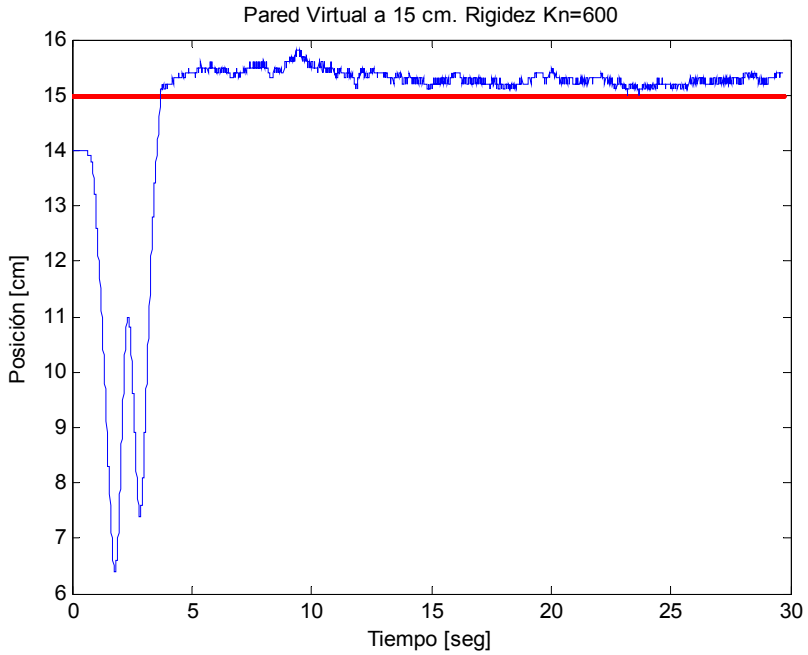

Figura 9.16 *Prueba 2 Modelo Elástico.*

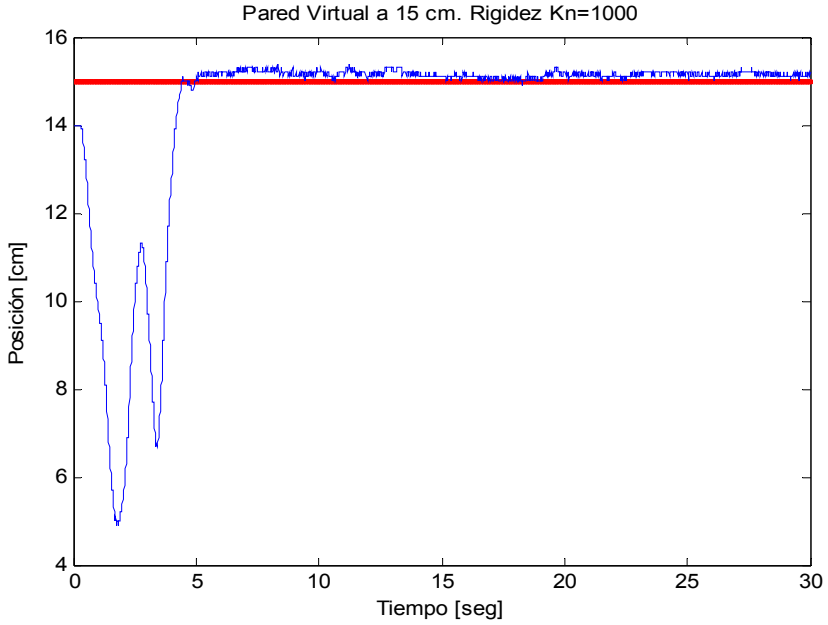

Figura 9.17 *Prueba 3 Modelo Elástico.*

Como se puede observar en las gráficas anteriores, cuando el valor de la rigidez (Kn) aumenta, el seguimiento de la pared es mucho más preciso. De igual forma, la penetración de la pared disminuye, pues el aumento de Kn provoca una respuesta en fuerza mayor cuando se presenta una colisión.

9.3.2 Modelo Viscoelástico

A continuación se presentan los resultados de las pruebas realizadas, empleando el modelo viscoelástico:

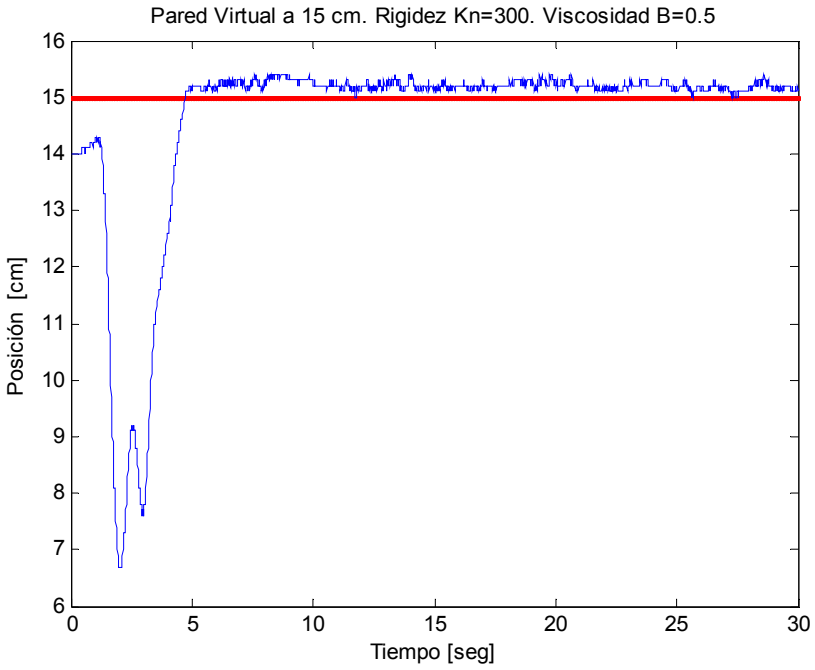

Figura 9.18 *Prueba 1 Modelo Viscoelástico.*

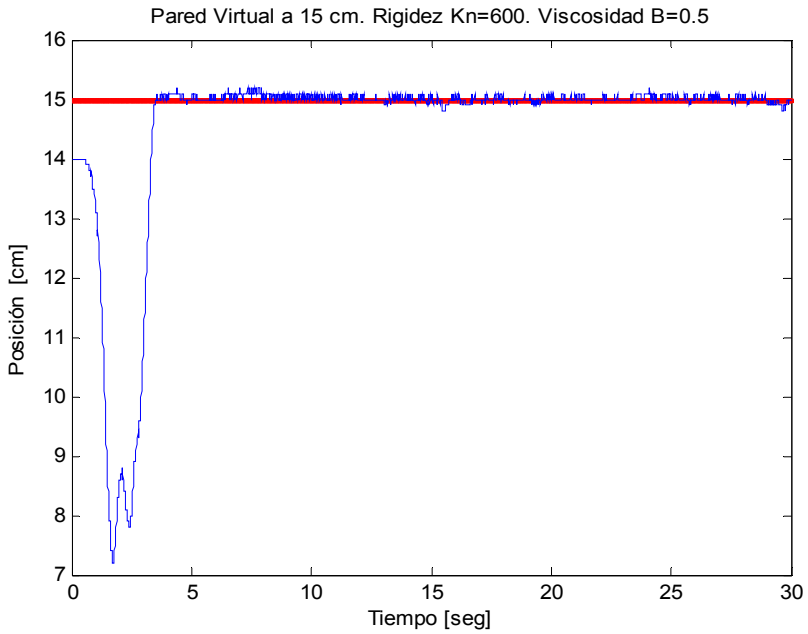

Figura 9.19 *Prueba 2 Modelo Viscoelástico.*

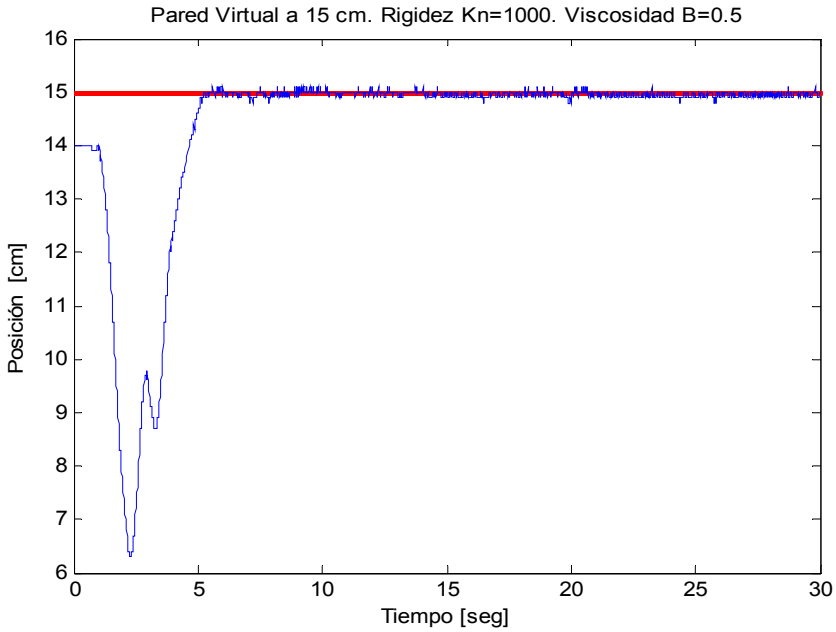

Figura 9.20 *Prueba 3 Modelo Viscoelástico.*

A partir de los resultados obtenidos, es posible inferir que la utilización del modelo viscoelástico permite mejorarla respuesta del dispositivo háptico cuando se presenta una colisión con la pared virtual.

10 CONCLUSIONES

A partir del trabajo investigativo realizado en este proyecto, fue posible inferir una serie de conclusiones que se presentan a continuación.

Con los resultados obtenidos se pudo comprobar que es posible optimizar el funcionamiento de la tarjeta EOS, utilizando conceptos de coprocesamiento de datos, implementados en hardware reconfigurable.

De esta forma, se ha desarrollado una nueva IP Core para la tarjeta EOS, mejorando el proceso de cálculo de funciones trigonométricas (seno y coseno), necesarias en la solución del problema cinemático y de reflexión de fuerzas para el control de interfaces hápticas.

Como parte final del proceso de desarrollo de la tarjeta EOS, se efectuó la validación de dicho dispositivo en el control de interfaces hápticas. Aplicando los modelos de contacto elástico y viscoelástico, en el modelo de control del PHANToM 1.0, se interactuó con una pared virtual para verificar el comportamiento del dispositivo en una sesión háptica.

De las pruebas realizadas con el PHANToM, se puede inferir que la tarjeta EOS resulta eficiente para implementar el control de este tipo de dispositivos, más aún cuando se empleó el modelo de contacto viscoelástico, lo que supone una complejidad considerable.

Teniendo en cuenta la frecuencia de muestreo mínima de 1 kHz, necesaria para que el usuario obtenga una sensación háptica realista, se pudo establecer que la tarjeta EOS utilizando la IPSC diseñada, cumple con suficiencia este requerimiento permitiendo frecuencias de muestreo de hasta 9 kHz.

Sin embargo también se pudo concluir que utilizando el MicroBlaze para efectuar todos los cálculos del lazo de control del sistema háptico, no se obtiene una respuesta satisfactoria, pues tan sólo se alcanza una frecuencia de muestreo máxima de 420 Hz, lo que resulta insuficiente para realizar sesiones hápticas adecuadas.

11 REFERENCIAS

Ashenden, Peter. The VHDL CookBook, University of Adelaide, Australia, 1990.

Brown, Stephen. Architecture of FPGAs and CPLDs Tutorial, Department of Electrical and Computer Engineering University of Toronto, 1996.

Çavuşoğlu, M. C. & Feygin, D. Kinematics and dynamics of PHANToM™ model 1.5 haptic interface (Tech. Rep.). University of California at Berkley, Electronics Research Laboratory Memo M01/15, 2001.

Craig, J.J. Introduction to Robotics:Mechanics and Control: International Edition 3rd Edition. Prentice Hall, 2004.

Embedded System Tools Reference Manual, *Embedded Development Kit EDK 8.1i,* 2005.

Gil, J. J. Control de Dispositivos Físicos de Gran Espacio de Trabajo para la Interacción Táctil con Entornos Virtuales, Tesis Doctoral, Universidad de Navarra, San Sebastián, España, 2003.

Goldsmith, W., Impact: The Theory and Physical Behaviour of Colliding Solids, Edward Arnold, London, 1960.

Hatwell, Yvette. Touching for Knowing: Cognitive Psychology of Haptic Manual Perception, Université René Descartes and Centre National de la Recherche Scientifique, Francia, 2000.

Holbein, Marc. A FPGA Haptics Controller, University of Guelph, Canada, 2005.

Hunt, K. H. y Crossley, F. R. E., “Coefficient of Restitution Interpreted as Damping in Vibroimpact”, ASME Journal of Applied Mechanics, pp. 440-445, Junio, 1975.

Hwang, J. D., Williams, M. D.,Niemeyer, G. (2004). "Toward Event-Based Haptics: Rendering Contact Using Open-Loop Force Pulses," in Proceedings of the Haptics Symposium 2004, Chicago, Illinois, pp. 27 - 28, March.

Johnson, K. L. "Contact Mechanics," Cambridge University Press, pp. 93, 1985.

Kurfess, Thomas. Robotics and Automation Handbook, CRC Press, United States of America, 2005.

Marschner, Alexander. An FPGA-based Target Acquisition System, Virginia Polytechnic Institute and State University, 2007.

Massie, T. H.,Salisbury, K. "The PHANTOM Haptic Interface: A Device for Probing Virtual Objects," in Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Chicago, IL, 1994.

Oliver, Juan. Utilización de FPGAs como Aceleradores de Cálculo, Universidad de la República, Facultad de Ingeniería Eléctrica, Uruguay, 2001.

Pearson, Martin. A Biomimetic Haptic Sensor, Intelligent Autonomous Systems laboratory, University of the West of England, Bristol, UK, 2006.

Pulido, Enrique. Bengoechea, Elixabete. Melo, Javier. Manual de Usuario para el Desarrollo de Software en la Tarjeta EOS, CEIT, 2007.

Rebello, Rohan. FPGA-Based High Performance Computing for Haptic Simulation in a Virtual Environment, University Chennai, India, 2004.

Rosenberg, L. B. y Adelstein, B. D., “Perceptual Decomposition of Virtual Haptic Surfaces”, Proceedings IEEE 1993 Symposium on Research Frontiers in Virtual Reality, San José, CA, pp. 46-53, Octubre, 1993.

Salcudean, S. E. y Vlaar, T. D., “On the Emulation of Stiff Walls and Static Friction with a Magnetically Levitated Input/Output Device”, Proceedings of the ASME Dynamic Systems and Control, Chicago, Illinois, vol. 1, pp. 303-309, Noviembre, 1994.

Siciliano, Bruno. Control of Interactive Robotics Interfaces, Springer, United States of America, 2006.

Tan, H., Srivasan, M., Eberman, B.,Cheng, B. "Human Factors for the Design of Force Reflecting Haptic Interfaces," in Proceedings of the ASME WAM, New York, vol. 55-1, pp. 353-360, 1994.

ZeidmanI, Bob. Introduction to FPGA Design, Embedded Systems Conference, 1999.

Zhao, Wei. FPGA Implementation of Closed-Loop Control System for Small-Scale Robot, Department of Electrical Engineering, University of Minnesota, Unites States of America, 2005.

ANEXO 1

PLANTILLA DE CÓDIGO VHDL GENERADA POR EL ASISTENTE DE CONFIGURACIÓN DEL XPS

```
------------------------------------------------------------------------------
-- Filename:          seno_coseno
-- Version:           1.00.b
-- Description:       Example FSL core (VHDL).
-- Date:              Fri May 30 13:21:09 2008 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                  "*_n"
--   clock signals:                       "clk", "clk_div#", "clk_#x"
--   reset signals:                       "rst", "rst_n"
--   generics:                            "C_*"
--   user defined types:                  "*_TYPE"
--   state machine next state:            "*_ns"
--   state machine current state:         "*_cs"
--   combinatorial signals:               "*_com"
--   pipelined or register delay signals:  "*_d#"
--   counter signals:                     "*cnt*"
--   clock enable signals:                "*_ce"
--   internal version of output port:     "*_i"
--   device pins:                         "*_pin"
--   ports:                               "- Names begin with Uppercase"
--   processes:                           "*_PROCESS"
--   component instantiations:            "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

------------------------------------------------------------------------------------------
--
--
-- Definition of Ports
-- FSL_Clk            : Synchronous clock
-- FSL_Rst            : System reset, should always come from FSL bus
-- FSL_S_Clk          : Slave asynchronous clock
-- FSL_S_Read         : Read signal, requiring next available input to be read
-- FSL_S_Data         : Input data
-- FSL_S_CONTROL      : Control Bit, indicating the input data are control word
-- FSL_S_Exists       : Data Exist Bit, indicating data exist in the input FSL bus
-- FSL_M_Clk          : Master asynchronous clock
-- FSL_M_Write        : Write signal, enabling writing to output FSL bus
-- FSL_M_Data         : Output data
```

```
-- FSL_M_Control   : Control Bit, indicating the output data are contol word
-- FSL_M_Full      : Full Bit, indicating output FSL bus is full
--
------------------------------------------------------------------------------

------------------------------------------------------------------------------
-- Entity Section
------------------------------------------------------------------------------

entity seno_coseno is
        port
        (
                -- DO NOT EDIT BELOW THIS LINE ----------------------
                -- Bus protocol ports, do not add or delete.
                FSL_Clk  : in       std_logic;
                FSL_Rst  : in       std_logic;
                FSL_S_Clk           : out       std_logic;
                FSL_S_Read          : out       std_logic;
                FSL_S_Data          : in        std_logic_vector(0 to 31);
                FSL_S_Control       : in        std_logic;
                FSL_S_Exists        : in        std_logic;
                FSL_M_Clk           : out       std_logic;
                FSL_M_Write         : out       std_logic;
                FSL_M_Data          : out       std_logic_vector(0 to 31);
                FSL_M_Control       : out       std_logic;
                FSL_M_Full          : in        std_logic
                -- DO NOT EDIT ABOVE THIS LINE ----------------------
        );

attribute SIGIS : string;
attribute SIGIS of FSL_Clk : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";

end seno_coseno;

------------------------------------------------------------------------------
-- Architecture Section
------------------------------------------------------------------------------

-- In this section, we povide an example implementation of ENTITY seno_coseno
-- that does the following:
--
-- 1. Read all inputs
-- 2. Add each input to the contents of register 'sum' which
--    acts as an accumulator
-- 3. After all the inputs have been read, write out the
--    content of 'sum' into the output FSL bus NUMBER_OF_OUTPUT_WORDS times
--
-- You will need to modify this example or implement a new architecture for
-- ENTITY seno_coseno to implement your coprocessor

architecture EXAMPLE of seno_coseno is
```

```
  -- Total number of input data.
  constant NUMBER_OF_INPUT_WORDS  : natural := 1;

  -- Total number of output data
  constant NUMBER_OF_OUTPUT_WORDS : natural := 2;

  type STATE_TYPE is (Idle, Read_Inputs, Write_Outputs);

  signal state       : STATE_TYPE;

  -- Accumulator to hold sum of inputs read at any point in time
  signal sum         : std_logic_vector(0 to 31);

  -- Counters to store the number inputs read & outputs written
  signal nr_of_reads  : natural range 0 to NUMBER_OF_INPUT_WORDS - 1;
  signal nr_of_writes : natural range 0 to NUMBER_OF_OUTPUT_WORDS - 1;

begin
  -- CAUTION:
  -- The sequence in which data are read in and written out should be
  -- consistent with the sequence they are written and read in the
  -- driver's seno_coseno.c file

  FSL_S_Read  <= FSL_S_Exists   when state = Read_Inputs   else '0';
  FSL_M_Write <= not FSL_M_Full when state = Write_Outputs else '0';

  FSL_M_Data <= sum;

  The_SW_accelerator : process (FSL_Clk) is
  begin  -- process The_SW_accelerator
   if FSL_Clk'event and FSL_Clk = '1' then     -- Rising clock edge
    if FSL_Rst = '1' then              -- Synchronous reset (active high)
      -- CAUTION: make sure your reset polarity is consistent with the
      -- system reset polarity
      state       <= Idle;
      nr_of_reads  <= 0;
      nr_of_writes <= 0;
      sum         <= (others => '0');
    else
      case state is
        when Idle =>
          if (FSL_S_Exists = '1') then
            state       <= Read_Inputs;
            nr_of_reads <= NUMBER_OF_INPUT_WORDS - 1;
            sum         <= (others => '0');
          end if;

        when Read_Inputs =>
          if (FSL_S_Exists = '1') then
            -- Coprocessor function (Adding) happens here
            sum         <= std_logic_vector(unsigned(sum) + unsigned(FSL_S_Data));
            if (nr_of_reads = 0) then
              state       <= Write_Outputs;
              nr_of_writes <= NUMBER_OF_OUTPUT_WORDS - 1;
```

```
            else
              nr_of_reads <= nr_of_reads - 1;
            end if;
          end if;

        when Write_Outputs =>
          if (nr_of_writes = 0) then
            state <= Idle;
          else
            if (FSL_M_Full = '0') then
              nr_of_writes <= nr_of_writes - 1;
            end if;
          end if;
      end case;
    end if;
  end if;
 end process The_SW_accelerator;
end architecture EXAMPLE;
```

ANEXO 2

CÓDIGO VHDL DE LA IP CORE DESARROLLADA PARA EL CÁLCULO DE SENO Y COSENO EN LA TARJETA EOS

```
----------------------------------------------------------------------------
-- Company:
-- Engineer:
----------------------------------------------------------------------------


package PARAMETERS is
        constant DATA_WIDTH         : natural := 32;
end package PARAMETERS;



library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.PARAMETERS.all;

----------------------------------------------------------------------------
-- Entity Section
----------------------------------------------------------------------------

entity seno_coseno is

port
        (
                -- DO NOT EDIT BELOW THIS LINE ----------------------
                -- Bus protocol ports, do not add or delete.
                FSL_Clk                 : in    std_logic;
                FSL_Rst                 : in    std_logic;
                FSL_S_Clk               : out   std_logic;
                FSL_S_Read              : out   std_logic;
                FSL_S_Data              : in    std_logic_vector(31 downto 0 );
                FSL_S_Control           : in    std_logic;
                FSL_S_Exists            : in    std_logic;
                FSL_M_Clk               : out   std_logic;
                FSL_M_Write             : out   std_logic;
                FSL_M_Data              : out   std_logic_vector(31 downto 0 );
                FSL_M_Control           : out   std_logic;
                FSL_M_Full              : in    std_logic
                -- DO NOT EDIT ABOVE THIS LINE ----------------------
        );
```

```
attribute SIGIS : string;
attribute SIGIS of FSL_Clk : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";


end seno_coseno ;

------------------------------------------------------------------------------
-- Architecture Section
------------------------------------------------------------------------------


architecture Behavioral of seno_coseno  is

component sen_cos
        port (
        THETA: IN std_logic_VECTOR(9 downto 0);
        SINE: OUT std_logic_VECTOR(31 downto 0);
        COSINE: OUT std_logic_VECTOR(31 downto 0));
end component;



  -- Total number of input data.
  constant NUMBER_OF_INPUT_WORDS  : natural := 1;

 -- Seno_coseno FSL states
  type STATE_TYPE is (Idle, Read_Inputs, Write_Outputs);

  signal state       : STATE_TYPE;

  -- Data to send to the master FSL
  signal data        :  std_logic_vector (31 downto 0);
  signal data2       :  std_logic_vector (31 downto 0);

  -- Counters to store the number inputs read & outputs written

  signal nr_of_writes : natural range 0 to 1;

        -- Señales de datos
        signal THETA                    : std_logic_vector (9 downto 0 );
        signal SINE                     : std_logic_vector (31 downto 0 );
        signal COSINE                   : std_logic_vector (31 downto 0 );

begin

        U1 : sen_cos
                port map (
                        THETA => THETA,
                        SINE => SINE,
                        COSINE => COSINE);
```

```
            FSL_S_Read  <= FSL_S_Exists   when state = Read_Inputs   else '0';
            FSL_M_Write <= not FSL_M_Full when state = Write_Outputs else '0';
            FSL_M_Data  <= data;

  The_accelerator : process (FSL_Clk) is
  begin  -- process The_SW_accelerator

   if FSL_Clk'event and FSL_Clk = '1' then     -- Rising clock edge
     if FSL_Rst = '1' then              -- Synchronous reset (active high)
       -- CAUTION: make sure your reset polarity is consistent with the
       -- system reset polarity
       state       <= Idle;
       data        <= (others => '0');
       THETA  <= (others => '0');

     else
       case state is
         when Idle =>
           if (FSL_S_Exists = '1') then
             state      <= Read_Inputs;
             data       <= (others => '0');

           end if;

         when Read_Inputs =>
           if (FSL_S_Exists = '1') then
             if ( FSL_S_Control = '1') then
                  THETA <= std_logic_vector(  unsigned(FSL_S_Data (9 downto 0)));
                  state <= Idle;
              else
                 data <=COSINE;
                 state       <= Write_Outputs;
                  nr_of_writes <= 1;
             end if;
         end if;

         when Write_Outputs =>
           if (nr_of_writes = 0) then
             state <= Idle;
           else
             if (FSL_M_Full = '0') then
                  data <=SINE;
                  nr_of_writes <= nr_of_writes - 1;

             end if;
           end if;
             end case;
     end if;
   end if;
  end process The_accelerator;

end Behavioral;
```

ANEXO 3

CONTROLADOR DE LA IP CORE PARA EL CÁLCULO DE SENO Y COSENO EN LA TARJETA EOS

Archivo Header (.h):

```
/*****************************************************************************
*****
* sine and cosine Header
* Ver   Who  Date     Changes
* ----- ---- -------- -----------------------------------------------
* 1.00a epl  25/05/06 First Release.
*
*****************************************************************************
*****/

#include "fsl.h"
#include "xparameters.h"


void cos_sin(float rad, float* out);
```

Archivo Source (.c):

```
/*****************************************************************************
*
* @file seno_coseno.c
*
* This file contains the drivers needed for controlling sine and cosine
through FSL
* bus.
*
*
* MODIFICATION HISTORY:
* <pre>
* Ver   Who  Date     Changes
* ----- ---- -------- -----------------------------------------------
* 1.00a epl  17/04/08 First Release.
*
/***************************** Include Files
#include "fsl.h"
#include "xparameters.h"
/************************** Constant Definitions
*****************************/

#define BUS_FSL     1         //ID of the fsl link with the sin_cos
```

```
/***************** Macros (Inline Functions) Definitions **************/
#define write_c_into_fsl(val, id)  cputfsl(val, id)
#define write_into_fsl(val, id)  putfsl(val, id)
#define read_from_fsl(val,id)       getfsl(val, id)

/**
* void cos_sin(int data, int* output) function
*
* This function calculates sine and cosine.
*
* @param    data is an angle to calculate sine and cosine. Output is a
pointer where is save the response.
*
*
* @return   None
*
* @note     None
*
*****************************************************************************
void cos_sin(float rad, float* out)
{
int sin_cos[2], theta;
float pi,tmp;
pi= 3.141592f;
rad = (rad*1024.0f)/(2.0f*pi);
theta= rad;
tmp=rad-theta;
theta=0.5f+tmp+theta;

      write_c_into_fsl(theta, BUS_FSL);
      write_into_fsl(1, BUS_FSL);
      read_from_fsl(sin_cos[0], BUS_FSL);
      read_from_fsl(sin_cos[1], BUS_FSL);

        if (sin_cos[0] >0x7FFFFFFF){

        out[0]= (float) -( 0x100000000 - sin_cos[0])/2147483648.0f;
        }
        else{
        out[0]=(float) sin_cos[0]/2147483648.0f;

        }
        if (sin_cos[1] >0x7FFFFFFF){

        out[1]=(float) -(0x100000000 - sin_cos[1])/2147483648.0f;
        }
        else{
        out[1]= (float)sin_cos[1]/2147483648.0f;
        }
}
```

ANEXO 4

PROGRAMA PARA EL CONTROL DEL PHANTOM UTILIZANDO LA TARJETA EOS

```
#include "xutil.h"
#include "xparameters.h"
#include "stdio.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "math.h"
#include "xspi.h"
#include "xspi_l.h"
#include "MCP23S17.h"
#include "fsl.h"
#include "ADCMAX1270.h"
#include "DACMAX5322.h"
#include "EncoderX6_FSL.h"
#include "seno_coseno.h"
#include "xtmrctr.h"
#include "xintc.h"


#define INIT_CREG_1 (XSP_CR_CLK_POLARITY_MASK | XSP_CR_TRANS_INHIBIT_MASK
| XSP_CR_MANUAL_SS_MASK | XSP_CR_MASTER_MODE_MASK)
#define INIT_CREG_0 (XSP_CR_TRANS_INHIBIT_MASK | XSP_CR_MANUAL_SS_MASK |
XSP_CR_MASTER_MODE_MASK)
#define TIMER_COUNT 75000   //1KHz de periodo a 75MHz
#define SAMPLE_FREQ 1000.0f  //frecuencia de muestreo

/*************************             Function              Prototypes
*****************************/
void timer_int_handler(void *baseaddr_p);
void PRINCIPAL (void);
void InicializacionParametros(void);
/*************************************************************************
*****/

//DEFINO PUNTEROS

XGpio Gpio;
XGpio_ex Gpio_ex;
XADC ADC;
XDAC DAC1;
XDAC DAC2;
XDAC DAC3;
float rad=0.0f;
float theta,theta1,theta2,theta3, E1=0.0f,B=0.5f;
```

```
unsigned int i;
int t=0;
float tiempo=0.0f;

float pi=3.141592f;

int encoder[3];
int a4,a5,a6,a7,a8,a9,a10,q1grad,q2grad,q3grad,s=0;
float q1, q2, q3, Kn=1000.0f, xh, yh, zh, fx, fy, fz, T1, T2, T3, T4,
l1=0.14f, l2=0.14f;
float senc1[2],senc2[2],senc3[2], senc23;

int main (void) {

PRINCIPAL();
return 0;
}


void PRINCIPAL(void){


// INICIALIZACION DE VARIABLES


rad=(320.0f*pi)/180.0f;
theta= (rad*1024.0f)/(2.0f*pi);


//INICIALIZACION DE PUERTOS

XGpio_Initialize(&Gpio,XPAR_GENERIC_GPIO_DEVICE_ID);

XGpio_SetDataDirection(&Gpio, 2, 0x00000000);

XGpio_SetDataDirection(&Gpio, 1, 0x00000000);


//INICIALIZACION DE SPI'S


//SPI 0
    XSpi_mSetControlReg(XPAR_OPB_SPI_0_BASEADDR, INIT_CREG_0);
   XSpi_mSetSlaveSelectReg(XPAR_OPB_SPI_0_BASEADDR, 0xF);
    XSpi_mEnable(XPAR_OPB_SPI_0_BASEADDR);


//SPI 1
    XSpi_mSetControlReg(XPAR_OPB_SPI_1_BASEADDR, INIT_CREG_0);
    XSpi_mSetSlaveSelectReg(XPAR_OPB_SPI_1_BASEADDR, 0xFF);
    XSpi_mEnable(XPAR_OPB_SPI_1_BASEADDR);
```

```
//SPI 2
    XSpi_mSetControlReg(XPAR_OPB_SPI_2_BASEADDR, INIT_CREG_0);
    XSpi_mSetSlaveSelectReg(XPAR_OPB_SPI_2_BASEADDR, 0xFF);
    XSpi_mEnable(XPAR_OPB_SPI_2_BASEADDR);


//INICIALIZO ADC'S

ADC_Init(&ADC, &Gpio,2, 0x0002, XPAR_OPB_SPI_1_BASEADDR);

//INICIALIZO DAC'S

DAC_Init(&DAC1, &Gpio,1, 0x0001, XPAR_OPB_SPI_1_BASEADDR);

DAC_Init(&DAC2, &Gpio,2, 0x0001, XPAR_OPB_SPI_1_BASEADDR);

DAC_Init(&DAC3, &Gpio,1, 0x0002, XPAR_OPB_SPI_1_BASEADDR);


//INICIALIZO I/O DIGITALES

XGpio_Init_Ex(&Gpio_ex, &Gpio, 0x10,XPAR_OPB_SPI_2_BASEADDR);

//DEFINO DIGITALES COMO SALIDA O ENTRADA


XGpio_Ex_SetDataDirection(&Gpio_ex,0,0x00);

XGpio_Ex_SetDataDirection(&Gpio_ex,2,0x00);


DAC_write(&DAC1, 0, 0);
    DAC_write(&DAC2, 0, 0);
    DAC_write(&DAC3, 0, 0);

XGpio_Ex_Write(&Gpio_ex,2, 0x00);


/* Enable microblaze interrupts */
    microblaze_enable_interrupts();

    /* Connect timer interrupt handler that will be called when an
interrupt
    * for the timer occurs
    */
    XIntc_RegisterHandler(                    XPAR_INTC_SINGLE_BASEADDR,
XPAR_OPB_INTC_0_OPB_TIMER_1_INTERRUPT_INTR,
    (XInterruptHandler)timer_int_handler,(void*)
XPAR_OPB_TIMER_1_BASEADDR);
```

```
    /* Start the interrupt controller */
    XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);

    /* set the number of cycles the timer counts before interrupting */
    XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_1_BASEADDR, 0, TIMER_COUNT);

    /* reset the timer and clean the interrupts */
    XTmrCtr_mSetControlStatusReg(      XPAR_OPB_TIMER_1_BASEADDR,      0,
XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK);

    /* Enable timer interrupts in the interrupt controller */
    XIntc_mEnableIntr(                          XPAR_INTC_SINGLE_BASEADDR,
XPAR_OPB_TIMER_1_INTERRUPT_MASK);

    /* Start the timer */
    XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_1_BASEADDR, 0,
    XTC_CSR_ENABLE_TMR_MASK  | XTC_CSR_ENABLE_INT_MASK |
    XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK );

EncoderZero(0);}
void timer_int_handler(void *baseaddr_p){

    unsigned int csr;

    int k=1;
    int sal;
float samples, f1,f2,aa,bb;

    /* Read timer 0 CSR to see if it raised the interrupt */
    csr = XTmrCtr_mGetControlStatusReg( XPAR_OPB_TIMER_1_BASEADDR, 0);


    EncoderRead(encoder);

    q1= encoder[0]/4.0f;
    q2= encoder[1]/4.0f;
    q3= encoder[2]/4.0f;

// Conversion a Coordenadas Articulares

    q1 = (float)((13.0f*2.0f*pi)*(q1/118.0f))/1000.0f;
    q2 = (float)10.0f*2.0f*pi*q2/1000.0f/76.0f;
    q3 = (float)(10.0f*2.0f*pi*q3/1000.0f/76.0f)-q2;



 cos_sin(q1,senc1);
 cos_sin(q2,senc2);
 cos_sin(q3,senc3);


    // Cinemática Directa:
```

```
    // Cinemática Directa sin aplicación de rotaciones a la herramienta

// xh=  -cos(t1)*cos(t2)*l2*sin(-t3+t2)+cos(t1)*sin(t2)*l2*cos(-
t3+t2)+cos(t1)*cos(t2)*l1;

// yh= -sin(t1)*cos(t2)*l2*sin(-t3+t2)+sin(t1)*sin(t2)*l2*cos(-
t3+t2)+sin(t1)*cos(t2)*l1;

// zh= -sin(t2)*l2*sin(-t3+t2)-cos(t2)*l2*cos(-t3+t2)+sin(t2)*l1+l2 ;


// Cinemática Directa con rotaciones:

        xh=
(float)(senc1[1]*(senc2[0]*(senc3[1]*0.14f+0.14f)+senc2[1]*senc3[0]*0.14f
));
        yh=                        (float)(senc2[1]*(senc3[1]*0.14f+0.14f)-
senc2[0]*senc3[0]*0.14f+0.14f);
        zh=
(float)(senc1[0]*(senc2[0]*(senc3[1]*0.14f+0.14f)+senc2[1]*senc3[0]*0.14f
));




    a4=xh*1000;
    a5=yh*1000;
    a6=zh*1000;

    //fx,fy,fx fuerza en newtons

    // Supongamos una pared virtual a 15cm en el eje x

    fx = 0;
    fy = 0;
    fz = 0;

    // Modelo Viscoelástico

    if (zh >= 0.150){
     fz = Kn*(0.150f-zh)+ B*(((0.150f-zh)-E1)/0.001f);
     E1=0.150f-zh;
    } else {
     fz = 0.0;
    }


    // Modelo Elástico

//if (zh >= 0.150){
//   fz = Kn*(0.150f-zh);
//
//  } else {
```

```
//   fz = 0.0;
//  }
//


    // Jacobiana Transpuesta



    T1 = (float) 0.1102f*(senc1[0]*( 0.14f*senc2[0] + 0.14f*senc3[1] )*fx
- senc1[1]*(0.14f*senc2[0]+0.14f*senc3[1])*fz);

    T2    =    (float)    0.1316f*(    -senc1[1]*0.14f*senc2[1]*fx    +
0.14f*senc2[0]*fy - senc1[0]*0.14f*senc2[1]*fz);

    T3 = (float) 0.1316f*( senc1[1]*0.14f*senc3[0]*fx + 0.14f*senc3[1]*fy
+ senc1[0]*0.14f*senc3[0]*fz);


    // Conversión a Amperios

    T1=T1*42.553f;
    T2=T2*42.553f;
    T3=T3*42.553f;


    // Limitacion de Corriente

    if ((T1 || T2 || T3) >= 3){
    T1=T2=T3=0;

    }

    // Conversión a Voltios

    T1=T1*3.33f;
    T2=T2*3.33f;
    T3=T3*3.33f;

    //Envio Voltaje a los motores

    DAC_write(&DAC1, 0, T1);
    DAC_write(&DAC2, 0, T2);
    DAC_write(&DAC3, 0, T3);

    XTmrCtr_mSetControlStatusReg( XPAR_OPB_TIMER_1_BASEADDR, 0, csr);
    }
```

ANEXO 5

```
clc;
clear all;
b1=link([pi/2 0 0 0.14], 'standart')
b2=link([0 0.14 0 0], 'standart')
b3=link([0 0.14 -pi/2 0], 'standart')

r=robot({b1 b2 b3})

plot(r,[0 0 0])

syms t1 t2 t3 l1 l2;

a01= [cos(t1) 0 sin(t1) 0;
       sin(t1) 0 -cos(t1) 0;
       0 1 0 l2;
       0 0 0 1];

a12= [cos(t2) -sin(t2) 0 l1*cos(t2);
      sin(t2) cos(t2) 0 l1*sin(t2);
      0 0 1 0;
      0 0 0 1
    ];

a23 = [cos(3/2*pi+t3-t2) -sin(3/2*pi+t3-t2) 0 l2*cos(3/2*pi+t3-t2);

    sin(3/2*pi+pi+t3-t2) cos(3/2*pi+t3-t2) 0 l2*sin(3/2*pi+t3-t2);
    0 0 1 0;
    0 0 0 1
        ];

t=a01*a12*a23

%%%%%%%%%%RESULTADO::
%
% b1 =
%
%   1.570796    0.000000    0.000000    0.140000    R   (std)
%
%
% b2 =
%
%   0.000000    0.140000    0.000000    0.000000    R   (std)
%
%
% b3 =
%
%   0.000000    0.140000    -1.570796   0.000000    R   (std)
%
```

```
%
% r =
%
% noname (3 axis, RRR)
%       grav = [0.00 0.00 9.81]     standard D&H parameters
%
%   alpha         A        theta        D      R/P
% 1.570796  0.000000    0.000000    0.140000    R   (std)
% 0.000000  0.140000    0.000000    0.000000    R   (std)
% 0.000000  0.140000    -1.570796   0.000000    R   (std)
%
%
% t =
%
% [                           -cos(t1)*cos(t2)*sin(-t3+t2)-
cos(t1)*sin(t2)*cos(-t3+t2),
cos(t1)*cos(t2)*cos(-t3+t2)+cos(t1)*sin(t2)*sin(-t3+t2),
sin(t1), -cos(t1)*cos(t2)*l2*sin(-t3+t2)+cos(t1)*sin(t2)*l2*cos(-
t3+t2)+cos(t1)*l1*cos(t2)]
% [                           -sin(t1)*cos(t2)*sin(-t3+t2)-
sin(t1)*sin(t2)*cos(-t3+t2),
sin(t1)*cos(t2)*cos(-t3+t2)+sin(t1)*sin(t2)*sin(-t3+t2),
-cos(t1), -sin(t1)*cos(t2)*l2*sin(-t3+t2)+sin(t1)*sin(t2)*l2*cos(-
t3+t2)+sin(t1)*l1*cos(t2)]
% [                                          -sin(t2)*sin(-
t3+t2)+cos(t2)*cos(-t3+t2),
sin(t2)*cos(-t3+t2)-cos(t2)*sin(-t3+t2),
0,                        -sin(t2)*l2*sin(-t3+t2)-cos(t2)*l2*cos(-
t3+t2)+sin(t2)*l1+l2]
% [
0,
0,
0,
1]
%
```