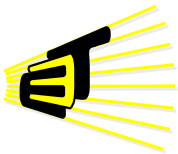
Diseño e implementación, sobre un FPGA, de un sistema reconfigurable dinámicamente instalado como un nodo de un cluster.

Ing. William Alexánder Salamanca Becerra



Escuela de Ingenierías
Eléctrica, Electrónica
y de Telecomunicaciones

Universidad Industrial de Santander
Facultad de Ingenierías Físico-Mecánicas
Escuela de Ingeniería Eléctrica, Electróncia y de Telecomunicaciones
Bucaramanga
Marzo, 2011

Diseño e implementación, sobre un FPGA, de un sistema reconfigurable dinámicamente instalado como un nodo de un cluster.

Ing. William Alexánder Salamanca Becerra

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

Magister en Ingeniería Electrónica

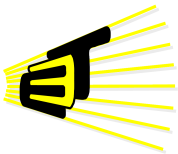
Tesis desarrollada dentro del convenio de cooperación UIS-ICP 005 de 2003

Directora:

MSc. Ana Beatríz Ramírez Silva

Co-Director:

PhD. William Mauricio Agudelo Zambrano



Escuela de Ingenierías
Eléctrica, Electrónica
y de Telecomunicaciones

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingeniería Eléctrica, Electróncia y de Telecomunicaciones

Bucaramanga

Marzo, 2011

Agradecimientos

Este trabajo investigativo ha llegado a feliz término y el mérito no es solo mío sino también de todas las personas que me apoyaron durante todo su desarrollo.

En primer lugar, agradezco a Dios, a mis padres y a mi hermano, a quienes dedico este trabajo por su apoyo infinitamente incondicional. Quiero resaltar la dedicación y diposición que tuvo mi directora de tesis, la Profesora Ana Beatriz Ramírez, a la cual le deseo muchos éxitos en sus proyectos. El apoyo incondicional y la confianza que tuvo el PhD. William Mauricio Agudelo, mi codirector, quien desde un principio creyó en esta idea y en nosotros para llevarla a cabo.

Al Profesor Jorge Ramón que me ha acompañado y apoyado en todo mi proceso de formación que aun sigue en desarrollo. Al grupo de investigación CPS, quienes nos acogieron y brindaron el espacio para desarrollar el presente trabajo. A todos los compañeros de oficina, con quienes hemos podido forjar buenas amistades y un buen ambiente que favoreció el desarrollo del proyecto.

Al grupo de investigación Petrosísmica por respaldar la idea en cabeza de sus interventores el Profesor Jorge Eduardo Pinto, William Mauricio Agudelo y Andrés Calle. A los compañeros del grupo que han mostrado interés, nos han dado apoyo y han hecho sus observaciones en su debido momento.

A todos mis amigos, que me han acompañado y apoyado, les quiero compartir este momento y este trabajo.

Índice

Lista de Figuras

Lista de Tablas

Resumen

TITULO: DISEÑO E IMPLEMENTACIÓN, SOBRE UN FPGA, DE UN SISTEMA RECONFIGURABLE DINÁMICAMENTE INSTALADO COMO UN NODO DE UN CLUSTER.*

AUTOR: WILLIAM ALEXANDER SALAMANCA BECERRA**

PALABRAS CLAVE: FPGA, HPRC, Computación Reconfigurable, HPC.

La tecnología de fabricación de circuitos integrados mejora día a día, y esto se ve reflejado en la capacidad de cómputo de las plataformas basadas en procesadores de propósito general (PPG). Así mismo, este avance favorece otros circuitos digitales como los Field Programmable Gate Arrays (FPGAs) y los hace competitivos como dispositivos de cómputo. Su desempeño ha sido evidenciado en aplicaciones donde son empleados para implementar procesadores de propósito específico (PPE) y logran acelerar procesos de cómputo hasta 1700 veces respecto a PPG.

La Computación Reconfigurable de Alto Rendimiento (HPRC) propone un nuevo paradigma de computación basada en la combinación de FPGAs y PPG con muy buenas expectativas debido a que la rata de crecimiento de la capacidad de cómputo de los FPGAs es muy superior a la que han tenido los PPGs. Teniendo en cuenta esto, empresas como el Instituto Colombiano de Petróleo (ICP), se han interesado en el tema y han apoyado investigaciones que permitan finalmente llevar a cabo sus algoritmos de procesamiento de datos en un menor tiempo.

El presente proyecto de maestría realizó un avance significativo hacia la construcción de una plataforma económica de HPRC, implementando un cluster heterogéneo compuesto por PPG y FPGAs. Adicionalmente se implementó un mecanismo que permite reconfigurar parcialmente los recursos lógicos del FPGA lanzando una aplicación desde el sistema operativo instalado en el FPGA o desde el cluster mediante MPI o PVM.

De esta forma, se pudo medir el tiempo de reconfiguración de los recursos lógicos y así establecer las características de las aplicaciones que se verían favorecidas al ser implementadas sobre este tipo de plataformas.

* Trabajo de investigación

** **Facultad:** Facultad de Ingenierías Físico Mecánicas. **Escuela:** Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. **Grupo de Investigación:** CPS. **Director:** PhD(c) Ana Beatríz Ramírez Silva

Abstract

TITLE: DESIGN AND IMPLEMENTATION, OVER AN FPGA, OF A DYNAMICALLY RECONFIGURABLE SYSTEM INSTALLED AS A NODE CLUSTER. *

AUTHOR: WILLIAM ALEXANDER SALAMANCA BECERRA**

KEY WORDS: FPGA, HPRC, Reconfigurable Computing, HPC.

The integrated circuit fabrication technology is improving with time and it can be observed in the performance of the latest computing platforms based on General Purpose Processors (PPG). Furthermore, it provides new benefits to other digital circuits such as Field Programmable Gate Arrays (FPGAs) making them more competitive as computing devices. FPGAs have been used in applications where Specific Purpose Processors (PPE) can be implemented and it has been demonstrated that they can reach accelerations up to 1700 times compared to PPG.

High Performance Reconfigurable Computing (HPRC) propose a new paradigm in computation based on the combination of FPGAs and PPGs with high expectations due to the computing performance growth rate of the FPGAs has been greater than PPGs. Bearing in mind this, companies, as the Colombian Petroleum Institute (ICP), are interested in this topic and they have supported investigations in HPRC that can execute their algorithms for data processing in less time.

This master thesis makes a significant progress to the construction of an inexpensive HPRC platform. In this work the implementation of a heterogeneous cluster with PPGs and FPGAs is proposed. In addition, a method to partially reconfigure the FPGA's logic resources is implemented by deploying an application from the FPGA's operating system or from the cluster through MPI or PVM.

Finally, the logic resources reconfiguration time was measured in the proposed heterogeneous cluster and characteristics of the applications that can be beneficed with the implementation in this type of computing platform were established.

* Trabajo de investigación

** **Faculty:** Physico-mechanical Engineering Faculty. **School:** School of Electrical, Electronics and Telecommunications Engineering. **Research group:** CPS. **Director:** PhD(c) Ana Beatríz Ramírez Silva

Introducción

La computación de alto rendimiento (CAR)[1], hasta hace poco, se ha basado principalmente en procesadores de propósito general (PPG). Los *clusters* y grid, implementados para formar plataformas con las más grandes capacidades de cómputo, se fundamentan en la agrupación de este tipo de procesadores. Por otro lado los procesadores de propósito específico (PPE), gran parte de estos implementados sobre FPGAs, se han caracterizado por obtener un desempeño superior a los PPG al llevar a cabo la función para la cual son diseñados. Las plataformas basadas en PPE han reportado en la literatura factores de aceleración que van desde 1 hasta varios miles de veces, comparado con plataformas basadas en PPG[5]. Esto le ha permitido a los FPGA introducirse fuertemente como una alternativa entre los dispositivos de cómputo disponibles.

Además de la posibilidad de configurar los FPGAs como PPE, existe la posibilidad de reconfigurarlos durante el tiempo de ejecución de un proceso. Esto ha abierto las puertas de un nuevo paradigma en computación que involucra el uso de diferentes PPE configurados en diferentes instantes de tiempo sobre los mismos recursos de un FPGA. Basados en este principio, los sistemas reconfigurables(RS) y los sistemas de computación reconfigurable de alto rendimiento (HPRC) han surgido como las nuevas arquitecturas de cómputo.

Aunque estas arquitecturas tienen mucho potencial, han tenido varias dificultades en su proceso de acogida por parte de los desarrolladores debido principalmente a dos características: el conocimiento sobre implementación de *hardware* requerido por parte del desarrollador y el tiempo de desarrollo significativamente alto para una aplicación. Por otro lado, aunque los fabricantes de los FPGAs han dado mecanismos para llevar a cabo la reconfiguración, solo actualmente se está comenzando a dar el soporte técnico y las herramientas *software* para llevar a cabo este tipo de aplicaciones mediante licencias comerciales.

[1] en inglés *High Performance Computing (HPC)*

El presente proyecto realiza un aporte en el avance hacia la implementación de una plataforma experimental, económica y flexible para HPRC con el objetivo de permitir el desarrollo de nuevos trabajos y estimar el impacto que este paradigma de computación reconfigurable puede tener.

Descripción del proyecto

Este trabajo es la base para la implementación de una plataforma que permite llevar a cabo aplicaciones HPRC, la cual ofrece los mecanismos para reconfigurar un FPGA en un entorno de computación paralela. Los sistemas HPRC se caracterizan por ser una combinación de PPGs y FPGAs interconectados por una red de datos. Según como éstos se organicen en los nodos de la red, se pueden clasificar como *Uniform Nodes Non-uniform Systems* (UNNS) y *Non-uniform Nodes Uniform System* (NNUS). En los sistemas UNNS se encuentra en cada uno de los nodos un solo tipo de procesador, es decir FPGA o PPG, mientras que los sistemas NNUS se caracterizan por tener todos los nodos homogéneos con igual cantidad de PPG y de FPGAs en cada uno de ellos. [6]

El sistema HPRC que se realizó, está representado en la Figura 0.1. En esta se observa la interconexión de PPG y sistemas de desarrollo ML507 de FPGAs mediante una red de datos ethernet formando un sistema tipo UNNS. Se seleccionó esta arquitectura porque sigue la filosofía de los *cluster beowulf*, lo cual lo hace flexible durante su implementación facilitando la adición de nodos. Así mismo el usuario puede decidir fácilmente la cantidad de FPGAs y PPG que desea usar para llevar a cabo la aplicación.

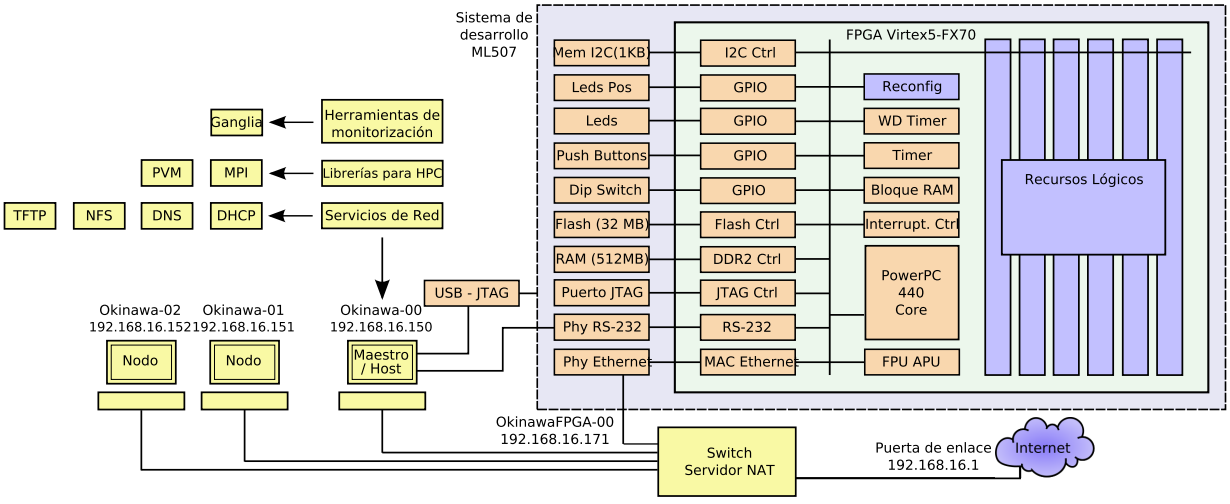


Figura 0.1: Diagrama general

El proyecto se desarrolló mediante la implementación en el FPGA de un *embedded system* basado en

un procesador PowerPC®, el cual tiene las funciones de un sistema reconfigurable y de un nodo del *cluster* heterogéneo formado con los PPG. Como sistema reconfigurable, el PowerPC®puede modificar el estado de sus recursos lógicos para implementar periféricos que le pueden ayudar a efectuar funciones de cálculo intensivo; y como nodo del *cluster*, el PowerPC®puede recibir carga e interactuar dentro de un entorno de computación paralela.

La implementación del sistema se realizó según el diagrama mostrado en la Figura 0.2. Inicialmente se creó el sistema base de la plataforma de cómputo, es decir el PowerPC® con sus periféricos básicos interconectados por el bus PLB, mediante la herramienta EDK del fabricante del FPGA, Xilinx® (1). Posteriormente se procedió a instalar un *bootloader* que permitiera al sistema iniciar un sistema operativo (2). Para que el sistema operativo pudiera correr, fue necesario compilar el *kernel*, preparar un sistema de archivos NFS, además de configurar otros servicios de red (3). Al mismo tiempo, se instaló un *cluster* de PPG que al ser integrado con el FPGA, se convirtió en un *cluster* heterogéneo formado por FPGAs y PPG (4).

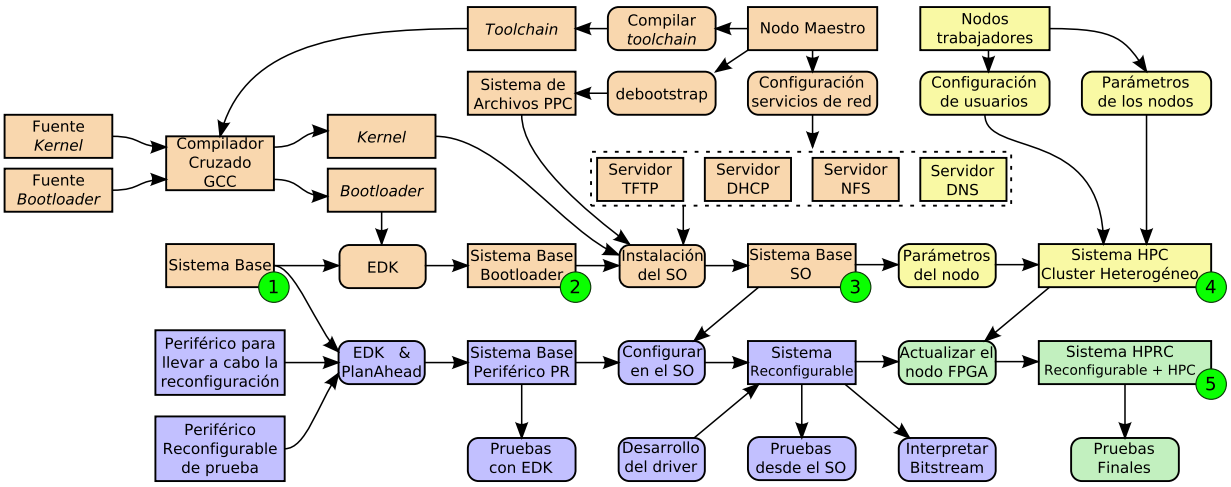


Figura 0.2: Pasos de la implementación del sistema.

El siguiente paso consistió en permitir que el *embedded system* tenga la capacidad de reconfigurar parcialmente sus recursos lógicos Para ello, se implementó un periférico basado en el módulo *Internal Configuration Access Port* (ICAP) del FPGA y se desarrolló un *driver* para el sistema operativo que permitiera administrar el periférico y los recursos lógicos bajo demanda por parte de los procesos que se ejecutan en el procesador PowerPC®. Y de esta forma el sistema pudo recibir desde el *cluster* una orden para reconfigurar sus recursos lógicos, con lo que se finalizó la implementación del sistema HPRC. (5).

Finalmente se estableció un procedimiento para determinar el correcto funcionamiento y medir el desempeño de la reconfiguración para establecer el impacto que tiene sobre una aplicación.

Mapa del documento

Este documento recopila el proceso de diseño, las experiencias y la implementación del sistema HPRC descrito en la sección anterior. Está distribuido en capítulos de la siguiente forma:

Capítulo 1: En este capítulo se presenta el proceso de implementación del *embedded system* sobre el FPGA, basado en el procesador PowerPC®, desde su implementación en EDK hasta la instalación y configuración del sistema operativo.

Capítulo 3: En este capítulo se introducen los conceptos básicos de la configuración del FPGA, de la arquitectura del FPGA y los archivos de configuración. Además como resumen de la documentación del fabricante y del proceso de ingeniería inversa, se presenta un script en matlab que interpreta un archivo de configuración.

Capítulo 4: En este capítulo se documenta todo el proceso de adición del periférico que permite hacer la reconfiguración y su puesta en marcha desde el entorno de EDK y el entorno del sistema operativo. Adicionalmente se presenta una aplicación desarrollada que permitió determinar y validar exactamente cuales bits del *bitstream* modifican el comportamiento de las LUT del FPGA.

Capítulo 5: Finalmente en este capítulo se recopilan las conclusiones a las que se llegó después de llevar a cabo esta implementación y la continuidad que se le puede dar a esta investigación en el futuro.

Capítulo 1

Implementación del *Embedded System* Sobre el FPGA

A lo largo del presente capítulo, se presentarán los detalles de la implementación del *embedded system*. La Figura 1.1 muestra los bloques del sistema que están involucrados en este capítulo, mientras la Figura 1.2 muestra el avance de esta implementación dentro del proceso general. A continuación se exponen algunos detalles de la plataforma y las herramientas empleadas y posteriormente se presentan detalles sobre la implementación del sistema.

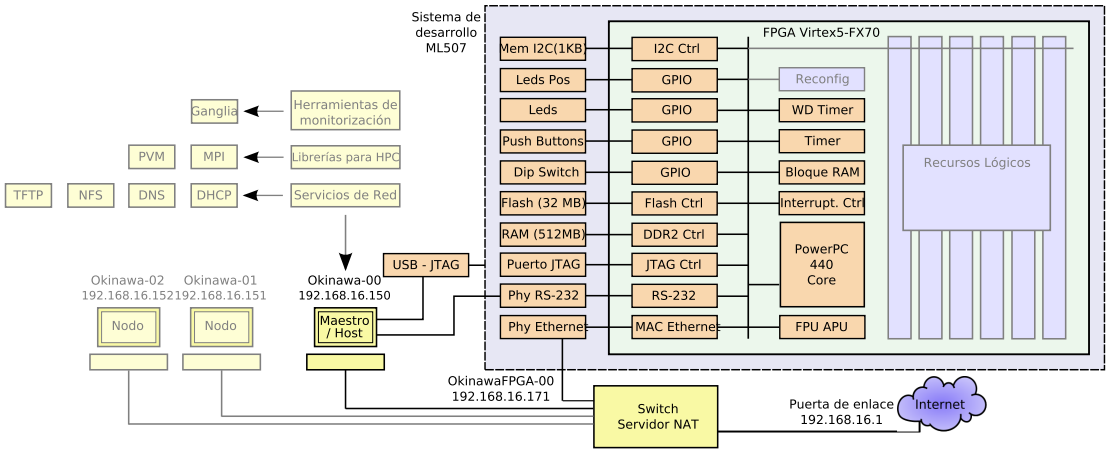


Figura 1.1: Diagrama general

1.1 Sistema de desarrollo ML507

Para llevar a cabo el proyecto se seleccionó el sistema de desarrollo ML507, el cual es una plataforma de propósito general para aplicaciones que involucran FPGA, RocketIO GTX o *embedded system*[2]. Esta

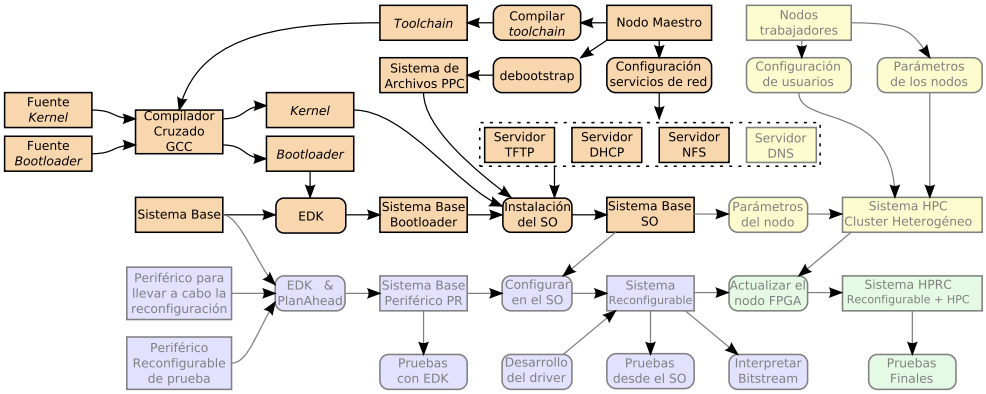


Figura 1.2: Pasos de la implementación del sistema.

plataforma tiene como componente principal el FPGA de Xilinx® `XC5VFX70TFFG1136` de la familia Virtex-5, cuyos recursos se resumen en la Tabla 1.1. Esta familia de FPGA está orientada hacia el desarrollo de *embedded system* con comunicaciones seriales de alta velocidad, por lo que dispone entre sus recursos de un núcleo del procesador PowerPC®y módulos RocketIO GTX. La Tabla 1.1 adicionalmente, dimensiona la cantidad de recursos del FPGA seleccionado al compararlo respecto al miembro más grande y al más pequeño de la subfamilia FX de la Virtex-5.

Dispositivo	Configurable Logic Blocks			Slices DSP48E	Bloques RAM		CMTs	PowerPC Blocks	Ethernet MACs	Max User I/O
	Arreglo FilxCol	V5 Slices	RAMD (Kb)		36 Kb	Max (Kb)				
XC5VFX30T	80 x 38	5120	380	64	68	2448	2	1	4	360
XC5VFX70T	160 x 38	11200	820	128	148	5328	6	1	4	640
XC5VFX200T	240 x 68	30720	2280	384	456	16416	6	2	8	960

Tabla 1.1: Recursos del FPGA (Adaptado de [4])

El diagrama de bloques del sistema ML507 se presenta en la Figura 1.3. En él se resumen los recursos del sistema y se resaltan los componentes utlizados en la realización del proyecto. Cabe aclarar que debido a que la tarjeta es un sistema de desarrollo, no todos sus recursos son empleados, pero aplicaciones futuras pueden hacer uso de ellos.

Como dispositivo central se encuentra el FPGA, el cual dispone de recursos lógicos con los que se implementó el sistema base y la sección reconfigurable. Como elementos de almacenamiento permanente, se empleó la memoria *Platform Flash* que almacenó el archivo de configuración del FPGA con el sistema base; la memoria *Flash* que contiene el ejecutable del bootloader y podría ser usada para almacenar el *kernel* del sistema operativo; y la memoria I2C que almacenó los parámetros de `u-boot` del arranque del sistema. La implementación de la memoria RAM DDR2 y la interfaz de ethernet, complementaron el sistema para permitir la ejecución del sistema operativo y el acceso a los servicios de red. El puerto RS-232

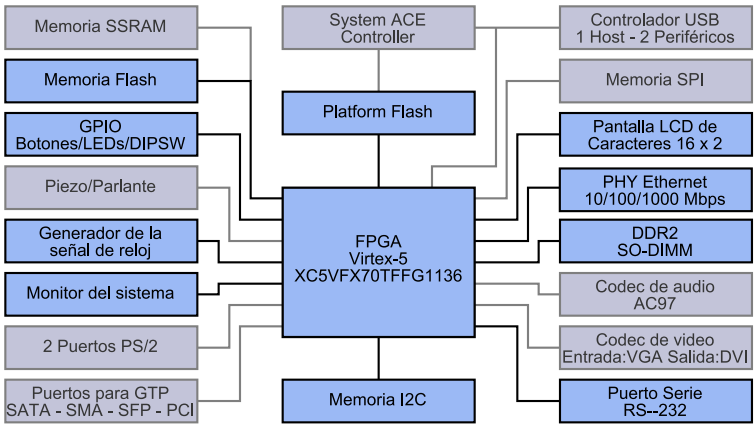


Figura 1.3: Diagrama de bloques del sistema de desarrollo ML507 (Adaptado de [2])

permitió implementar la entrada y salida estándar por consola en un terminal, y los demás periféricos como los GPIO, y la pantalla LCD permiten desplegar información del sistema o hacer pruebas.

1.2 *Integrated Software Environment* (ISE)

El entorno de trabajo *Integrated Software Environment* (ISE) es un conjunto de herramientas *software* que permite llevar hacer el diseño de un circuito desde el ingreso de las fuentes hasta su implementación sobre un dispositivo FPGA o CPLD. Estas herramientas se acceden por medio de una interfaz gráfica que ilustra el flujo de diseño de un circuito digital, además permite administrar y acceder a los archivos intermedios que se generan durante el proceso como por ejemplo los reportes de cada una de las etapas. La administración de los archivos permite manejar proyectos que comprometen gran cantidad de archivos fuentes o jerarquías en el diseño. Para entender mejor las funciones de la herramienta que aplican al proyecto, es necesario comprender el flujo de diseño de circuitos sobre FPGA.

1.2.1 Flujo de diseño tradicional con FPGA (Tomado de [1])

La metodología para la implementación de un circuito en un FPGA Xilinx® está bien definida por el fabricante y se puede resumir en el diagrama de la Figura 1.4. En ella se pueden observar los procesos y los archivos intermedios en el proceso desde que se ingresa los archivos fuente hasta la generación del archivo de configuración del dispositivo. Estos procesos se pueden agrupar en tres secciones: la síntesis, la implementación y la validación. Para implementar un diseño es necesario llevar a cabo solo las dos primeras etapas, por lo tanto el texto profundizará únicamente en estas dos secciones.

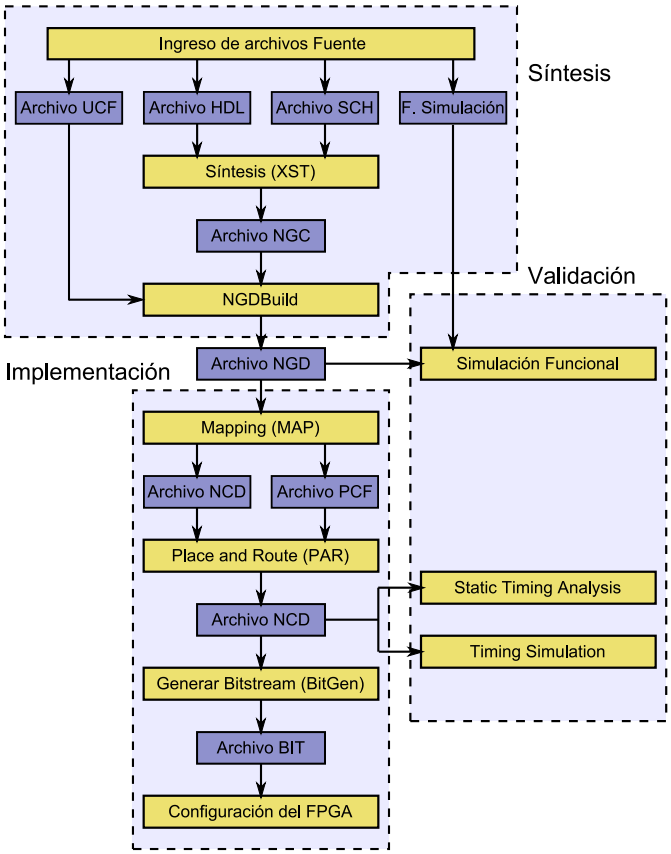


Figura 1.4: Flujo de diseño con FPGA (Adaptado de [1])

1.2.1.1 Ingreso de archivos fuente

En esta etapa, el diseñador ingresa o crea los archivos fuentes del proyecto. Existen diferentes tipos de archivos fuente, según la etapa de diseño a la cual afectan. Los archivos que contienen la información del funcionamiento o interconexión del circuito que se va a implementar se pueden ingresar en modo esquemático (gráfico) o por medio de un lenguaje de descripción de *hardware* (HDL). Los archivos que imponen restricciones a las herramientas *software* se pueden ingresar en modo texto o en modo gráfico por medio de una interfaz gráfica. Estas restricciones pueden estar relacionadas con la ubicación de las partes del diseño en el FPGA o restricciones de tiempos o frecuencias de trabajo de algunas señales. Finalmente algunas fuentes están orientadas a la simulación del diseño a diferentes niveles; éstas fuentes se pueden ingresar por medio de un lenguaje de HDL o mediante una interfaz gráfica otorgada por el simulador de ISE.

1.2.1.2 Síntesis

Una vez ingresados los archivos fuente, la herramienta de síntesis los interpreta y crea a partir de ellos un archivo de objetos genéricos que contiene la información de los componentes e interconexiones del circuito a implementar. Este proceso se lleva a cabo en dos etapas: la primera la lleva a cabo la herramienta XST que crea un *netlist* del circuito (archivo NGC) y posteriormente la herramienta `NGDBuild` crea una base de datos (archivo NGD) que contiene la descripción lógica del circuito y su jerarquía.

1.2.1.3 *Mapping*

Durante esta etapa se hace una asociación de las primitivas[1] de Xilinx® que se encuentran en el archivo NGD a elementos propios del FPGA de Xilinx® sobre el cual se va a implementar el circuito. Uno de los archivos de salida es de extensión NCD, el cual es una descripción física del diseño en función de elementos del FPGA a emplear; el otro es de extensión PCF, el cual incluye una traducción a restricciones físicas de las restricciones del archivo UCF ingresado por el usuario. Este proceso de *mapping* puede variar considerablemente cuando implementa el circuito sobre una u otra familia de FPGA.

1.2.1.4 *Place and Routing*

Esta etapa busca seleccionar los recursos que se van a emplear para la implementación del diseño y realizar la interconexión que el diseño exija para su correcto funcionamiento. Esta etapa se puede llevar a cabo de forma automática y manipulada por medio de restricciones o manualmente por medio de las herramientas FPGA Editor y PlanAhead.

1.2.1.5 Generación del archivo de configuración

Finalmente una vez seleccionados los recursos a usar, se procede a crear el archivo de configuración que contiene la información para ajustar cada uno de los elementos a usar en el FPGA y su interconexión con los demás. Con este archivo es posible configurar el FPGA o crear un archivo a partir de él para programar una memoria de configuración externa al FPGA.

1.3 *Embedded Development Kit* (EDK)

El entorno de trabajo *Embedded Development Kit* (EDK), permite crear sistemas de cómputo basados en los procesadores PowerPC®y MicroBlaze. Mediante su interfaz gráfica se puede manipular tanto el *software* que se ejecuta en el procesador como la interconexión de los elementos del sistema, los buses, interrupciones y demás elementos del *hardware*. La implementación de estos sistemas se realiza en FPGAs

[1] La palabra primitiva será usada como traducción del término del idioma inglés *primitive* que hace referencia a los elementos lógicos básicos que componen un FPGA y que pueden aparecer como instancias en una descripción HDL.

de Xilinx® , por lo EDK hace uso de las herramientas de síntesis e implementación de ISE. Además de las herramientas para la implementación del diseño, EDK permite realizar la simulación del sistema o la depuración paso a paso del programa mientras se ejecuta en el procesador.

En la interfaz de EDK, el proceso de implementación es presentado al usuario en forma de menú, donde se llevan a cabo secuencialmente un conjunto de órdenes que están representadas en la Figura 1.5. Cada una de las órdenes involucra la ejecución de programas de ISE y algunas otras aplicaciones complementarias. Durante las siguientes secciones del documento se ampliará lo que sucede en cada una de las etapas del proceso.

1.3.1 *Hardware > Generate Netlist*

Durante esta etapa, EDK se encarga de sintetizar cada uno de los módulos del diseño, es decir, cada periférico, interfaz de bus, procesador, controlador de memoria, etc. Para ello usa la herramienta `xst` y de cada uno de ellos, se genera un archivo netlist NGC; así mismo se genera un archivo NGC del sistema completo que contiene únicamente cajas negras interconectadas.

1.3.2 *Hardware > Generate Bitstream*

En esta etapa, se realiza toda la implementación del sistema, donde se emplean las herramientas de Xilinx® `NGDBuild`, `map`, `par` y `BitGen` para generar finalmente el archivo `system.bit`, que permite configurar el FPGA con el sistema diseñado, pero este sistema carece de un programa de arranque.

1.3.3 *Software > Build User Applications*

Esta función de EDK se encuentra del lado de la implementación del *software* y busca compilar el programa que se desea que el procesador ejecute al energizarse el sistema. Su resultado es un archivo ejecutable de extensión ELF.

1.3.4 *Device Configuration > Update Bitstream*

Mediante esta función se hace la integración *hardware/software* y consiste en la adición del código compilado a los bloques RAM del sistema. Para efectuar esta tarea, se emplea el ejecutable `Data2Mem` que toma el ejecutable de extensión ELF, el archivo de configuración `system.bit` y un archivo BMM para generar un nuevo archivo de configuración actualizado. El archivo BMM es un archivo de texto que contiene la ubicación de los bloques de memoria cuando se ejecutó PAR durante la implementación.

1.3.5 *Device Configuration > Download Bitstream*

Esta función finalmente descarga el archivo de configuración al FPGA ejecutando un script para el *software* `iMPACT` encargado de manejar el puerto JTAG para la configuración del FPGA.

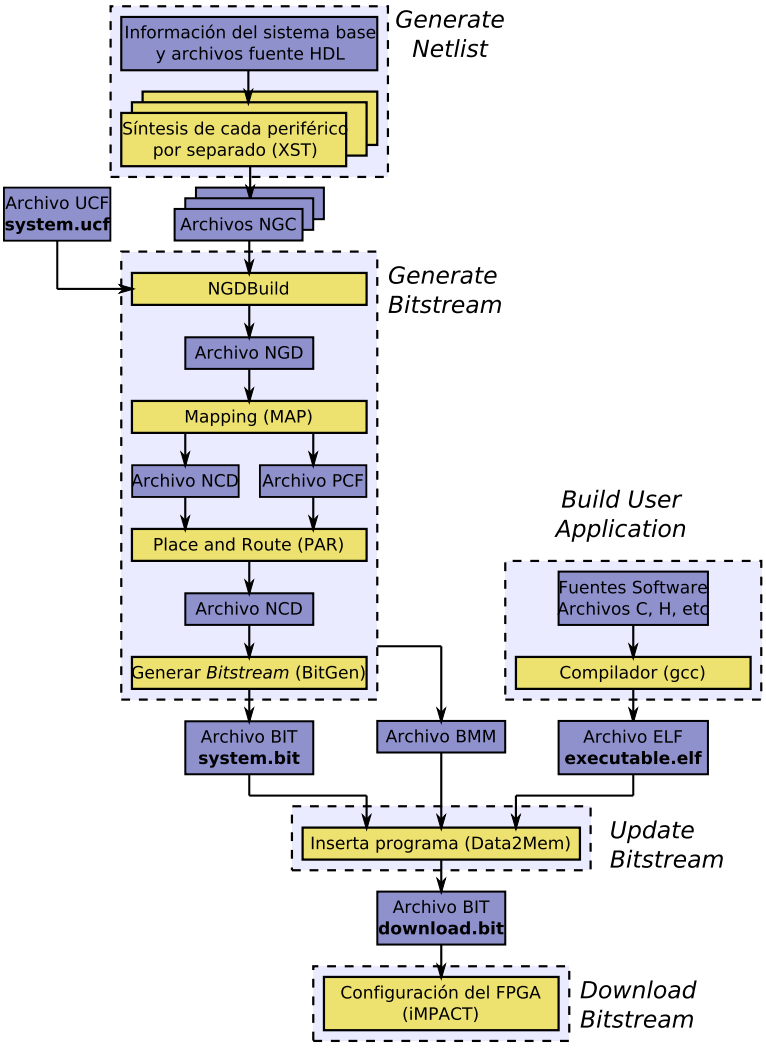


Figura 1.5: Flujo de EDK

1.3.6 PlanAhead

Es una herramienta complementaria a ISE que permite optimizar la implementación de un diseño porque permite manipular más fácilmente las restricciones tanto de ubicación de componentes como las de tiempos. Adicionalmente permite realizar varias veces la implementación bajo diferentes parámetros con el objetivo de encontrar la mejor implementación. Por otro lado, Xilinx® ha integrado su última versión del flujo

de diseño de reconfiguración parcial con esta herramienta. Dentro del proceso de implementación, se ha usado esta herramienta para restringir el sistema base a un sector específico del FPGA, cumpliendo las restricciones de tiempo propias del mismo. Aunque este proceso será explicado más ampliamente en el capítulo 4.

1.4 Implementación del sistema base

Teniendo en cuenta los requerimientos básicos del sistema y los recursos disponibles en el sistema de desarrollo ML507, se configuró el sistema base compuesto por el procesador PowerPC®y los periféricos. Esto se hizo con ayuda del asistente para la creación de un nuevo proyecto, el cual permite seleccionar entre los periféricos disponibles, cuales se quieren implementar en el sistema. La Tabla 1.2 presenta el mapa de memoria del sistema, donde se listan los componentes del sistema y el rango de memoria asignado.

Elemento	**Dirección base**	**Tamaño Rango**
Procesador PowerPC®440		
Bloque RAM	`0xFFFF0000`	64K
GPIO Leds	`0x81400000`	64K
GPIO Positions	`0x81420000`	64K
GPIO PushButtons	`0x81440000`	64K
GPIO DipSwitch	`0x81460000`	64K
IIC_EEPROM	`0x81600000`	64K
Controlador de Interrupciones	`0x81800000`	64K
MAC HW Ethernet	`0x81C00000`	64K
SysACE_CompactFlash	`0x83600000`	64K
Temporizador Watchdog	`0x83A00000`	64K
Temporizador	`0x83C00000`	64K
Puerto Serie RS-232	`0x83E00000`	64K
RAM DDR2	`0x00000000`	256MB
Memoria Flash	`0xFC000000`	32MB

Tabla 1.2: Mapa de memoria del sistema base

1.5 Sistema operativo

El sistema operativo es el *software* que, en un sistema de cómputo, administra los recursos que éste posee como los periféricos, el procesador, la memoria, el acceso al sistema de archivos, etc. El sistema operativo interactúa con las aplicaciones y los procesos que se están ejecutando y les permite acceder al *hardware* del sistema mediante colas y otros mecanismos definidos en el *kernel* del mismo. Esta es la parte más importante del sistema operativo, debido a que brinda a las aplicaciones una abstracción del *hardware*

sencilla con mecanismos de acceso.

De la misma forma en que se administran los periféricos y la memoria, el *kernel* puede administrar el tiempo de uso del procesador y así multiplexar el uso del mismo y generar un entorno multi-tareas. Para el desarrollo del proyecto se manejó el sistema operativo Linux debido a su amplia documentación y su código abierto. Este se empleó tanto en los nodos del *cluster*, como en el sistema de desarrollo ML507.

La utilización de un sistema operativo en el sistema ML507 tiene su justificación debido a los siguientes aspectos:

La administración de los recursos lógicos del sistema: Un sistema operativo permitiría la administración de los recursos lógicos del FPGA bajo un modelo de abstracción para el usuario que facilite su utilización; de forma similar a la administración de la memoria en un sistema. El diseño modular del *kernel* de Linux permitiría crear esta estructura para la implementación posterior de aplicaciones.

La necesidad de integrar la plataforma dentro de un *cluster* de procesadores Linux: Con el objetivo de que el sistema pueda ser reconocido e interactúe con los demás nodos del *cluster*, es necesario que éste ejecute servicios de red, servicios para desplegar entornos paralelos, servicios de monitorización y administración del *cluster*, etc. Esta tarea se facilita significativamente si se trabaja en un entorno multitarea Linux con la misma distribución a lo largo de todo el *cluster*.

Es posible encontrar gran variedad de distribuciones de Linux, que se diferencian en la adición o configuración de algunos paquetes que son de interés para algún grupo de usuarios. Aunque casi ninguna distribución realiza cambios sobre el núcleo, la uniformidad en la distribución de Linux a lo largo del sistema facilita la configuración del mismo y elimina uno de los factores que definen un *cluster* como heterogéneo[7]. Por ello se ha decidido utilizar la distribución Debian.

Debido a la estandarización de las plataformas de cómputo de propósito general, la instalación y el manejo de Linux se puede llevar a cabo sin necesidad de tener muchos detalles sobre el funcionamiento del sistema operativo, gracias a las herramientas que se han generado para la automatización de estos procesos. En el caso de un *embedded system*, como lo es el sistema de desarrollo ML507, es necesario conocer más a fondo la plataforma y el sistema operativo para instalarlo y configurarlo adecuadamente.

1.5.1 Arranque del Sistema operativo

Antes de realizar la instalación del sistema operativo fue necesario comprender cómo es el proceso de arranque del mismo; las Figuras 1.6 y 1.7 muestran de forma gráfica este proceso. Para que un sistema operativo Linux pueda iniciar, requiere como mínimo disponer de un sistema de archivos y del ejecutable

de un *kernel*; el archivo DTB no es obligatorio, pero ofrece información acerca del *hardware* del sistema. Estos archivos pueden estar ubicados en una memoria local dentro del sistema de desarrollo ML507, o pueden estar en un servidor de archivos remoto. En caso de que el sistema esté en etapa de desarrollo, puede ser más fácil trabajar con un servidor de archivos, debido a que este puede estar ubicado en el mismo equipo host; mientras que si el sistema ya pasó a la etapa de producción, es mejor que estos archivos se encuentren de forma local debido a que esto agilizaría el arranque del sistema operativo.

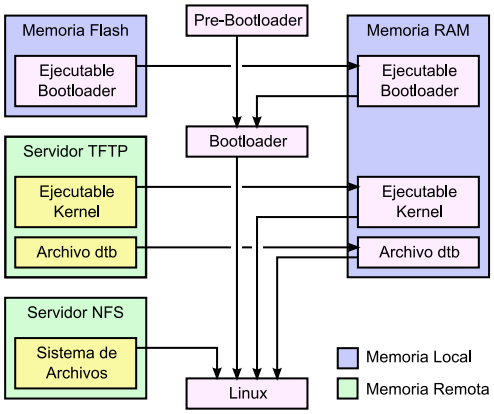


Figura 1.6: Alternativa orientada al desarrollo

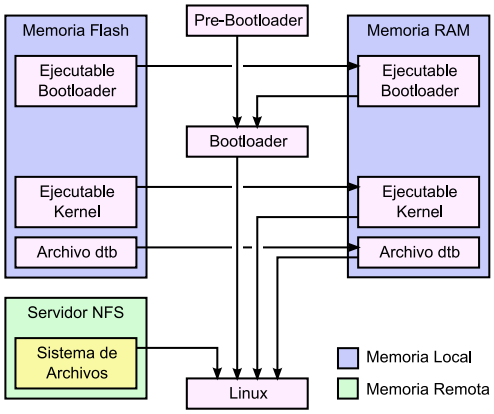


Figura 1.7: Alternativa para un funcionamiento definitivo.

En las Figuras 1.6 y 1.7 también se observa que el primer *software* que se ejecuta al iniciar el sistema no es el del sistema operativo. Normalmente el primer *software* que se ejecuta, prepara el sistema para ejecutar el bootloader, el cual es un *software* más elaborado que realiza los preparativos para cargar el sistema operativo en sí. En el caso del sistema impementado, nos referiremos al primer *software* que se ejecuta como pre-bootloader. Su función es cargar en la memoria RAM el archivo del bootloader que ha sido previamente almacenado en la memoria Flash del sistema para desde ahí ejecutarlo.

El *bootloader* es un *software* que se encarga de detectar los recursos disponibles en el sistema y ofrecer mecanismos para descargar o configurar el *kernel* y el sistema de archivos requerido para el arranque del sistema operativo. El *bootloader* puede configurar los periféricos del sistema como la interfaz de red para acceder a servidores de archivos; algunos *bootloader* implementan un intérprete de comandos que permite interactuar con el usuario para configurar los parámetros de inicio del sistema.

El *bootloader* seleccionado para el sistema fue `u-boot`. Los requerimientos del sistema en cuanto a la selección sugieren que este debe permitir acceder al *kernel* por medio de la red de datos, debido a que esto facilita la actualización del *kernel* en el *cluster* principalmente en la etapa de la implementación donde es

posible que se requiera compilar el *kernel* varias ocaciones. Esta facilidad está representada en el hecho de que todos los nodos de FPGA pueden acceder al mismo *kernel* por medio de un servidor de archivos y en que no es necesario descargarlo a una memoria local no volátil para que comience a trabajar. Otro requerimiento está relacionado con el tamaño del ejecutable, el cual debe permitir que este se almacene en la memoria Flash del sistema.

La mayoría de *bootloaders* cumplen esta condición pero la afinidad que tiene `u-boot` con las herramientas de Xilinx® y el trabajo adelantado en esta área por el grupo de trabajo de [8] permitió seleccionar esta alternativa.

1.5.2 Entorno de trabajo del sistema de desarrollo.

El desarrollo de sistemas embebidos, normalmente requiere el uso de sistemas host como lo muestra la Figura 1.8. Durante el desarrollo del proyecto se empleó el equipo que iba a tomar el papel de maestro en el *cluster* para esta tarea.

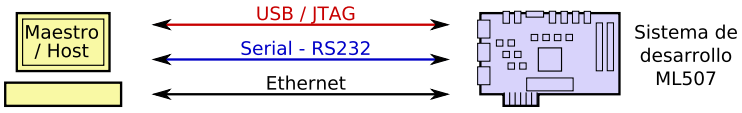


Figura 1.8: Entorno de trabajo para el desarrollo e implementación del *embedded system*

Este sistema host permite al sistema *target* realizar las siguientes funciones:

- Las herramientas cruzadas y de desarrollo para el sistema target, permiten compilar aplicaciones y generar archivos de configuración para el FPGA.
- Extender los periféricos del sistema al manejar el puerto serie RS232 como salida y entrada estándar.
- Depurar la ejecución en *software* y *hardware* mediante el protocolo JTAG y las herramientas del fabricante del FPGA.
- Intercambiar información mediante la red de datos ethernet. En esta red pueden correr gran variedad de servicios de red que amplian las posibilidades para el sistema *target*.

Basado en este entorno de desarrollo, la implementación del *pre-bootloader*, el `u-boot` y el sistema operativo se llevó acabo como se muestra en la Figura 1.9.

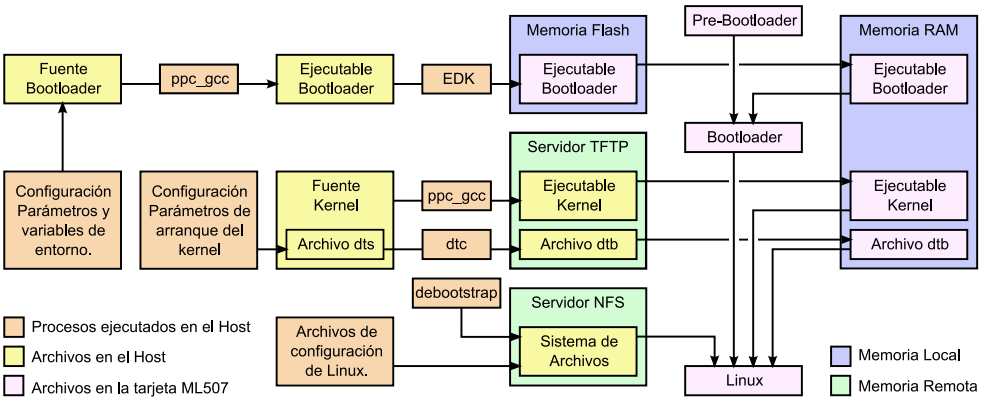


Figura 1.9: Instalación y configuración del *bootloader* y los requerimientos del sistema operativo

1.5.3 Instalación del *bootloader*

Para llevar a cabo la instalación de `u-boot` se requeire compilar el código fuente, por lo cual se requirió un compilador cruzado que permita crear el ejecutable para el procesador PowerPC® desde el computador host de arquitectura x86.

El compilador cruzado normalmente está incluido dentro de un conjunto de herramientas de desarrollo llamadas *toolchain*, el cual incluye las herramientas GNU básicas para la generación cruzada de ejecutables como: `ar`, `as`, `g++`, `gcc`, `gdb`, `ld`, `readelf`, etc. Para este proyecto se empleó el *Embedded Linux Development Kit* (ELKD), el cual es una compilación de herramientas, que incluyen el *toolchain*, que permiten implementar *Embedded Linux* y configurarlo con un sistema de paquetes basado en RPMs.

Una vez instaladas las herramientas se compiló el *bootloader*, para lo cual se descargaron las fuentes del sitio git://git.xilinx.com/u-boot-xlnx.git/. Se comprobó que el archivo `xparameters.h` del código fuente fuera el mismo que se generó con EDK en el sistema base, se dejaron las opciones del código por defecto y se compiló con ayuda del `makefile` que está en la fuente.

La instalación del ejecutable en el sistema consiste en que este se almacene en la memoria flash, para lo cual fue necesario crear el *pre-bootloader* que cargara el ejecutable a la RAM y comenzara la ejecución del *bootloader*. Para esta tarea se empleó EDK, debido a que éste permite especificar el programa de arranque y permite llevar información desde la memoria del *host* a la memoria flash del sistema. Con esta etapa finalizada, el sistema fue capaz de iniciar el *bootloader* y se pudo emplear la línea de comandos para configurar manualmente el arranque.

1.5.4 Compilación del *kernel*

Compilar el *kernel* de Linux es una tarea comunmente realizada durante el desarrollo de *embedded system*, esto permite generar un ejecutable más liviano únicamente con los módulos necesarios por el sistema. La configuración del *kernel* requiere de experiencia, conocimiento del *hardware* y de las aplicaciones que estarán ejecutándose. Para la compilación realizada para el proyecto, se accedió al sitio http://git.xilinx.com/ que contiene un *kernel* preconfigurado para los sistemas de desarrollo de Xilinx® y para los procesadores MicroBlaze y PowerPC®.

Las modificaciones más relevantes para el sistema están relacionadas con la generación del archivo DTB que contiene información sobre el *hardware* disponible en la tarjeta y a través de él se pueden establecer los parámetros de arranque del *kernel*. Estos se definieron de la siguiente forma en el archivo DTS:

```
1 bootargs =  console=ttyS0
2             ip=on
3             root=/dev/nfs
4             nfsroot=192.168.16.150:/opt/debian/ppc_4xxFP_Okinawa-21,tcp rw
5 local-mac-address = [ 00 0A 35 01 D7 4E ];
```

La generación del archivo DTB require la compilación del archivo DTS mediante el script DTC, disponible en la fuente del *kernel*. Finalmente el *kernel* se ha compiado cargando las opciones por defecto para el procesador PowerPC® 440 e indicando al momento de compilar que el *kernel* iba a ser cargado por `u-boot` para que se añadiera el cabecero requerido por este. Terminado esta etapa, se procedió a crear el sistema de archivos base para el sistema operativo.

1.5.5 Sistema de archivos

Un sistema de archivos permite organizar y almacenar información ofreciendo métodos de acceso a los datos que lo componen y administrando el dispositivo de almacenamiento. En el caso del proyecto, se ha optado por implementar un sistema de archivos remoto de tipo NFS, aprovechando el entorno de computación paralela que se despliega en el *cluster*. El *kernel* puede acceder al sistema de archivos gracias a la configuración realizada al crear el archivo DTB y este se implementó en el nodo maestro del *cluster*.

El sistema de archivos está ubicado en un subdirectorio del nodo host, y su creación se realiza de forma similar a la instalación que se hace de un sistema operativo en un ordenador personal. El hecho de que el sistema de archivos esté sobre un equipo con una arquitectura diferente al que lo va a usar, representa una dificultad, debido a que gran parte de los archivos base para el sistema operativo son ejecutables y librerías dinámicas compiladas para la arquitectura que va a correr el sistema operativo.

El sistema de archivos guarda relación con la distribución de linux que se seleccionó, para ello la herramienta emplea un repositorio de Debian vesión Lenny, la misma que se seleccionó para el *cluster*. Inicialmente la herramienta descarga los archivos necesarios para un sistema base, y posteriormente es necesario que se ejecute una segunda etapa directamente por el procesador que usará el sistema.

Finalmente dentro de la preparación que se puede hacer al sistema de archivos se pueden modificar los archivos de configuración desde el host para que estos se adapten a los parámetros que este nodo tendrá dentro de la red de datos.

1.5.6 Configuración de `u-boot`

Una vez compilado el *kernel* y el `u-boot`, configurado el sistema de archivos y generado el archivo DTB, se configuraron los parámetros de `u-boot` para que éste permita iniciar el sistema de forma automática. Para ello fue necesario iniciar `u-boot` mediante la opción de la línea de comandos donde se puedieron realizar ajustes y pruebas sobre el tipo de arranque y los parámetros del mismo. Una vez quedaron establecidos los parámetros de arranque, se compiló nuevamente el `u-boot` de modo que estos parámetros de inicio quedaran fijos. Esto se hizo modificando la variable `CONFIG_EXTRA_ENV_SETTINGS` del código fuente, lo cual permite crear un pequeño script basado en variables de entorno.

Al iniciar, `u-boot` despliega un conteo regresivo para permitir al usuario interrumpir la ejecución automática. En caso de terminar el conteo, `u-boot` ejecuta el comando almacenado en la variable de entorno `CONFIG_BOOTCOMMAND` el cual fue ajustado al valor `''run bootnfs''`, lo cual realiza las siguientes acciones:

- Carga el *kernel* y el archivo DTB por TFTP.
- Borra la variable del sistema de archivos en RAM, debido a que se va a usar el sistema de archivos en red (NFS).
- Iniciar el sistema operativo.

Las modificaciones realizadas se resumen en el siguiente cuadro:

Modificaciones a las variables de entorno de u-boot

```
1
2 #define CONFIG_BOOTCOMMAND "run bootnfs"
3
4 #define CONFIG_EXTRA_ENV_SETTINGS \
5   "serverip=192.168.16.150\0" \
6   "ipaddr=192.168.16.171\0" \
7   "bootnfs=run loadkernel;run loaddtb;setenv ramimg_addr -;run runkernel\0" \
8   "boottftp=run loadkernel;run loadramimg;run loaddtb;run runkernel\0" \
```

```
9       "loadkernel=tftp $(kernel_addr) $(bootfile)\0" \
10        "kernel_addr=0x1c00000\0" \
11        "bootfile=Linuxboot/uImage\0" \
12      "loadramimg=tftp $(ramimg_addr) $(ramfile)\0" \
13        "ramimg_addr=0x1800000\0" \
14        "ramfile=Linuxboot/ramimg.gz\0" \
15      "loaddtb=tftp $(dtb_addr) $(dtbfile)\0" \
16        "dtb_addr=0x1000000\0" \
17        "dtbfile=Linuxboot/ml507-Okinawa-21.dtb\0" \
18      "runkernel=bootm $(kernel_addr) $(ramimg_addr) $(dtb_addr)\0" $
19
```

1.6 Resumen del capítulo

Con el avance mostrado en el presente capítulo, se ha configurado el sistema de desarrollo ML507 como un *embedded system* basado en el procesador PowerPC®que corre Linux Lenny con ayuda de varios servicios de red y de un sistema de archivos NFS implementados sobre el nodo maestro del *cluster*. Esto ha permitido que tanto el sistema de desarrollo ML507, como el *cluster* obtengan beneficios mutuos. El sistema de desarrollo ha suplido requisitos para correr el sistema operativo, mientras el *cluster* ha ganado un trabajador con recursos lógicos para procesamiento embebido.

Capítulo 2

Configuración del clúster

A lo largo del presente capítulo se presentan los aspectos más relevantes de la configuración del *cluster* heterogéneo conformado por el nodo maestro `Okinawa-00`, nodos trabajadores `Okinawa-01` y `Okinawa-02` de arquitectura x86 implementados como máquinas virtuales y el sistema ML507 configurado como se mostró en el capítulo 1. La Figura 2.1 muestra los elementos involucrados durante este proceso, mientras la Figura 2.2 presenta las etapas expuestas en este capítulo dentro del plan para el desarrollo del proyecto.

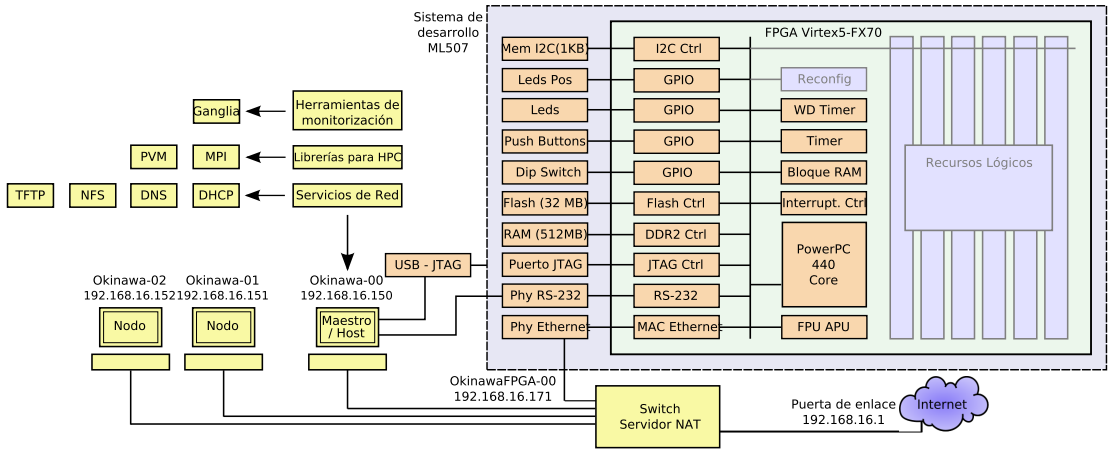


Figura 2.1: Diagrama general. Recursos empleados durante este capítulo.

La estrategia adoptada para la configuración de las herramientas que permiten convertir el sistema en un *cluster*, inicialmente se centró en los nodos de arquitectura x86 y por último en los nodos FPGA. Esto debido a que la instalación y pruebas son más fáciles de llevar a cabo sobre sistemas homogéneos.

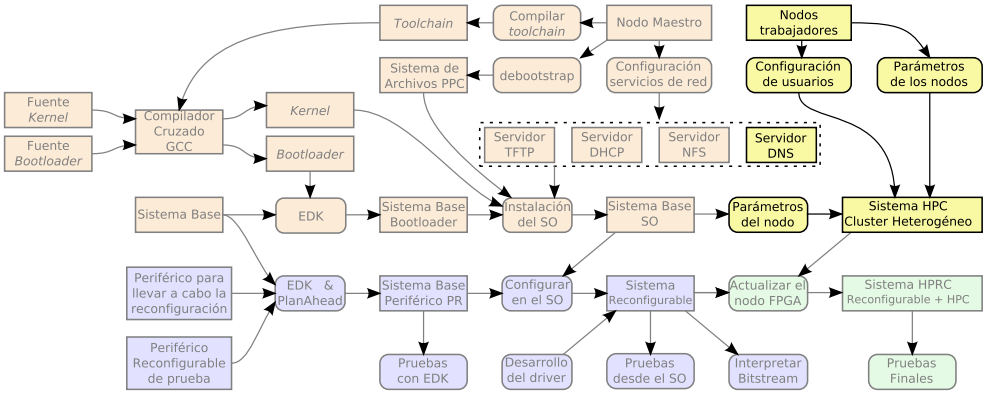


Figura 2.2: Pasos de la implementación del sistema.

2.1 Organización del *cluster*

Antes de iniciar las labores de configuración de los sistemas de cómputo, se definieron algunos parámetros del sistema relacionados con el sistema de archivos compartido y las direcciones IP de cada uno de los nodos. La Figura 2.3 muestra la disposición física de los nodos y la asignación de los parámetros de configuración de red.

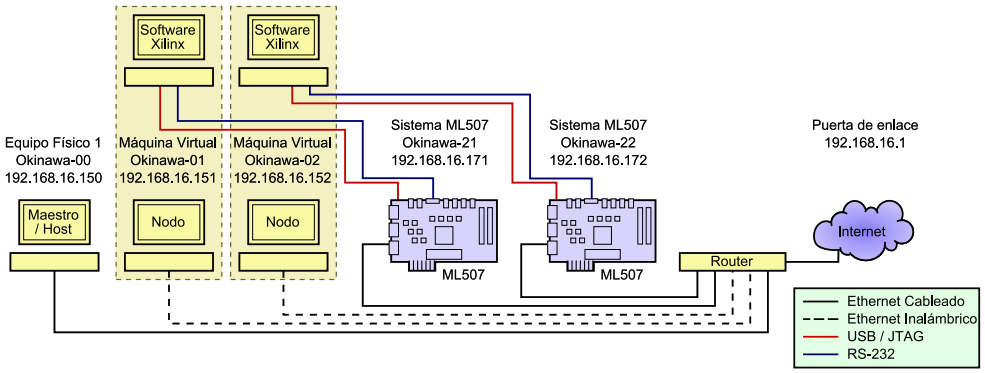


Figura 2.3: Diagrama de la implementación física del sistema.

La organización del sistema de archivos se realizó con el objetivo de cumplir con los requerimientos que tiene el *cluster* y con los que tiene el sistema de desarrollo ML507 para ejecutar el sistema operativo. La Tabla 2.1 muestra la organización final de los directorios compartidos. Allí se observa que existen directorios que son compartidos entre todos los nodos del *cluster* y otros directorios que son compartidos para los nodos de una arquitectura en particular. De esta forma, la información que es común, como por ejemplo los directorios `home` de los usuarios del *cluster*, pueden ser accedidos desde cualquier nodo; pero los archivos que son compilados para una arquitectura en particular, como ejecutables o librerías son

accedidos en directorios compartidos para esa arquitectura.

La Tabla 2.1 también muestra algunos directorios compatidos que corresponden a los sistemas de archivos de los nodos FPGA, tanto en su versión final, como en las etapas de desarrollo.

Directorio compartido en el **Nodo Maestro**	Punto de montaje en **Nodos x86**	Punto de montaje en **Nodos PowerPC®**
/compartido/comun	/compartido/comun	/compartido/comun
/compartido/x86	/compartido/x86	–
/compartido/powerpc	–	/compartido/powerpc
/opt/ELDK/4.2/ppc_4xxFP	–	/ (Solo durante la configuración inicial)
/opt/debian/ppc_4xxFP	–	/ (Sistema de archivos base)
/opt/debian/ppc_4xxFP_Okinawa-21	–	/ (Copia del sistema de archivos para Okinawa-21)

Tabla 2.1: Configuración del sistema de archivos compartido.

2.2 Herramientas del *cluster*.

Aunque cada herramienta fue probada inicialmente en el *cluster* homogéneo que forman los nodos de arquitectura x86, la selección de cada una de ellas y la asignación de parámetros durante su instalación se realizó teniendo en cuenta que la implementación final es de uno heterogéneo.

Las herramientas seleccionadas para el desarrollo de aplicaciones paraleas fueron PVM y LAM-MPI, las cuales se caracterizan por dar soporte a *cluster* con cierto grado de heterogeneidad. Adicionalmente se instaló ganglia como herramienta de monitorización.

2.2.1 Ajustes preliminares

Las herramientas que se instalaron, sugieren que el *cluster* disponga de algunos servicios de red y de algunas configuraciones particulares que son requeridas o que facilitan su funcionamiento. Dentro del *cluster* se realizaron específicamente los siguientes ajustes:

Instalación de ssh: Esta utilidad es requerida por todas las heramientas porque da soporte a la seguridad de las conexiones entre los nodos del sistema.

Usuarios del *cluster*: Los usuarios del *cluster*, fueron creados en cada uno de los nodos del mismo. Estos fueron configurados para que cuando se inicie sesión remota desde otro nodo, no se requiriera clave.

Ubicación de los directorios de usuario: El directorio `home` de cada usuario se ubicó en un directorio compartido mediante NFS. De esta forma, con el mismo usuario, se accede a la misma información desde cualquier nodo.

Servidor DNS: Es más fácil siempre hacer referencia a los nodos del sistema mediante nombres que mediante sus IPs. Por ello se instaló este servicio de red.

2.2.2 Instalación de PVM

La instalación de PVM requiere la compilación de las librerías y los servicios en cada una de las arquitecturas. Para ello se siguieron los procedimientos del sitio de soporte de la herramienta, donde también se descargaron las fuentes `http://www.netlib.org/pvm3/`. Inicialmente se realizó este procedimiento con los nodos de arquitectura x86 y posteriormente con el nodo FPGA. En cada uno de los casos se ejecutó un programa de prueba para comprobar la funcionalidad de la librería, y los procedimientos para compilar y ejecutar el ejemplo en ambas arquitecturas. Los detalles de la instalación y las pruebas realizadas se encuentran en el Anexo A. Al ejecutar el programa de ejemplo, se solicitó que cada uno de los nodos del sistema imprimiera la fecha y la hora locales. La salida fue como se muestra a continuación:

```
 william@Okinawa-00# ./helloPVM_master

The master process runs on Okinawa-00
I found the following hosts in your virtual machine
        Okinawa-00
        Okinawa-01
        Okinawa-02
        Okinawa-21
        Okinawa-22
Comienzo intento con el servidor numero 0
Comienzo intento con el servidor numero 1
Comienzo intento con el servidor numero 2
Comienzo intento con el servidor numero 3
Comienzo intento con el servidor numero 4
Okinawa-00's time is Thu Jan 19 16:15:45 2010

Okinawa-01's time is Thu Jan 19 09:03:18 2010

Okinawa-02's time is Thu Jan 19 09:01:34 2010

Okinawa-21's time is Thu Jan 19 16:15:45 2010

Okinawa-22's time is Thu Jan 19 16:15:46 2010
```

2.2.3 Instalación de LAM-MPI

La instalación de LAM-MPI requirió, al igual que PVM, que cada una de las arquitecturas compile las librerías y los ejecutables de la herramienta. Adicionalmente fue necesario modificar los archivos de configuración de la herramienta que describen la organización del *cluster*. Las instrucciones seguidas en este proceso se encuentran en `http://www.lam-mpi.org` y los detalles del mismo se encuentran en el Anexo A.

Para comprobar el funcionamiento de esta herramienta se ejecutó un programa de prueba, que identifica a cada uno de los nodos del sistema. Su salida se muestra a continuación:

```
 william@Okinawa-00#  mpiexec -configfile my_appfile
Hello, World.  I am 0 of 2
Hello, World.  I am 1 of 5
Hello, World.  I am 3 of 5
Hello, World.  I am 2 of 5
Hello, World.  I am 4 of 5
```

2.2.4 Instalación de Ganglia

Como herramienta de monitorización se seleccionó Ganglia, debido a que este ha sido usado ampliamente en diferentes implementaciones de *clusters*. Esta herramienta ofrece la posibilidad de consultar via web el histórico de características de los nodos del *cluster* como el porcentaje de uso de la cpu y la memoria. Su puesta en marcha requirió la instalación de un servidor web, php y algunas herramientas web en el nodo maestro. La apariencia de ganglia se muestra en la Figura 2.4

2.3 Resultados del capítulo

Mediante el procedimiento descrito en el presente capítulo se logró implementar una plataforma de cómputo paralelo compuesta por nodos con procesadores x86 de propósito general y FPGAs con procesadores PowerPC®. Se han instalado y configurado las dos herramientas más populares para cómputo en paralelo en sistemas heterogéneos: LAM-MPI y PVM. Su funcionamiento se comprobó mediante programas de ejemplo y finalmente se instaló la herramienta de monitorización ganglia.

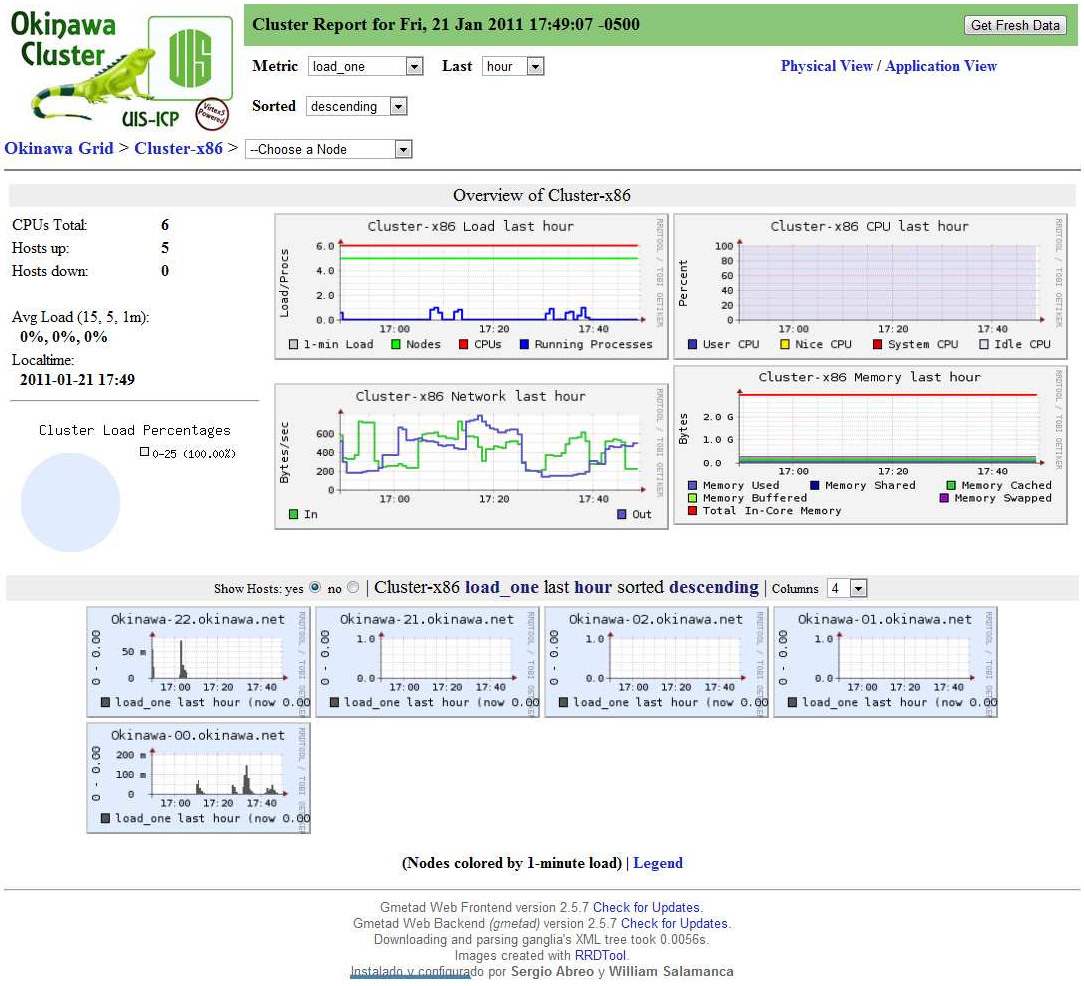


Figura 2.4: Visualización de los nodos del sistema en ganglia.

Capítulo 3

Interpretación de los *bitstreams*

El propósito del presente capítulo es inicialmente presentar los conceptos básicos sobre la configuración del FPGA y la arquitectura del mismo. Debido a que la documentación de los archivos de configuración y la arquitectura del FPGA no está completa, fue necesario hacer un proceso de ingeniería inversa para interpretar estos archivos. Para ello fue necesario emplear algunas herramientas existentes y generar algunas. Al final del documento se presentan los primeros resultados obtenidos más relevantes.

La Figura 3.1 muestra la etapa que se está adelantando dentro de la metodología completa del proyecto. Es posible adelantar parte de la ejecución de esta taréa debido a que gran parte de la documentación se obtuvo del fabricante.

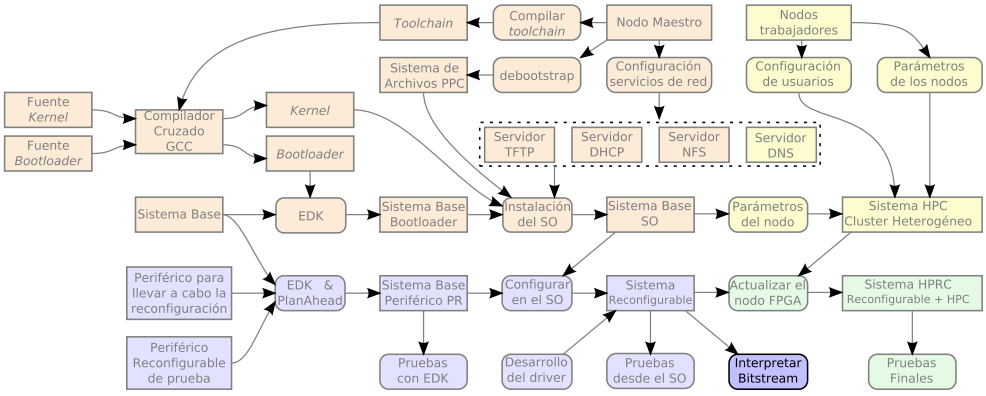


Figura 3.1: Pasos de la implementación del sistema.

3.1 Configuración del FPGA

Un FPGA es un dispositivo electrónico que consta de recursos lógicos y recursos de interconexión que le permiten implementar cualquier circuito digital, siempre que se disponga de la cantidad de elementos

lógicos necesarios. El proceso de configurar el FPGA consiste en descargar a este, la información sobre la interconexión y configuración de sus recursos lógicos para hacer que éste se comporte como circuito digital en particular. Esta información es almacenada en el FPGA de forma volátil, debido a esto es necesario configurar el FPGA cada vez que se enciende. Así mismo, este proceso puede repetirse una vez el dispositivo esté configurado para modificar completamente o parcialmente el circuito implementado.

3.1.1 Métodos de configuración

Al energizar el FPGA, este no tiene ningún circuito implementado. Su archivo de configuración inicial debe ser descargado y para ello se emplean los pines dedicados para ello, los cuales pueden trabajar bajo diferentes modos según el estado de los pines M[2:0]. Estos modos de trabajo permiten obtener la información de configuración desde diferentes dispositivos de almacenamiento como memorias (seriales, paralelas), procesadores u otro dispositivo digital. La Tabla 3.1 presenta los modos de configuración válidos en los FPGA Virtex 5.

$M[2:0]$	**Modo**	**Descripción**
0 0 0	Master Serial	Interfaz serie síncrona donde el FPGA es el maestro. Permite manejar memorias PROM fabricadas por el fabricante Xilinx®
0 0 1	Master SPI	Interfaz serie síncrona donde el FPGA es el maestro. Permite manejar memorias genéricas de cualquier fabricante que manejen el protocolo SPI
0 1 0	Master BPI-Up	Interfaz Paralela de 8 o 16 bits donde el FPGA es maestro.
0 1 1	Master BPI-Down	Interfaz Paralela de 8 o 16 bits donde el FPGA es maestro.
1 0 0	Master SelectMAP	Interfaz Paralela de 8 o 16 bits donde el FPGA es maestro.
1 0 1	JTAG	Interfaz serie síncrona por medio del conocido protocolo JTAG
1 1 0	Slave SelectMAP	Interfaz Paralela de 8 o 16 bits donde el FPGA es esclavo.
1 1 1	Slave Serial	Interfaz serie síncrona donde el FPGA es esclavo.

Tabla 3.1: Modos de configuración del FPGA

La interfaz JTAG emplea pines dedicados para esta interfaz y puede ser empleada en cualquier momento, incluso sin necesidad de configurar los pines $M[2:0]$ en unos niveles particulares. Al momento de activarse la interfaz JTAG, cualquier otro modo de configuración queda deshabilitado. Los otros modos

de configuración emplean pines que pueden ser utilizados como pines de propósito general una vez esté configurado el FPGA.

Durante el desarrollo del proyecto, se empleó el modo JTAG en los momentos que se estuvo depurando el sistema base. Mientras que cuando el *hardware* se debía mantener fijo, se descargó el archivo de configuración en las memorias PROM del sistema de desarrollo, las cuales pueden configurar el FPGA por medio de SelectMAP maestro.

3.1.2 Arquitectura del FPGA V5FX70

Los FPGA son dispositivos con gran cantidad de recursos lógicos que permiten implementar sistemas digitales. Estos recursos van desde los elementales *Look up tables* (LUT) y Flip Flops, hasta bloques de procesamiento DSP, bloques de comunicaciones y procesadores de propósito general enteros. Cada familia de FPGA se caracteriza por tener determinados recursos que la especializan en cierto tipo de tareas específica (procesamiento de señales, transmisión de datos, procesamiento embebido, etc). Todos estos recursos comparten el área del chip y tienen una organización ya establecida. En el FPGA seleccionado para el proyecto, la organización se presenta en la Figura 3.2. El bloque más grande que se aprecia es el procesador PowerPC®, el cual según las proporciones de la imagen ocupa aproximadamente el 7% del área del chip.

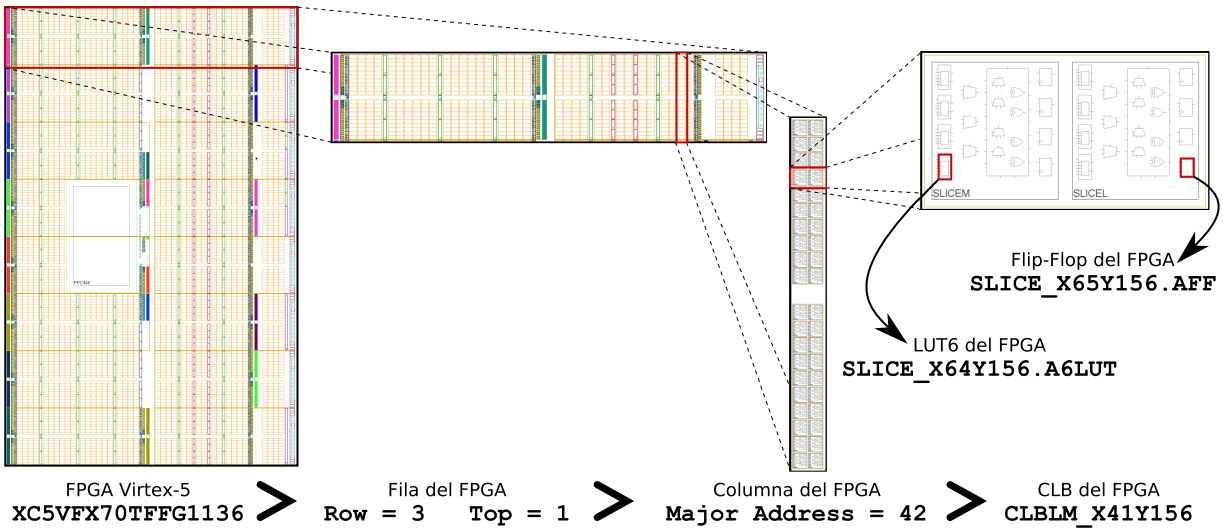


Figura 3.2: Arquitectura del FPGA Virtex 5 FX70, donde se aprecia la organización en filas, columnas y elementos lógicos. (Imágenes tomadas de PlanAhead.)

El FPGA `XC5VFX70TFFG1136` está segmentado en 8 filas, las cuales a su vez están divididas en

columnas donde cada columna tiene solo un tipo de recurso lógico (CLBs, RAMBs, IOB, etc). Cada columna tiene organizados verticalmente una cantidad de bloques lógicos que varía según el tipo de recurso. Durante el desarrollo de esta tesis, se tomó como objetivo llegar a la configuración de los LUTs que están contenidos dentro de los Slices que a su vez están contenidos en los CLBs del FPGA.

3.1.3 Bitstreams

Los *bitstreams* son archivos, normalmente binarios, que contienen la información necesaria para configurar el FPGA. Los *bitstreams* pueden ser descargados al FPGA de varias formas, y están compuestos basicamente por escrituras y lecturas de los registros de configuración. Finalmente lo que buscan es manipular la memoria de configuración del FPGA. Con el objetivo de realizar la configuración parcial, fue necesario conocer la estructura del *bitstream* y la memoria de configuración, donde los términos frame y paquetes tipo I y II adquieren vital importancia.

3.1.3.1 Frames

Los frames corresponden a las secciones del FPGA más pequeñas que se pueden reconfigurar parcialmente. Están compuestos por 41 palabras de 32 bits. El archivo de configuración del FPGA V5FX70 está compuesto por 21.080 frames, los cuales se pueden direccionar mediante una palabra de 32 bits, segmentada en los siguientes campos:

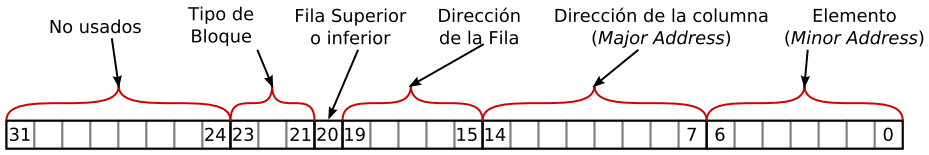


Figura 3.3: Campos de la dirección de un Frame (Adaptada de [3])

Tipo de Bloque: Este campo hace referencia a 4 posibles valores que diferencian entre 4 tipos de información de configuración:

- Configuración o interconexión de algún recurso lógico.
- Contenido de un bloque RAM.
- Configuración o interconexión de recursos especiales.
- frames de bloques RAM que no son de configuración.

Fila Superior o Inferior: Permite saber si la fila a la que se hace referencia está en la mitad superior o inferior del FPGA.

Dirección de la Fila: Hace referencia a una de las filas del FPGA, las cuales están numeradas comenzando desde el centro con el número cero.

Dirección de la columna: En el interior de la fila, hace referencia a una de las columnas, las cuales están numeradas de izquierda a derecha comenzando en cero.

Minor Address: Mediante este dato se direcciona cada uno de los frames que conforman una columna. El máximo valor de este campo varía según el tipo de recurso lógico que compone la columna según lo muestra la Tabla 3.2.

Tipo de recurso	**Número de frames por columna**
CLB	36
DSP	28
Bloque RAM	30
Bloque IO	54
Clock column	4

Tabla 3.2: Número de frames que componen cada una de las columnas según el tipo de recurso

3.1.3.2 Estructura del *bitstream*

Un *bitstream* es una serie de comandos y datos que son aplicados sobre los registros y la memoria de configuración. Más específicamente, el *bitstream* se compone de paquetes tipo I, que corresponden a escrituras y lecturas de los registros de configuración y paquetes tipo II que se usan en combinación de paquetes tipo I para escribir grandes volúmenes de información.

Inicialmente el *bitstream* contiene información del archivo como la hora y fecha en la que fue generado. Posteriormente configura los registros e inicia la comprobación del CRC y finalmente escribe el contenido del archivo de configuración.

3.2 Intérprete de *bitstream* en Matlab

De forma paralela con el desarrollo de la tesis se fue elaborando un script en Matlab que permitió recopilar la información obtenida de los documentos técnicos de Xilinx® sobre la configuración del FPGA, y de las pruebas de ingeniería inversa. El objetivo del script es interpretar un *bitstream* identificando sus partes y etiquetando cada uno de los comandos y los datos en un archivo ASCIIcon comentarios al final de cada línea. El código fuente de este script se encuentra en el Anexo C.

El script está en capacidad de interpretar *bitstream* en formato binario y en formato ASCII modificando la variable `tipodearchivo` (1 en caso de ser ASCII, 2: en caso de ser binario).

3.2.1 Organización del script

Las fuentes del script se organizaron como se muestra en la Figura 3.4

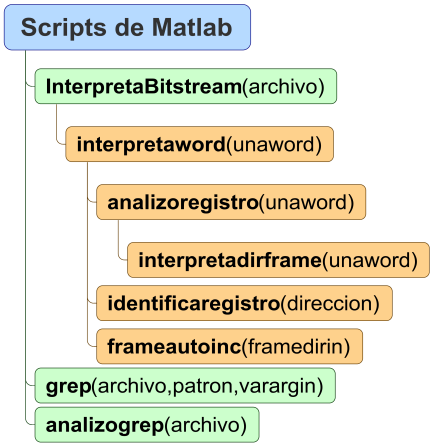


Figura 3.4: Scripts de matlab para interpretar los *bitstreams*.

InterpretaBitstream(archivo): Este es el script principal y se encarga de abrir los archivos de entrada y de salida, identificar el tipo de archivo de entrada, leer el cabecero, entregar cada palabra del contenido del archivo a la rutina interpretaword y finalmente imprimir en el archivo de salida y cerrar los archivos.

interpretaword(unaword): Esta rutina se encarga de interpretar cada una de las palabras del archivo de configuración. Estas palabras pueden representar comandos, cabeceros de paquetes tipo I y cabeceros de paquetes tipo II.

analizoregistro(unaword): Cuando `interpretaword` ha leido un paquete tipo I y busca saber que efecto tendrá la modificación del registro en particular, entonces llama a esta rutina que devuelve una cadena de caracteres con un comentario sobre esta acción.

interpretadirframe(unaword): Esta rutina segmenta la dirección de un frame en los campos según lo indica la Figura 3.3.

identificaregistro(direccion): Rutina que devuelve el nombre de un registro en específico.

frameautoinc(framedirin): Esta rutina contiene la información acerca del orden en el que Xilinx® empaqueta los frames en un *bitstream*. De esta forma se puede saber cual es la dirección del siguiente frame a partir de la dirección del actual.

grep(archivo,patron,varargin): Permite tomar un archivo interpretado por InterpretaBitstream e imprimir únicamente las líneas que contengan la cadena definida por el patrón. Fue empleado para

dejar únicamente las líneas impresas por interpretadirframe donde se evidencia el orden de los frames en un *bitstream* generado con la opción de debug.

analizogrep(archivo): Toma como entrada el archivo generado por grep y genera otro archivo con un resumen donde indica la cantidad de frames que se encuentran de forma secuencial con todos sus campos idénticos pero con diferente minor address.

3.2.2 Resultados obtenidos con Matlab

Al llevar a cabo el análisis del *bitstream* con matlab se logró validar la coherencia entre la documentación de Xilinx® y los *bitstream* generados por `bitgen`. Además se logró inferir el orden en el que están dispuestos los diferentes tipos de recursos lógicos en forma de columnas a lo largo de cada una de las filas del dispositivo `XC5VFX70TFFG1136`. Los resultados de esta inferencia fueron comparados satisfactoriamente con la representación gráfica que presentan PlanAhead y FPGA Editor. El resumen de esta organización se presenta en la tabla 3.3.

Tabla 3.3: Organización de los recursos del FPGA por columnas

Major Address	**Tipo de Recurso**	**Número de Frames**
0	IOB	54
1	CLB	36
2	CLB	36
3	CLB	36
4	CLB	36
5	RAMB	30
6	CLB	36
7	CLB	36
8	CLB	36
9	CLB	36
10	CLB	36
11	CLB	36
12	RAMB	30
13	CLB	36
14	CLB	36
15	CLB	36
16	CLB	36
17	CLB	36
18	CLB	36
19	RAMB	30
20	CLB	36
21	CLB	36
22	CLB	36
23	CLB	36
24	IOB	54
25	ClkCol	4

Major Address	**Tipo de Recurso**	**Número de Frames**
26	CLB	36
27	CLB	36
28	CLB	36
29	CLB	36
30	RAMB	30
31	CLB	36
32	CLB	36
33	DSP	28
34	CLB	36
35	CLB	36
36	DSP	28
37	CLB	36
38	CLB	36
39	RAMB	30
40	CLB	36
41	CLB	36
42	CLB	36
43	CLB	36
44	IOB	54
45	CLB	36
46	CLB	36
47	CLB	36
48	CLB	36
49	RAMB	30
50	¿? NoID	34

Esta organización contiene implícita la información sobre el orden en el direccionamiento de los frames, el cual se convierte en otro resultado importante de este proceso.

Capítulo 4

Periférico para la reconfiguración parcial

En este capítulo se desarrolla el proceso mediante el cual se puso en funcionamiento el periférico que permitió manipular la configuración del FPGA y además el proceso que permitió identificar los bits que se deben modificar en la memoria de configuración para manipular el comportamiento de cada look-up table (LUT) del FPGA. La Figura 4.1 destaca las etapas que se documentan en este capítulo y la Figura 4.2 resalta los elementos del sistema involucrados.

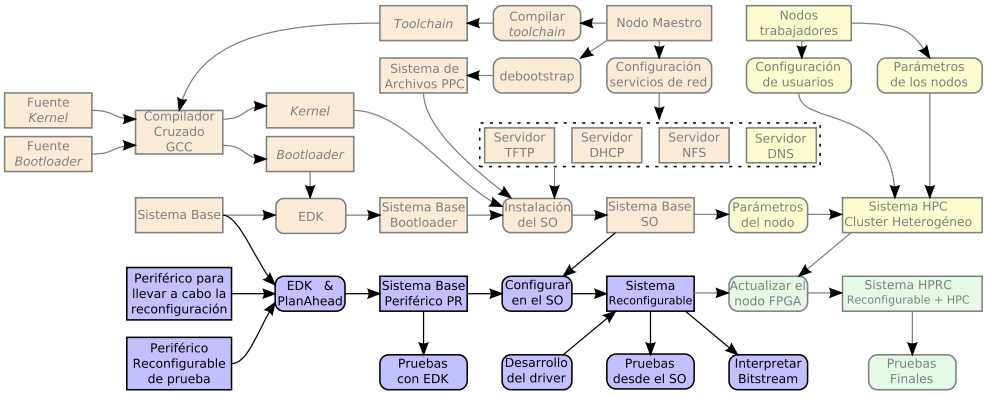


Figura 4.1: Pasos de la implementación del sistema.

Para llevar a cabo estas tareas se requirió la implementación de dos periféricos en el sistema: uno para modificar la configuración del FPGA y otro reconfigurable de prueba para validar el funcionamiento del primero.

4.1 Modificaciones al sistema base

El sistema base al cual se le instaló el sistema operativo tuvo que ser modificado para añadir los periféricos necesarios para la reconfiguración y para distribuir mejor los recursos lógicos del sistema base y los recursos

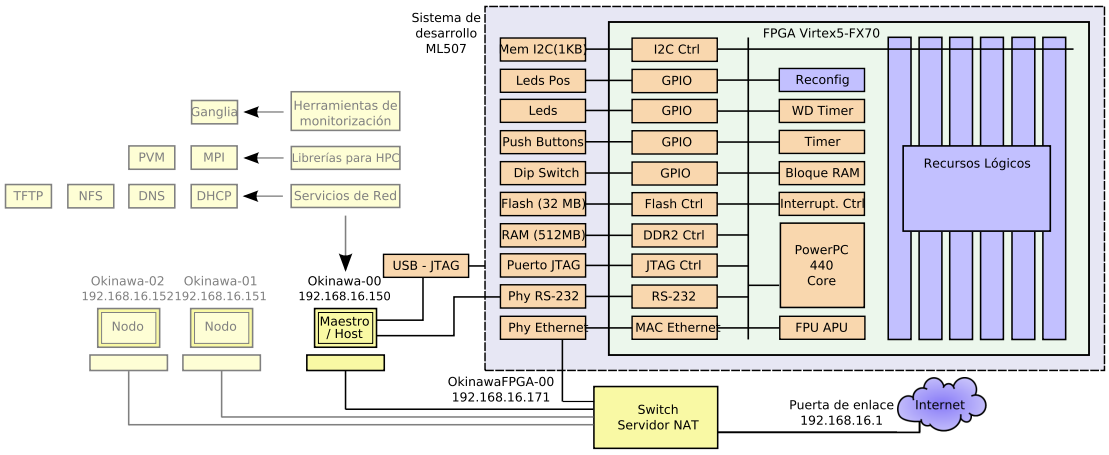


Figura 4.2: Diagrama general. Recursos empleados durante este capítulo.

lógicos disponibles para ser reconfigurados.

4.1.1 Periférico reconfigurable de prueba

Con el objetivo de poder llevar a la práctica los resultados del análisis de los *bitstreams* y la estructura del FPGA, se creó un periférico de prueba en el sistema base que lleva a cabo la operación lógica AND bit a bit entre 6 palabras de 32 bits. La interfaz con el procesador se realiza mediante 6 registros como muestra la Figura 4.3 y la operación lógica es realizada por medio de 32 LUTs.

La instanciación de las LUTs se realizó mediante las primitivas de Xilinx® después de llevar a cabo el asistente para la creación de un nuevo periférico. Todo esto dentro del archivo `user_logic.vhd` que genera EDK al final del asistente. El Cuadro de Código 4.1 muestra la descripción de las LUTs.

```
LasLUTs:
  for xx in 0 to 31 generate
    begin
    Inst : LUT6
    generic map (
      INIT => I0 and I1 and I2 and I3 and I4 and I5) -- Specify LUT Contents
    port map (
      O  => S(xx),           -- LUT general output
      I0 => slv_reg0(xx),    -- LUT input
      I1 => slv_reg1(xx),    -- LUT input
      I2 => slv_reg2(xx),    -- LUT input
      I3 => slv_reg3(xx),    -- LUT input
      I4 => slv_reg4(xx),    -- LUT input
      I5 => slv_reg5(xx)     -- LUT input
    );
 end generate;
```

Cuadro de Código 4.1: Implementación de las LUTs en el periférico de pruebas

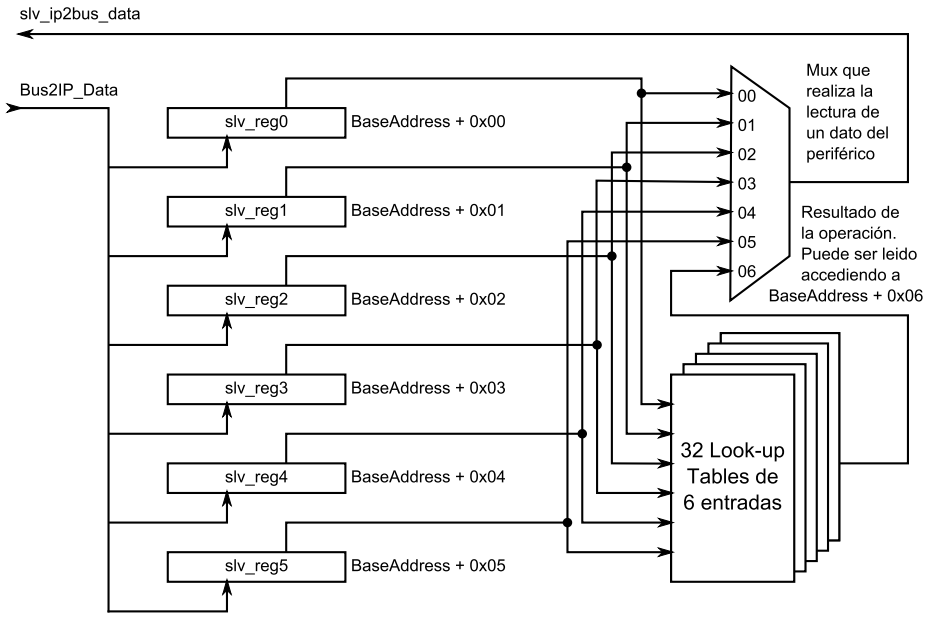


Figura 4.3: Estructura del periférico reconfigurable de pruebas.

El objetivo de implementar este periférico es realizar la reconfiguración parcial sobre las LUTs que lo componen y modificar la función lógica que lleva a cabo.

4.1.2 Periférico para acceder a la configuración del FPGA

Dentro del sistema implementado era necesario ofrecer al sistema base la posibilidad de acceder a su configuración interna. Para llevar a cabo esta función se debe emplear un recurso dentro del FPGA llamado *Internal Configuration Access Port* (ICAP). Inicialmente se estudió la posibilidad de crear el periférico que contuviera el ICAP, pero dado que Xilinx® ya ofrece un periférico que maneja este recurso mediante una interfaz para el bus PLB, se optó por emplearlo en el sistema.

4.1.2.1 XPS_HWICAP

Este es el nombre del periférico creado por Xilinx® que permite acceder a la configuración del FPGA. Su estructura interna se basa en registros que modifican el comportamiento de una máquina de estados que finalmente maneja las señales del ICAP. Estos registros son accedidos por el procesador por medio de la interfaz PLB, la cual está configurada para soportar el modo *burst* y aprovecha la memoria FIFO que tiene implementado el HWICAP. Entre los registros más relevantes dentro del funcionamiento del módulo HWICAP se encuentran los siguientes:

Write FIFO Register (WF) y Read FIFO Register (RF): Permiten escribir y leer datos sobre las memorias FIFO implementadas sobre el HWICAP para lectura y escritura.

Write FIFO Vacancy Register (WFV) y Read FIFO Occupancy Register (RFO): Estos registros permiten saber el nivel de datos que tienen las memorias FIFO.

Control Register (CR) y Status Register (SR): Contienen información del estado actual del periférico y permiten manipular cada uno de los procesos que puede realizar el periférico.

La adición del periférico al sistema se realizó de la forma estándar que tiene EDK para ello, donde se le asigna un rango dentro del mapa de memoria junto a los otros periféricos del sistema (práctica 2 de [9]).

4.1.3 Optimizaciones en la implementación mediante PlanAhead

Para poder realizar pruebas de la reconfiguración sobre el periférico de prueba, fue necesario restringir el diseño del sistema base sobre un área específica. Para ello se empleó la herramienta PlanAhead 12.2, que facilita la manipulación de los parámetros de la implementación del circuito mediante una interfaz gráfica que permite la edición del archivo UCF de restricciones del diseño.

Por ello fue necesario intervenir el flujo de diseño de EDK para añadir las restricciones del sistema. La Figura 4.4 muestra la forma en la que se llevó a cabo este proceso. Básicamente durante este procedimiento, se deja que EDK realice las etapas para generar el *bitstream* `system.bit`, pero de forma paralela se importan los archivos NGC a PlanAhead para generar otro archivo `system.bit`, el cual se ha generado bajo unas nuevas condiciones de localización de las primitivas del circuito. El nuevo archivo de configuración sustituye al que generó EDK, sin que esto afecte el flujo de diseño y de esa forma pueda finalmente configurarse el FPGA.

4.1.3.1 Objetivos de las restricciones de localización

La localización de los elementos del sistema base busca hacer una distribución de recursos del FPGA entre la sección fija del sistema (no reconfigurable), el periférico de prueba reconfigurable y los posibles periféricos adicionales que se puedan implementar. Para ello se tuvieron en cuenta los siguientes dos aspectos:

- Se debe conocer la ubicación de las primitivas del periférico reconfigurable de prueba para que ésta pueda ser llevada a cabo.
- Con el objetivo de mejorar el uso de los recursos lógicos y de favorecer la futura implementación de periféricos en el sistema, se destinará la mayor cantidad posible de recursos para periféricos de cómputo que implementen en *hardware* funciones específicas de algoritmos.

Este último aspecto busca particularmente que queden la mayor parte de los recursos especiales del FPGA disponibles para dichos periféricos. Especialmente los bloques RAM y bloques DSP.

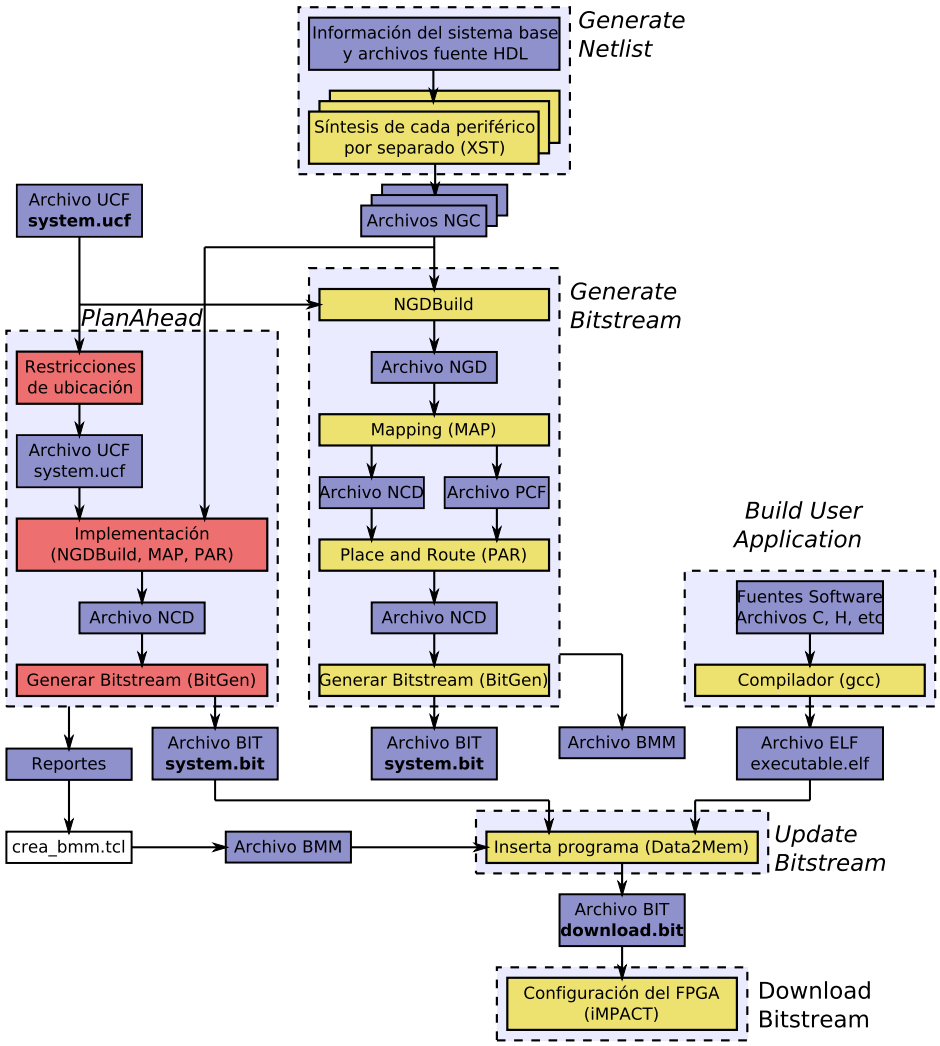


Figura 4.4: Flujo de EDK modificado para emplear PlanAhead

Desde el punto de vista del sistema base, es necesario que el área que se sea asignada, contenga los recursos necesarios que exige el circuito y que su distribución permita que se puedan cumplir las restricciones de tiempo. Adicionalmente el controlador de la memoria DDR2 y el periférico de ethernet tienen exigencias particulares respecto a algunos recursos que reciben las señales directamente desde los pines, por lo cual es necesario que el área asignada al sistema base contenga dichos recursos. La Figura 4.5 muestra el panorama de la localización de los recursos requeridos por parte del sistema base.

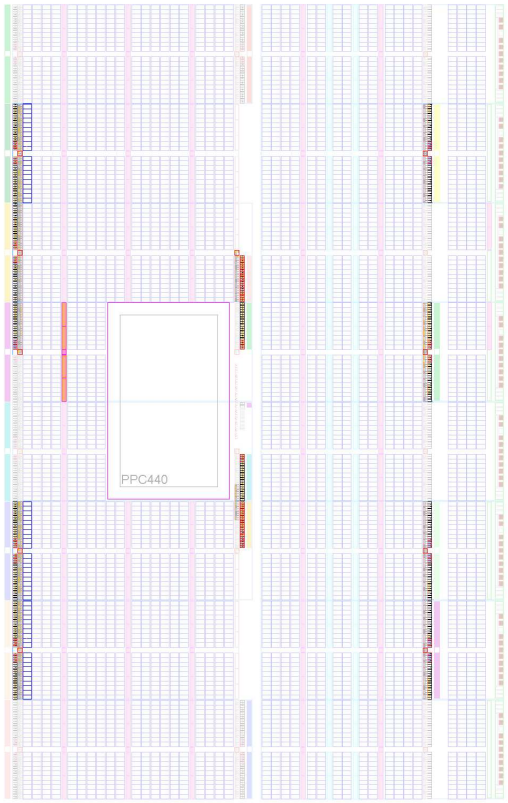


Figura 4.5: Restricciones del sistema base impuestas por EDK

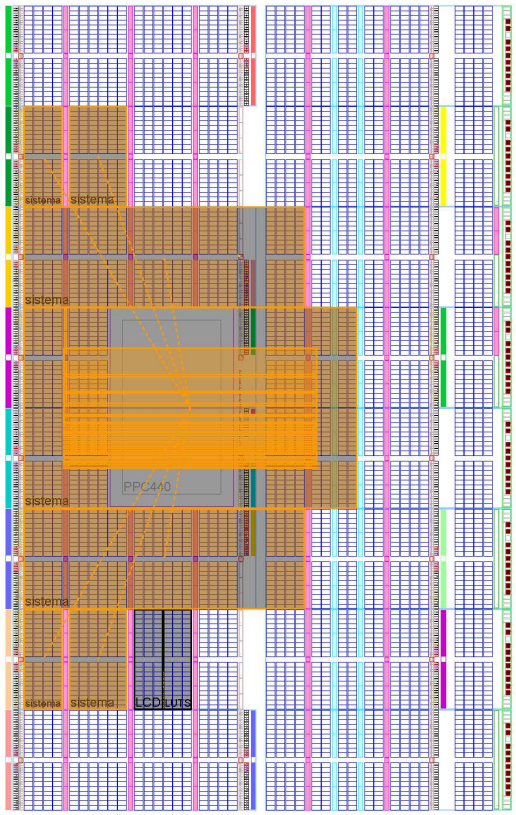


Figura 4.6: Área reservada para el sistema base.

Por estas razones se ha seleccionado para el sistema base, un área que contiene los recursos circundantes al procesador PowerPC®y que se extiende desde los pines requeridos para la memoria DDR2 hasta la primera columna de bloques DSP. La Figura 4.6 muestra el área asignada al sistema base. Otras alternativas estudiadas se muestran en la Tabla 4.1.

Se observa que *cuadrado*, la distribución seleccionada, da prioridad a la utilización de los recursos que rodean al procesador PowerPC®. Esto ha permitido cumplir de mejor forma los requerimientos de tiempo. Por otro lado se están dejando completamente libre solamente 6 regiones de reloj del FPGA, lo cual es un inconveniente cuando se requiera implementar gran cantidad de periféricos con diferentes frecuencias de trabajo. De cualquier forma si la aplicación final lo requiriera, es posible seleccionar cualquiera de las alternativas presentadas, sin que esto afecte el funcionamiento del sistema base. En el Cuadro de Código 4.2, se detallan las modificaciones al archivo UCF introducidas.

Tabla 4.1: Algunas distribuciones estudiadas para ser asignadas al sistema base

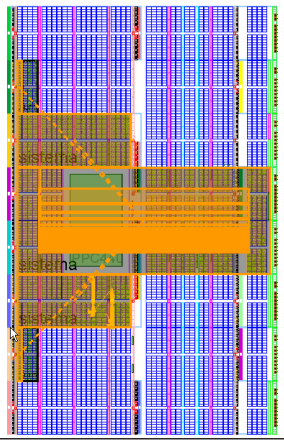
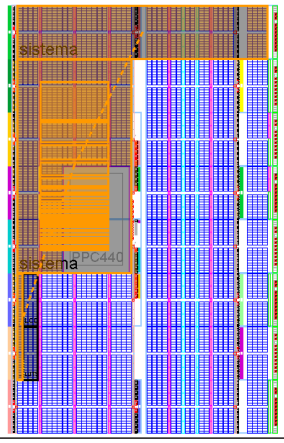
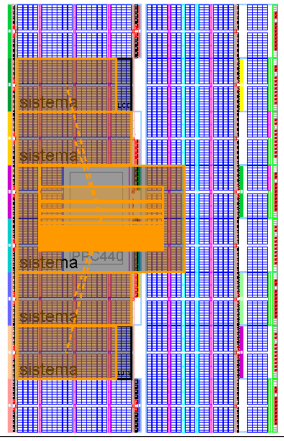
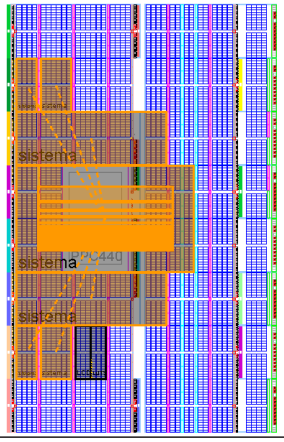
Distribución	Ventajas	Desventajas	Recursos
FranjaHorizontal 	• Emplea de forma completa las regiones del reloj X0Y2:X0Y5, X1Y3:X1Y4 y parcialmente los X0Y1, X0Y6 sin involucrar los X0Y0, X0Y7, X1Y0:X1Y2, X1Y5:X1Y7. • Tiene la relación de recursos LUT y FFD más alta.	• Segmenta el área libre en dos partes. • La relación de bloques de memoria requeridos sobre reservados es baja y la de los bloques DSP es la más baja de todas las configuraciones.	% de utilización de recursos. $\frac{R_{requeridos}}{R_{reservados}}$ $\text{LUT} = \frac{12.877}{15.040} = 86\%$ $\text{FFD} = \frac{9.723}{15.040} = 65\%$ $\text{BRAM} = \frac{25}{50} = 50\%$ $\text{BDSP} = \frac{13}{32} = 41\%$
LaF_V2 	• Al igual que en la FranjaHorizontal, esta configuración emplea completamente 6 regiones de reloj, de forma parcial 2 regiones y deja libres 8 regiones. • El área libre está en un solo bloque con una forma bastante regular. • La relación de bloques RAM y bloques DSP requeridos sobre los reservados mejora.	• La relación de recursos requeridos sobre reservados se ha incrementado con respecto a la FranjaHorizontal. • Algunos pines reservados por EDK se encuentran alejados de la zona del sistema base, lo cual exige algunos recursos de enrutado en la zona libre. • La zona que rodea el procesador PowerPC® no es empleada. Esto hace que sea más difícil cumplir los requerimientos de tiempos.	% de utilización de recursos. $\frac{R_{requeridos}}{R_{reservados}}$ $\text{LUT} = \frac{12.877}{15.360} = 84\%$ $\text{FFD} = \frac{9.723}{15.360} = 64\%$ $\text{BRAM} = \frac{25}{52} = 48\%$ $\text{BDSP} = \frac{13}{16} = 82\%$
LaT_V2 	• La zona del sistema base se hace más compacta. • Al igual que en las versiones anteriores se tienen 8 regiones de reloj libres. • Contiene bloques DSP necesarios para la implementación de la unidad de punto flotante del PowerPC®.	• Tiene la menor eficiencia de utilización de bloques RAM debido a que reserva 64 pero solo usa 25. • Es la distribución que más recursos LUT y FFD reserva.	% de utilización de recursos. $\frac{R_{requeridos}}{R_{reservados}}$ $\text{LUT} = \frac{12.877}{16.320} = 79\%$ $\text{FFD} = \frac{9.723}{16.320} = 60\%$ $\text{BRAM} = \frac{25}{64} = 40\%$ $\text{BDSP} = \frac{13}{16} = 82\%$

Tabla 4.1 – continuación desde la página anterior

Distribución	Ventajas	Desventajas	Recursos
cuadrado 	• Es la distribución que mejor rodea el procesador PowerPC® • Su área, contiene los bloques DSP requeridos para la unidad de punto flotante.	• Ocupa parcialmente mayor cantidad de regiones de reloj	% de utilización de recursos. $\frac{R_{requeridos}}{R_{reservados}}$ LUT $= \frac{12.877}{16.000} = 81\%$ FFD $= \frac{9.723}{16.000} = 61\%$ BRAM $= \frac{25}{40} = 63\%$ BDSP $= \frac{13}{16} = 82\%$

```
INST "apu_fpu_virtex5_0" AREA_GROUP = "sistema";
INST "clock_generator_0" AREA_GROUP = "sistema";
INST "DDR2_SDRAM" AREA_GROUP = "sistema";
INST "DIP_Switches_8Bit" AREA_GROUP = "sistema";
INST "fcb_v20_0" AREA_GROUP = "sistema";
INST "FLASH_util_bus_split_0" AREA_GROUP = "sistema";
INST "FLASH" AREA_GROUP = "sistema";
INST "Hard_Ethernet_MAC" AREA_GROUP = "sistema";
INST "IIC_EEPROM" AREA_GROUP = "sistema";
INST "jtagppc_cntlr_0" AREA_GROUP = "sistema";
INST "LEDs_8Bit" AREA_GROUP = "sistema";
INST "LEDs_Positions" AREA_GROUP = "sistema";
INST "plb_v46_0" AREA_GROUP = "sistema";
INST "ppc440_0" AREA_GROUP = "sistema";
INST "proc_sys_reset_0" AREA_GROUP = "sistema";
INST "Push_Buttons_5Bit" AREA_GROUP = "sistema";
INST "RS232_Uart_1" AREA_GROUP = "sistema";
INST "SysACE_CompactFlash" AREA_GROUP = "sistema";
INST "xps_bram_if_cntlr_1_bram" AREA_GROUP = "sistema";
INST "xps_bram_if_cntlr_1" AREA_GROUP = "sistema";
INST "xps_hwicap_0" AREA_GROUP = "sistema";
INST "xps_intc_0" AREA_GROUP = "sistema";
INST "xps_timebase_wdt_1" AREA_GROUP = "sistema";
INST "xps_timer_1" AREA_GROUP = "sistema";

AREA_GROUP "sistema" RANGE=SLICE_X48Y60:SLICE_X55Y99, SLICE_X20Y40:SLICE_X47Y119;
AREA_GROUP "sistema" RANGE=SLICE_X0Y20:SLICE_X19Y139;
AREA_GROUP "sistema" RANGE=DSP48_X0Y24:DSP48_X0Y39;
AREA_GROUP "sistema" RANGE=RAMB36_X3Y12:RAMB36_X3Y19, RAMB36_X0Y8:RAMB36_X2Y23;
AREA_GROUP "sistema" GROUP=CLOSED;
AREA_GROUP "sistema" PLACE=CLOSED;
```

Cuadro de Código 4.2: Restricciones impuestas al sistema en la configuración *cuadrado*.

4.1.3.2 Restricciones del periférico reconfigurable.

El periférico de prueba fue creado con el objetivo de realizar pruebas reconfigurando su estructura para modificar su función. Originalmente el periférico lleva a cabo la función de la compuerta AND bit a bit

entre 6 registros de 32 bits y arroja el resultado en un último registro; por ello su estructura está basada en 32 LUTs de 6 entradas. Estas LUTs serán modificadas para llevar a cabo una función lógica díferente y así demostrar el funcionamiento de la reconfiguración.

Para llevar a cabo esta tarea fue necesario restringir la ubicación de estas LUTs en el diseño. Por ello se ha restringido tanto la ubicación de los elementos del periférico, como la ubicación específica de estas LUTs. Las modificaciones añadidas al archivo de restricciones UCF se presentan en el cuadro de código 4.3.

```
# Restricciones para el perif'erico de prueba compuesto por LUTs
INST "andsobreluts_0/andsobreluts_0/USER_LOGIC_I/LasLUTs[0].Inst" BEL = A6LUT;
INST "andsobreluts_0/andsobreluts_0/USER_LOGIC_I/LasLUTs[0].Inst" LOC = SLICE_X26Y20;

INST "andsobreluts_0" AREA_GROUP = "LUTS";
AREA_GROUP "LUTS" RANGE=SLICE_X26Y20:SLICE_X31Y39;
AREA_GROUP "LUTS" GROUP=CLOSED;
AREA_GROUP "LUTS" PLACE=CLOSED;
```

Cuadro de Código 4.3: Restricciones impuestas sobre las LUTs instanciadas en el periférico reconfigurable

4.1.3.3 Regreso de archivos a EDK

Una vez realizada la implementación en PlanAhead, se generó el archivo *bitstream* `system.bit`, pero para empalmar nuevamente el flujo de diseño de EDK se debe generar igualmente el archivo bmm como se mostró en la Figura 4.4. Este archivo indica la ubicación física de los bloques de memoria que componen el periférico `xps_bram_if_cntlr_1`. Este periférico contiene el código de máquina del prebootloader.

Normalmente esta información es extraida automáticamente por EDK, pero cuando se manejó la implementación con PlanAhead, fue necesario consultar allí mismo esta información. Para ello se empleó la línea de comandos tcl que tiene implementado PlanAhead y se hizo el script del Anexo B. Este script, básicamente solicita reportes sobre la ubicación de las celdas (*cells*) después de ejecutar `par.exe` (*Place and Route*) e imprime la información según la sintaxis del archivo bmm.

Una vez generado este archivo, se empleó la aplicación `data2mem` que toma como entrada un *bitstream* y actualiza el contenido de los bloques RAM del FPGA. En este caso inserta el programa del prebootloader en los bloques RAM que empleo la implementación de PlanAhead generando el archivo `download.bit` que finalmente será descargado al FPGA. La línea de comando para usar `data2mem` fue la siguiente:

```
EDK# data2mem -bm "implementation/system_bd" -bt \
"implementation/system.bit" -bd "u-boot/executable.elf" \
tag ppc440_0  -o b implementation/download.bit
```

4.2 Driver del periférico XPS_HWICAP

Para llevar a cabo una aplicación *software* que emplee el periférico, es necesario hacer un *driver* dentro del sistema operativo que acceda correctamente el mismo. La implementación del *driver* se basó en el *driver* que Xilinx® da para manejar el mismo desde EDK. Y fue llevado a lo largo de varias etapas para su adaptación al sistema operativo Linux. El proceso que se llevó a cabo durante el desarrollo del proyecto, se espera quede como un lineamiento que pueda solucionar problemas similares en el futuro.

4.2.1 *Driver* de Xilinx del XPS_HWICAP

Una vez añadido el periférico XPS_HWICAP, es posible manipularlo con la librería que Xilinx® ofrece junto al periférico. Esta librería fue fundamental en el desarrollo del proyecto, porque ofrece información sobre la configuración del FPGA y porque se tomó como base para el *driver* que funcionó sobre Linux.

Con el objetivo de entender la forma en la que se relacionan las funciones que lo componen y con el objetivo de determinar una estrategia para compilarlo en el sistema operativo, se analizó la dependencia de archivos y de funciones. De este análisis surgió la gráfica de la Figura 4.7, donde se exponen algunos archivos fuente del *driver* que contienen declaraciones de funciones. Cada flecha indica que una función hace un llamado a otra. Así se identifican funciones en 4 niveles o capas.

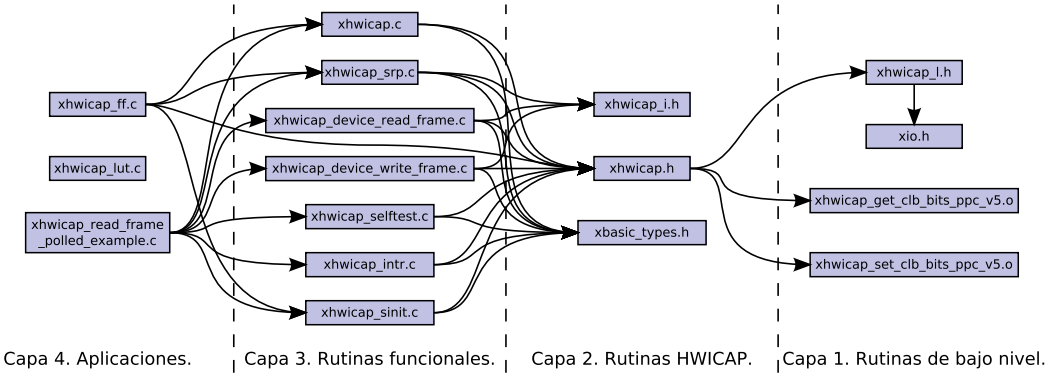


Figura 4.7: Dependencia en los archivos del código fuente del *driver* del HWICAP dado por Xilinx® .

En la primera capa se encuentran las funciones básicas con las que inicia la plantilla de un periférico en EDK, mediante las cuales se puede acceder a un registro del periférico. Estas funciones se encuentran en el archivo `xhwicap_l.h` y emplean funciones básicas de Xilinx® como lo son las que se encuentran definidas en `xio.h`. Estas son:

`XHwIcap_mWriteReg:` y **`XHwIcap_mReadReg:`** Permiten escribir y leer sobre un registro del periférico conociendo su *baseaddress* y su *offset*.

En la segunda capa se encuentran funciones relacionadas con el funcionamiento del módulo HWICAP. Algunas de ellas relacionadas con la escritura y lectura de los registros del periférico, otras relacionadas con consultas del estado del periférico, transferencias de datos, etc. Estas funciones en algunos casos se encuentran definidas como macros y están en su gran mayoría en el archivo `xhwicap.h`.

En esta capa se encuentran definidas dos funciones cuyo código fuente no se hizo público por parte de Xilinx® , sino que se distribuyen como archivos objeto. Estas funciones son:

`XHwIcap_SetClbBits` y `XHwIcap_GetClbBits`: Permiten modificar y leer los bits de configuración de un CLB

En la tercera capa se encuentran rutinas de tipo funcional, que involucran acciones de la capa dos. Entre estas rutinas se encuentran la escritura y lectura de datos mediante el periférico, pruebas de autodiagnóstico, de reset, etc. Estas rutinas se encuentran declaradas en varios archivos: `xhwicap.c`, `xhwicap_srp.c`, `xhwicap_selftest.c`, `xhwicap_intr.c`, `xhwicap_device_read_frame.c`, `xhwicap_device_write_frame.c` y `xhwicap_sinit.c`. Entre las más importantes funciones se encuentran:

`XHwIcap_DeviceReadFrame` y `XHwIcap_DeviceWriteFrame`: Permiten leer y escribir un frame completo del FPGA indicando los parámetros de su dirección.

`XHwIcap_selftest`: Realiza una prueba habilitando y desabilitando la interupción en el dispositivo.

Finalmente la cuarta capa es una capa de aplicación que se compone de ejemplos que emplean las funciones del *driver* de las capas inferiores. Entre estos ejemplos se encuentran los siguientes:

`xhwicap_ff`: Este ejemplo busca modificar el valor que almacena un flipflop específico, al interior de un slice del FPGA. Solo se garantiza su funcionamiento en FPGA de la familia Virtex 4.

`xhwicap_lut`: Este ejemplo busca verificar que todos los posibles valores que admite una LUT pueden ser escritos y leidos. Su implementación se puede realizar sobre dispositivos Virtex 4, 5 y 6.

`xhwicap_read_frame_polled_example`: Este ejemplo, lee un frame para demostrar el uso de la función `XHwIcap_DeviceReadFrame()` de la capa 3. Este ejemplo soporta su implementación en dispositivos de la familia Virtex 4, 5 y 6.

La estructura descrita se ha mantenido en las 4 versiones que habían aparecido hasta el momento del análisis del *driver*. Al momento de añadir el periférico, se realizaron pruebas desde el entorno de EDK empleando la versión 2.01, la cual era la más moderna que había sido distribuida con la versión 10.1 de

EDK. Estas pruebas básicamente incluían la ejecución de la función de inicialización del periférico, el autodiagnóstico (*selftest*) y la lectura de un frame, con su respectiva impresión en el terminal.

Al momento de llevar el *driver* al entorno del sistema operativo, se decidió implementar la versión 4, la cual era la última que se había distribuido hasta ese momento. En esta versión hay algunas correcciones realizadas sobre las versiones anteriores, principalmente permitiendo que los ejemplos que ellos otorgan puedan ser implementados sobre un mayor número de dispositivos. Aun así, no todos los ejemplos de la distribución pueden ser compilados para el FPGA `XC5VFX70TFFG1136`.

4.2.2 Compilación en Linux desde el espacio del usuario

El proceso de adaptar un *driver* de EDK para que trabaje en el *kernel* de Linux se hizo en dos etapas. La primera etapa logró que las fuentes se pudieran compilar en Linux, supliendo todas las funciones de bajo nivel que EDK posee haciendo una implementación desde el espacio del usuario. Durante la segunda etapa se mejoró la implementación al realizar las funciones de bajo nivel en el espacio del *kernel*.

Para poder llevar las fuentes a Linux y compilarlas, es necesario revisar el diagrama de dependencias de la Figura 4.7 e identificar las librerías de Xilinx® que son requeridas. La Figura 4.8 muestra la estrategia empleada para compilar las fuentes teniendo en cuenta la dependencia de los archivos de la fuente. Se ha escogido una implementación mediante librerías estáticas, pero puede ser modificada para que sean compartidas cuando varios *drivers* de EDK o varias aplicaciones requieran hacer uso de estas rutinas.

Como resultado de esa compilación se generaron las librerías `libxil_assert.a` y `libxil_io.a`, que definen rutinas y tipos de datos propios de los *drivers* de EDK. Empleando los archivos propios del *driver* del `HWICAP`, surge la librería `libxhwicap.a`. Estas tres librerías, junto a los cabeceros de las funciones son los requerimientos que tiene la compilación de una aplicación, como los ejemplos del *driver*.

Para que este flujo se pueda llevar a cabo fue necesario modificar algunas de las fuentes de Xilinx® y durante la implementación se buscó que los cambios no afectaran las fuentes del `HWICAP`, sino que preferiblemente afectaran a las de bajo nivel como `xil_io` y `xil_assert`, para que de esta forma otros *drivers* se puedan migrar a Linux sin mayores modificaciones.

Los cambios realizados sobre las fuentes se exponen en detalle en el Anexo D, pero se pueden resumir en los siguientes items:

Función de impresión: La mayoría de los ejemplos contiene una línea donde sustituye el `printf` de la librería estándar de C por `xil_printf`. Esta definición debe ser comentada manualmente en cada

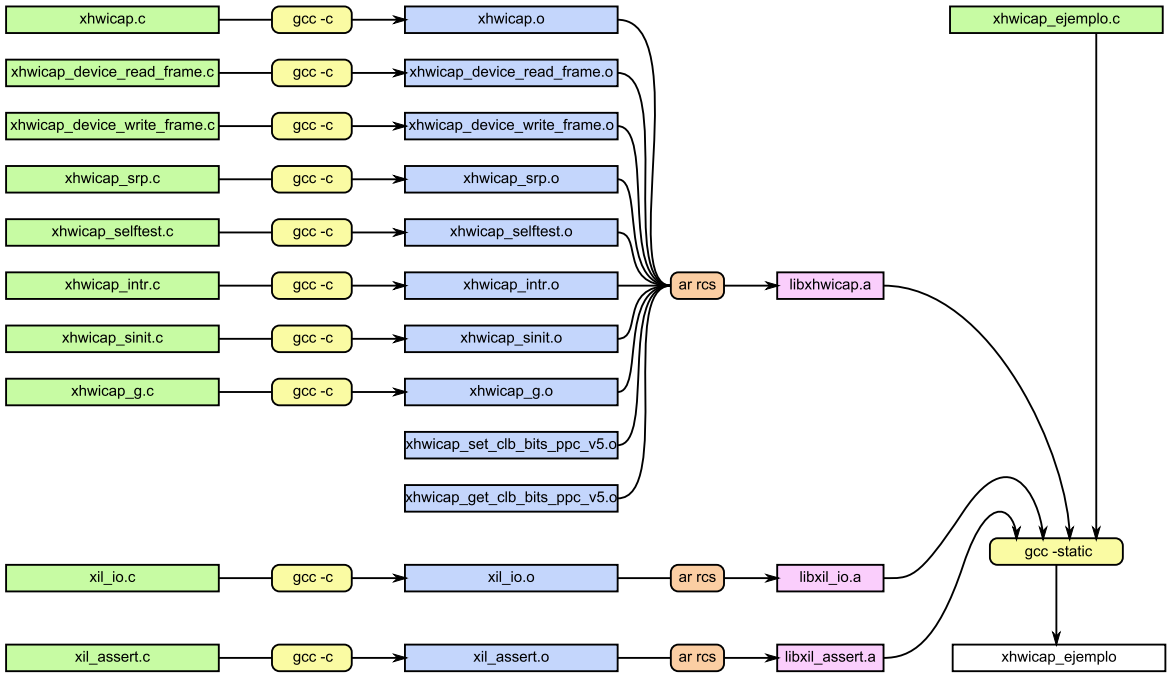


Figura 4.8: Compilación de las fuentes del *driver* y una aplicación sobre una fuente de ejemplo.

ejemplo.

Traslado de fuentes de EDK al directorio de compilación en el SO: Algunas fuentes y definiciones es mejor tomarlas directamente del proyecto de EDK. Para este proyecto fue necesario traer las fuentes:

- `xstatus.h`
- `xreg440.h`
- `xpseudo_asm_gcc.h`
- `xpseudo_asm.h`
- `xpseudo_asm.h`
- `xio.h`
- `xio.c`
- `xintc_l.h`
- `xintc.h`
- `xil_types.h`
- `xil_io.h`
- `xil_io.c`
- `xil_exception.h`
- `xil_assert.h`
- `xil_assert.c`
- `xhwicap_family.c`
- `ppc-asm.h`
- `xbasic_types.c`
- `xbasic_types.h`

Modificación de las funciones de escritura de registros: Los métodos que por defecto trae EDK para el acceso a los registros asumen que la aplicación que se desarrolla es la única que se está ejecutando. Esta suposición no es válida en el entorno que ofrece el sistema operativo, por lo tanto, los métodos que modifican y que leen los registros del periférico deben solicitarle al *kernel* el acceso a

los mismos. Los métodos contenidos en las rutinas de EDK en los archivos `xil_io.h` y `xil_io.c`, fueron modificados para cumplir con los métodos de acceso del sistema operativo.

Durante esta primera etapa, se accedió al mapa de memoria del periférico mediante el nodo `/dev/mem`, el cual ofrece un método sencillo y se puede llevar a cabo desde el espacio del usuario del sistema operativo. Los detalles de esta implementación se pueden observar en el Anexo E.

4.2.3 Ejemplos modificados y scripts de Linux hechos para pruebas

Mediante las modificaciones descritas en la sección anterior se logró compilar los ejemplos del *driver* que son compatibles con los FPGA Virtex 5. Algunos de estos ejemplos son bastante útiles para recolectar información sobre el dispositivo y sobre la estructura del *bitstream*, por eso fueron modificados para ofrecer una mayor funcionalidad. A continuación se presentan las modificaciones realizadas sobre los ejemplos y algunos scripts que los usan:

4.2.3.1 xhwicap_read_frame_polled_example_args.c

Esta es una modificación del ejemplo `xhwicap_read_frame_polled_example`, la cual permite modificar la forma en la que se imprimen los frames y además permite introducir por línea de comandos cual de los frames del FPGA se desea leer. Las opciones de visualización se resumen en la Tabla 4.2.

Tabla 4.2: Opciones de visualización de xhwicap_read_frame_polled_example_args.c

Opción	Descripción
`--debug`	Imprime cabeceros al inicio de cada frame que muestran los parámetros de su dirección como el tipo de bloque, top, fila, major y minor. Ejemplo: `FRAME=> Block=0 Top=1 Hclk/Row=1 Major=1 Minor=20`
`--hex`	Imprime en una columna la información del frame en hexadecimal. Ejemplo: `21000303`
`--bin`	Imprime en una columna la información del frame en binario. Ejemplo: `00100001000000000000001100000011`
`--info`	Imprime en frente de cada palabra del frame su índice. Ejemplo: `Frame Word 70 -> 21000303`

Continua en la siguiente página

Tabla 4.2 – continuación desde la página anterior

Opción	Descripción
`--quit_null_frame`	No imprime un frame adicional que es leido siempre que se hace la lectura de un frame. Este frame es cero en su totalidad y no contiene información.

Este programa fue empleado como base para otros que requieren leer un frame en específico. Su código fuente y una explicación mas detallada de sus opciones se encuentra en la sección F.1.

4.2.3.2 leer_frames_de_una_columna.sh

En las pruebas realizadas y en la documentación referente al tema, se ha logrado identificar los frames que contienen la información de cada una de las columnas. Sin embargo la documentación no especifica exactamente cual frame contiene la información de cada uno de los recursos ubicados en las diferentes columnas. Por ello este script lee el contenido de los frames asociados a una columna completa del FPGA y lo imprime en consola. Su salida puede ser redireccionada a un archivo para posteriores comparaciones. Su código fuente y un ejemplo de uso se encuentra en la sección F.2.

4.2.3.3 leer_toda_la_configuracion.sh

Este script hace un recorrido por todo el rango de direcciones del FPGA y busca leer toda la configuración en tiempo de ejecución. En la variable `num_minors` está almacenada la información sobre la cantidad de frames que contiene cada columna del FPGA y de esa forma se realiza todo el recorrido por el rango de direcciones de los frames. Normalmente este script nunca llega a feliz término debido a que la lectura de ciertos frames hace que el sistema se bloquee y sea necesario reiniciar. La sección F.3 muestra el código fuente del script y su modo de uso.

4.2.3.4 w_lee_y_escribe_frame.c

Este programa se desarrolló basado en los ejemplos del *driver*, y empleando las funciones del mismo. Su objetivo es modificar solamente un bit de una palabra de un frame específico. Por ello desde la línea de comandos el programa puede recibir los campos de la dirección del frame, el número de la palabra y el bit específico que se quiere invertir. Para llevar a cabo esta acción se hace uso de las funciones del *driver* `XHwIcap_DeviceReadFrame` y `XHwIcap_DeviceWriteFrame`. El código fuente y una explicación de cada una de las opciones de la función se encuentra en el Anexo F.4.

4.2.3.5 w_cambia_cada_bit_de_una_columna.sh

Este script hace uso del programa `w_lee_y_escribe_frame.c` para recorrer cada uno de los bits de los frames de una columna, modificándolo y llevándolo nuevamente a su estado original. Este script fue usado

en conjunto con otros que permiten conocer el contenido de los LUTs para encontrar los bits exactos que modifican el comportamiento de la LUT. Su código fuente se encuentra en la sección F.5.

4.2.4 Identificación de los bits que afectan el comportamiento de las LUTs.

Con el *driver* funcionando desde el espacio del usuario, la experiencia del manejo del periférico desde el espacio del usuario y el conocimiento de la arquitectura del FPGA, se logró inferir exáctamente cuáles bits son los que modifican el comportamiento de cada LUT del sistema. Pero para ello fue necesario hacer uso de las herramientas desarrolladas y crear un programa adicional que hace uso del periférico de prueba descrito en la sección 4.1.1.

4.2.4.1 Programa operaLUTs y operaLUTs_verbose

Este programa tiene en cuenta la arquitectura del periférico reconfigurable de prueba y determina la tabla de verdad de cada una de las LUTs que lo componen. Su objetivo principal es determinar cuantas LUTs han sido modificadas y su resultado es mostrado en consola al ejecutar `operaLUTs`. El programa `operaLUTs_verbose` imprime en consola la tabla de verdad de cada una de las tablas de verdad de los LUTs con un comentario adicional en caso de comportarse como una compuerta AND, NAND, OR o NOR. En el estado inicial del periférico reconfigurable, la salida de `operaLUTs_verbose` es como se muestra en el Cuadro de Código 4.4 y su código fuente se encuentra en el Anexo F.6.

El número 32 de la última línea indica que todas las LUTs mantienen su estado original. El código fuente de ambos programas es el mismo, pero para generar los dos ejecutables se modifica una definición del precompilador en el Makefile del Anexo D.2.

4.2.4.2 FPGA Editor

FPGA Editor es una herramienta que Xilinx® ofrece para acceder de forma bastante detallada a la información de mapeo y de enrutamiento. Esta herramienta fue de gran importancia en el proceso de identificación de los bits que definen el comportamiento de una LUT debido a los siguientes dos aportes:

- Inicialmente permitió verificar que las restricciones impuestas al diseño se estaban cumpliendo. Aquí se observa la ubicación y la configuración de cada LUT empleada en el periférico de prueba, y adicionalmente se observa el orden de entrada de los bits a la LUT.
- El aporte más importante del *software* fue el de realizar ingeniería inversa, dado que no solo permite visualizar los resultados del `par`, sino que además permite modificarlo. Con esto se pudo realizar pequeños cambios en la configuración de los LUTs y generar diferentes archivos *bitstreams* que fueron comparados.

```
Okinawa-21:etapa1_espacio_usuario# ./prueba_LUTs/operaLUTs/operaLUTs_verbose
Resultados:
LUT00 : 8000000000000000 Compuerta AND de 6 entradas
LUT01 : 8000000000000000 Compuerta AND de 6 entradas
LUT02 : 8000000000000000 Compuerta AND de 6 entradas
LUT03 : 8000000000000000 Compuerta AND de 6 entradas
LUT04 : 8000000000000000 Compuerta AND de 6 entradas
LUT05 : 8000000000000000 Compuerta AND de 6 entradas
LUT06 : 8000000000000000 Compuerta AND de 6 entradas
LUT07 : 8000000000000000 Compuerta AND de 6 entradas
LUT08 : 8000000000000000 Compuerta AND de 6 entradas
LUT09 : 8000000000000000 Compuerta AND de 6 entradas
LUT10 : 8000000000000000 Compuerta AND de 6 entradas
LUT11 : 8000000000000000 Compuerta AND de 6 entradas
LUT12 : 8000000000000000 Compuerta AND de 6 entradas
LUT13 : 8000000000000000 Compuerta AND de 6 entradas
LUT14 : 8000000000000000 Compuerta AND de 6 entradas
LUT15 : 8000000000000000 Compuerta AND de 6 entradas
LUT16 : 8000000000000000 Compuerta AND de 6 entradas
LUT17 : 8000000000000000 Compuerta AND de 6 entradas
LUT18 : 8000000000000000 Compuerta AND de 6 entradas
LUT19 : 8000000000000000 Compuerta AND de 6 entradas
LUT20 : 8000000000000000 Compuerta AND de 6 entradas
LUT21 : 8000000000000000 Compuerta AND de 6 entradas
LUT22 : 8000000000000000 Compuerta AND de 6 entradas
LUT23 : 8000000000000000 Compuerta AND de 6 entradas
LUT24 : 8000000000000000 Compuerta AND de 6 entradas
LUT25 : 8000000000000000 Compuerta AND de 6 entradas
LUT26 : 8000000000000000 Compuerta AND de 6 entradas
LUT27 : 8000000000000000 Compuerta AND de 6 entradas
LUT28 : 8000000000000000 Compuerta AND de 6 entradas
LUT29 : 8000000000000000 Compuerta AND de 6 entradas
LUT30 : 8000000000000000 Compuerta AND de 6 entradas
LUT31 : 8000000000000000 Compuerta AND de 6 entradas
32
```

Cuadro de Código 4.4: Implementación de las LUTs en el periférico de pruebas

4.2.4.3 Estrategia para identificar los bits del *bitstream*

Con el objetivo de establecer cuáles de los bits modifican el comportamiento de las LUTs del FPGA y poder de esta forma manipular el comportamiento de la tabla de verdad de las mismas se siguió la estrategia mostrada en la Figura 4.9. Partiendo del trabajo de ruteo realizado por Planahead, se tomó el archivo NCD enrutado y se manipuló con FPGA editor para en algunos casos invertir la salida de la función lógica de la LUT y en otros casos para convertirla en una compuerta OR. Esto con el objetivo de obtener la mayor cantidad de información de esta prueba. De esta forma se obtuvo un segundo archivo NCD.

Estos dos archivos NCD se pasaron por el programa `bitgen`, el cual arrojó como resultado un *bitstream* para cada uno de ellos, los cuales se analizaron mediante el script de matlab `analizobitstream.m`. Este script segmenta el cabecero y cada uno de los frames del *bitstream* y arroja como resultado un archivo de texto para cada archivo de entrada. Estos archivos de texto fueron comparados por la utilidad `VimDiff` que arrojó las siguientes diferencias:

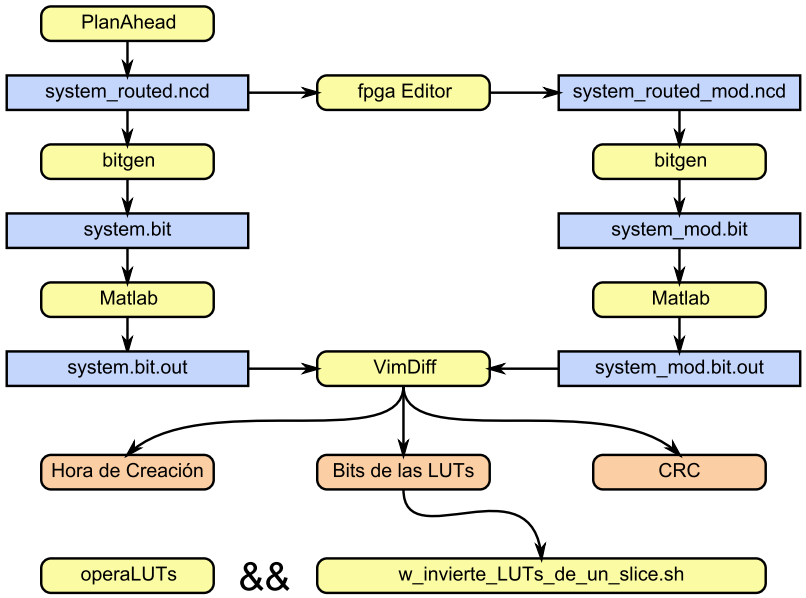


Figura 4.9: Estrategia para identificar los bits que modifican el comporamiento de las LUT de un slice

- Fecha de creación del archivo .bit.
- Cambios en las primeras 16 palabras de los frames con $block = 0$, $row = 2$, $top = 1$, $major = 16$ y $minor = 32 \cdots 35$.
- Cambio en el valor del CRC.

El primer y tercer cambio, se pueden predecir y no son relevantes. La segunda diferencia encontrada indica los bits exactos que determinan la tabla de verdad de la LUT. La cantidad de bits modificados durante la prueba fueron $32 \times 16 \times 4 = 2048\,bits$, y sabiendo que la cantidad de bits necesarios para establecer la tabla de verdad de 32 LUTs de 6 entradas es $2^6 \times 32 = 2048\,bits$, pues fortaleció la hipótesis de que estos bits son los que realmente modifican el comportamiento de las tablas de verdad.

Ahora el siguiente paso fue establecer cómo afecta cada uno de los bits la tabla de verdad de la LUT. Para ello se hizo uso de la implementación del *driver* en el sistema operativo desde el espacio del usuario y se puso a correr indefinidamente el programa `operaLUTs` que imprime la tabla de verdad de las 32 LUTs. Por otro lado, se usó la información de la ubicación de los bits que modifican el comportamiento de las LUTs, obtenida mediante `VimDiff`, y se ajustaron los parámetros del script `w_cambia_cada_bit_de_una_columna.sh` para crear el script `w_invierte_LUTs_de_un_slice.sh` el cual sólamente cambia los bits de las LUTs.

Los efectos que realiza el script se van haciendo evidentes en la salida del programa `operaLUTs` y de esa forma mediante observación se pudo analizar el efecto de modificar cada bit de las LUTs. Al final de la prueba se tuvo como resultado que el comportamiento de cada LUT se había invertido, como se esperaba.

4.2.4.4 Análisis de los resultados

Cruzando el orden en el que se modificaron los bits y la forma en la que fueron cambiando los bits en la tabla de verdad se puede en el futuro manipular más fácilmente el comportamiento de las LUTs. Este orden se muestra en las Tablas 4.3 y 4.4. En la Tabla 4.3 se observa el orden en el que el script `w_invierte_LUTs_de_un_slice.sh` manipuló los bits de los frames y en la Tabla 4.4 se observa el orden en el que fueron cambiando en la tabla de verdad.

minor	**word**	**bits**																													
		31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
32	**0**	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
32	**1**	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
33	**0**	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
33	**1**	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
34	**0**	160	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129
34	**1**	192	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161
35	**0**	224	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193
35	**1**	256	255	254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225

Tabla 4.3: Orden en el que el script `w_invierte_LUTs_de_un_slice.sh` manipuló los bits de los frames

LUT	**bits**																														
	63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
D	49	113	241	177	114	50	178	242	51	115	243	179	116	52	180	244	53	117	245	181	118	54	182	246	55	119	247	183	120	56	184	248
C	33	97	225	161	98	34	162	226	35	99	227	163	100	36	164	228	37	101	229	165	102	38	166	230	39	103	231	167	104	40	168	232
B	17	81	209	145	82	18	146	210	19	83	211	147	84	20	148	212	21	85	213	149	86	22	150	214	23	87	215	151	88	24	152	216
A	1	65	193	129	66	2	130	194	3	67	195	131	68	4	132	196	5	69	197	133	70	6	134	198	7	71	199	135	72	8	136	200

LUT	**bits**																														
	31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
D	57	121	249	185	128	64	192	256	59	123	251	187	124	60	188	252	61	125	253	189	126	62	190	254	63	127	255	191	128	64	192	256
C	41	105	233	169	112	48	176	240	43	107	235	171	108	44	172	236	45	109	237	173	110	46	174	238	47	111	239	175	112	48	176	240
B	25	89	217	153	96	32	160	224	27	91	219	155	92	28	156	220	29	93	221	157	94	30	158	222	31	95	223	159	96	32	160	224
A	9	73	201	137	80	16	144	208	11	75	203	139	76	12	140	204	13	77	205	141	78	14	142	206	15	79	207	143	80	16	144	208

Tabla 4.4: Orden en el que el script manipuló los bits de las 4 LUTs del un slice

De estas tablas se han deducido las siguientes fórmulas que permiten determinar el frame y los bits precisos para manipular un resultado de una LUT específica dentro del FPGA. Con estas fórmulas se implementaron las funciones `calcula_word`, `calcula_bit` y `calcula_frame_minor` contenidas en la fuente `w_escribe_una_LUT.c` para posteriores etapas. La fuente del programa `w_escribe_una_LUT.c` se encuentran en el Anexo F.7.

$$B = 15 - \left\lfloor \frac{E}{4} \right\rfloor + 16\,[LUT(mod\ 2)] \tag{4.1}$$

$$W = 2\,[S_y(mod\ 10)] + \left\lfloor \frac{LUT}{2} \right\rfloor \tag{4.2}$$

$$m = 26 + 6\,[S_x + 1(mod\ 2)] + \Phi\,[E(mod\ 8)] \tag{4.3}$$

donde:

E → Combinación de entrada dentro de la LUT. Puede tomar valores entre 0 y 63.

LUT → Número de la LUT al interior del Slice. Se ha codificado de la siguiente forma: $A \to 0$, $B \to 1$, $C \to 2$ y $D \to 3$.

S_x → Coordenada x del slice según el sistema de coordenadas usado en los archivos `ucf` de restricciones.

S_y → Coordenada y del slice según el sistema de coordenadas usado en los archivos `ucf` de restricciones.

B → Indica cuál de los 32 bits de la palabra se debe modificar. El bit 0 es el LSB y el 31 el MSB.

W → Señala cuál palabra dentro del frame contiene el bit solicitado.

m → Indica el minor address del frame que contiene el bit solicitado.

$\Phi(x)$ → Es una función definida de Z_8 en Z_4 así:

- $\Phi(0) = 3$
- $\Phi(2) = 0$
- $\Phi(4) = 2$
- $\Phi(6) = 1$
- $\Phi(1) = 2$
- $\Phi(3) = 1$
- $\Phi(5) = 3$
- $\Phi(7) = 0$

4.2.5 Compilación en Linux desde el espacio del *kernel*

Una vez implementado el *driver* desde el espacio del usuario, se pudo obtener información importante sobre el funcionamiento del periférico y la arquitectura del FPGA. Adicionalmente se logró estructurar en gran parte el *driver* que finalmente quedará en la implementación final. Durante esta etapa, se implementó el *driver* que permite manipular el periférico `XPS_HWICAP` desde el espacio del *kernel*. Mediante este *driver*, se busca tener una implementación con mejor desempeño.

La función de un *driver* es generar los mecanismos de acceso a un recurso del sistema de cómputo. En el caso de un periférico, esto implica el acceso a los registros que controlan su funcionamiento y los registros por medio de los cuales se intercambia información. Adicionalmente dentro del sistema operativo Linux, todos los recursos son abstraidos como archivos, por lo tanto el *driver* también tiene que llevar a

cabo las acciones adecuadas en el momento que un usuario o un proceso acceda a dicho archivo.

Desde el punto de vista del usuario, o de un proceso que corre en el sistema operativo, el periférico aparece como un archivo cuya longitud es la cantidad de registros del periférico y para su acceso se deben ejecutar los llamados al sistema `open`, `close`, `write`, `read`, `lseek`, etc. Estos llamados al sistema son funciones que solicitan al *kernel* ejercer una acción sobre los archivos, pero en el caso de un periférico, estas funciones son atendidas por el *driver*. De esta forma el creador del *driver* puede decidir la acción en el perfiérico más adecuada según la operación solicitada desde el espacio del usuario.

4.2.5.1 Estructura de la implementación del *driver* desde el espacio del *kernel*

Para llevar a cabo la versión del *driver* desde el espacio del *kernel*, se optó por implementar únicamente las funciones de bajo nivel del *driver* desarrollado en el espacio del usuario, de esta forma se obtiene el beneficio que ofrece llevar a cabo las operaciones de lectura y escritura de bajo nivel en el *kernel*, mientras que las operaciones de mas alto nivel como escribir y leer frames se realizan desde el espacio del usuario y pueden ser modificadas mas fácilmente en futuras versiones.

La nueva estructura del *driver* no cambió significativamente, pero si hay que hacer unas modificaciones en las fuentes y crear el *driver* que trabajará desde el espacio del *kernel*.

4.2.5.2 Desarrollo del módulo del *kernel*

Con el objetivo de adquirir mayor destreza con el manejo de los datos en el espacio del *kernel*, el desarrollo del módulo se hizo de forma genérica para que pueda ser empleado para cualquier periférico y después fue adaptado para el XPS_HWICAP.

Como resultado de este proceso ha quedado una plantilla para cualquier periférico basado en registros conectados al bus, la cual se encuentra en el Anexo H. Allí mismo se presenta una pequeña guía para que pueda ser adaptado rápidamente a cualquier periférico. Este procedimiento fue aplicado sobre el periférico XPS_HWICAP.

El *driver* del periférico tiene a cargo las funciones de más bajo nivel, por lo tanto es muy similar a un *driver* genérico de cualquier periférico que se manipule mediante sus registros añadidos al bus principal del procesador. A continuación se describen las acciones que el *driver* lleva a cabo en cada una de las rutinas implementadas.

XPS_HWICAP_init: Esta función es invocada cuando se lleva a cabo desde el espacio el usuario el comando `insmod` que añade al *kernel* el *driver*. En esta función se realizan las siguientes funciones:

- Reservar los números `major` y `minor` para nuestro periférico.

- Reservar memoria en el espacio del *kernel* para una estructura de tipo XPS_HWICAP_dev que contiene información del periférico y que es utilizada por todas las funciones del mismo.
- Inicializar algunos campos de la estructura como un mutex que hace que el periférico solo sea abierto una vez.
- Inicializar el tipo de dato `cdev` del periférico para posteriormente registrarlo en el *kernel*.
- Poner en cero una bandera indicando que este periférico nunca ha sido abierto.

XPS_HWICAP_exit: esta función es llamada cuando se ejecuta el comando `rmmod` para retirar el *driver* del *kernel*. Esta función debe deshacer todo lo que `XPS_HWICAP_init` hizo, por eso cumple las siguientes funciones:

- Libera los número `major` y `minor` registrados anteriormente.
- Remueve el cdev del *kernel*.
- Libera la memoria reservada para mapear los registros. Esto se hace únicamente si se ha ejecutado la rutina de open. Por lo cual se pregunta por la bandera desactivada en init y activada en open.
- Libera la memoria reservada para la estructura XPS_HWICAP_dev.

XPS_HWICAP_open: Esta rutina está asociada con el llamado al sistema open, el cual busca tener acceso al dispositivo. En esta rutina se llevan a cabo las siguientes acciones:

- Asignar el campo `private_data` del puntero `file` con la estructura que contiene el `cdev`, de esta forma se tiene acceso a esta estructura desde cualquier función del *driver*.
- Se reserva el periférico activando el `mutex`.
- En caso de ser la primera vez que se ejecuta el open, se inicializan los campos de la estructura XPS_HWICAP_dev, esto incluye hacer la reserva de memoria donde se mapean los registros del periférico. Así mismo se activa la bandera que indica que ya se inicializó la estructura.
- Se realiza el ioremap para acceder a los registros del periférico.

XPS_HWICAP_release: Esta función está asociada con el llamado al sistema close. En esta función se deshacen la mayor parte de las acciones de la función XPS_HWICAP_open. Las acciones específicas son:

- Libera la memoria y lo necesario para acceder a los registros del periférico.
- Libera el periférico modificando el mutex.

XPS_HWICAP_write: Esta función se asocia con el llamado al sistema write y busca traer datos desde el espacio del usuario hacia el espacio del *kernel*. Realiza las siguientes acciones:

- Verifica que la cantidad de información que se va a escribir no sobrepasa la cantidad de registros del periférico. En caso de hacerlo, determina la cantidad de bytes que se pueden escribir.
- usa la función `copy_from_user` para traer la información desde el espacio del usuario.
- Actualiza los datos que se quieren escribir en la estructura, como una copia de la información que se escribe en los registros.
- Se escriben los datos en los registros mediante la función `iowrite32`.
- Se actualiza el puntero del archivo según la cantidad de datos escritos.
- Se devuelve la cantidad de datos escritos.

XPS_HWICAP_read: La función read está asociada con el llamado al sistema `read` y busca devolver información al usuario sobre los registros del periférico. Durante su ejecución se llevan a cabo las siguientes tareas:

- Verifica que la cantidad de información solicitada se pueda dar, en caso de que no, solo da la información disponible.
- Lee de los registros la cantidad de información solicitada.
- Cambia el endianness a los datos leidos.
- Envía los datos al usuario por medio de la función `copy_to_user`.
- Actualiza el puntero del archivo según la cantidad de datos devueltos.

XPS_HWICAP_llseek: Esta función se encarga de cambiar la posición actual del puntero. Su implementación se hizo de forma estándar, debido a que no requiere ejercer ninguna acción particular en el periférico.

XPS_HWICAP_iocntl: Esta función permite ejecutar funciones personalizadas según el periférico que se está manejando. En el caso del XPS_HWICAP no se requirió alguna acción particular.

4.2.5.3 Modificaciones sobre las librerías del espacio del usuario.

La librería `xil_io.c` en esta versión tiene la función de manipular el puntero dentro del archivo para que se dirija al registro específico que se desea acceder y luego escribe o lee el dato. Adicionalmente se debe modificar las fuentes para que en algún momento se ejecuten los llamados al sistema `open` y `close`. Inicialmente se llevaron a cabo estos llamados al sistema en las rutinas de la librería `xil_io.c` pero esto es ineficiente debido a la gran cantidad de escrituras y lecturas que se deben hacer en el periférico para

llevar a cabo una reconfiguración. Por ello se optó finalmente por hacer los llamados sólamente una vez en los archivos de mas alto nivel realizando las siguientes cuatro modificaciones:

Adición de librerías: Con el objetivo de que las rutinas que escriben en los registros del periférico, puedan manipular el nodo `/dev/periferico_XPS_HWICAP`, es necesario incluir las siguientes librerías

```
#include <stdlib.h>
#include <fcntl.h>
```

Cuadro de Código 4.5: Adición de las librerías `stdlib.h` y `fcntl.h`.

Declaración del puntero del archivo: Es necesario que las funciones de bajo nivel reciban el puntero con el que deben manipular el archivo del periférico, el cual se obtiene del llamado al sistema open, por eso se debe declarar de forma global en el archivo de más alto nivel:

```
size_t filedesc;
```

Cuadro de Código 4.6: Declaración del puntero del archivo.

Adición de la función open: La función open solo se debe hacer una vez para cada programa que use el periférico porque esto garantiza que solo él está usando el periférico en este momento. Cualquier otro proceso que quiera acceder al periférico debe esperar a que el proceso que lo tiene ocupado pues lo libere mediante la función close. Por ello se ubicó comenzando la función principal del programa.

```
        filedesc = open("/dev/Periferico_XPS_HWICAP",O_RDWR);
    /* En caso de no poder abrirlo, entonces se termina el programa          */
    if(filedesc < 0) {
        printf("No se pudo abrir el nodo /dev/Periferico_XPS_HWICAP\n");
        exit(EXIT_FAILURE);
    }
```

Cuadro de Código 4.7: Adición del llamado al sistema open.

Adición de la función close: Así como la función open, la función close libera el periférico y se lleva a cabo al final de la función principal del programa.

```
    /* Llamado al sistema close                                              */
    if(close(filedesc) < 0) {
        printf("No se pudo cerrar el nodo /dev/Periferico_XPS_HWICAP\n");
        exit(EXIT_FAILURE);
    }
```

Cuadro de Código 4.8: Adición del llamado al sistema close.

4.2.5.4 Nueva librería `xil_io.c`

La librería `xil_io.c` cambió completamente debido a que su forma de acceso al periférico cambió radicalmente. Su versión modificada se puede ver en el Anexo G. Ahora debe usar el puntero declarado en el archivo de mayor jerarquía para ajustarlo y escribir o leer adecuadamente los registros. Con el objetivo de mantener compatibilidad con las funciones de la versión que funcionó desde el espacio del usuario, el

prototipo de las funciones `Xil_Out32` y `Xil_In32` no se modificó.

Las funciones de lectura y escritura tienen una estructura similar. Inicialmente reciben la dirección a la que se desea escribir o leer, la cual es validada para garantizar que esta esté en el rango del periférico. Allí se calcula el *offset* del registro que se quiere acceder a partir del baseaddress del periférico. Posteriormente se ajusta el puntero del archivo mediante `lseek` y se hace la operación de lectura o escritura con `write` o `read`.

4.3 Resultados del capítulo

Al llevar a cabo el procedimiento del presente capítulo, se ha modificado el sistema base al añadirle dos periféricos nuevos. El primero es un periférico de prueba que permitió realizar pruebas de reconfiguración que finalmente determinaron los bits exactos que modifican el comportamiento de las LUTs del FPGA dentro de un archivo de configuración. Simultáneamente se requirió la implementación del periférico reconfigurable y de la puesta en marcha del *driver* que permitió hacer la reconfiguración del comportamiento de las LUTs. Con esto el sistema ML507 tiene los mecanismos para fácilmente manpular el comportamiento de cada una de sus LUTs mediante reconfiguración parcial.

Capítulo 5

Pruebas finales y conclusiones

5.1 Medición del tiempo de reconfiguración

La estrategia empleada para la medición del tiempo que tarda el proceso de reconfiguración se basó en funciones que trabajan en el espacio del *kernel*. Otra alternativa consiste en el uso de funciones en el espacio del usuario que consultan la hora y fecha actuales del sistema, sin embargo estas funciones tienen mayor impacto sobre la medida del tiempo debido a que no tienen los privilegios de acceso que tienen las funciones del espacio del *kernel*. Adicionalmente la resolución de la medida en muchos casos no es menor a $1ms$.

En el espacio del *kernel* se seleccionó la función `get_cycles()` para hacer la medición debido a que está implementada con instrucciones de ensamblador propias del procesador y además obtiene la mejor resolución posible. Esta función consulta la cantidad de ciclos de reloj que han sucedido desde que se encendió el sistema. Conociendo los ciclos de reloj y teniendo en cuenta que la frecuencia de trabajo del procesador es de $400\,MHz$, (ésto significa una resolución por ciclo de reloj de $\frac{1}{400\,MHz} = 2,5\,ns$), podemos medir el tiempo empleado en la reconfiguración.

Para llevar a cabo la medida se tomó una marca de tiempo al iniciar, en puntos intermedios y al terminar el proceso de reconfiguración. Gracias a que las aplicaciones que usan el *driver* se implementaron haciendo una única vez la apertura y el cierre del nodo `/dev/Periferico_XPS_HWICAP`, allí se tomaron las marcas de tiempo inicial y final. Para llevar a cabo las marcas de tiempo intermedias, fue necesario emplear el llamado al sistema `ioctl` desde el espacio del usuario para dar al programador la posibilidad de ordenar la toma de estas muestras desde el espacio del usuario. La aplicación que se usó para hacer las mediciones fue `w_escribe_una_LUT`

Esto requirió algunas modificaciones sobre el código del *driver* y de la aplicación `w_escribe_una_LUT`

que se detallan a continuación:

5.1.1 Modificaciones a la aplicación `w_escribe_una_LUT`.

Desde el espacio del usuario se modificó el archivo `w_escribe_una_LUT.c` para agregar los llamados al sistema que imprimen las marcas de tiempo antes y después de cada lectura o escritura de un frame. Debido a que son 4 frames los que se leen y se escriben para poder configurar una LUT, entonces se hicieron en total 16 llamados `ioctl` por cada ejecución de este programa.

La adición del llamado `ioctl` fue necesaria porque en las rutinas contenidas en el *driver* es muy difícil identificar cuando han comenzado y finalizado los subprocesos que se llevan a cabo en la reconfiguración. El siguiente listado muestra la modificación al archivo:

```
// Marca de tiempo
if(ioctl(filedesc, COMANDO_0)<0)
    printf("Hubo un error al hacer ioctl.\n")
```

Cuadro de Código 5.1: Modificaciones sobre el archivo `w_escribe_una_LUT.c`.

Para hacer el llamado al sistema `ioctl` fue necesario también añadir estas líneas en el cabecero del archivo.

```
// Parámetros del ioctl
#include <linux/ioctl.h>

#define PERIFERICO_MAGIC 'W'
#define COMANDO_0 _IO(PERIFERICO_MAGIC,1)
```

Cuadro de Código 5.2: Modificaciones en el cabecero del archivo `w_escribe_una_LUT.c`.

5.1.2 Modificaciones al *driver*

En el *driver* fue necesario declarar las variables que almacenan las marcas de tiempo, junto con alguna información adicional sobre cada una de ellas para facilitar su interpretación.

```
/* Habilita la impresión de marcas de tiempo para medir el desempeño.        */
#define XPS_HWICAP_TIMESTAMPS

/* Variables que almacenan información de las marcas de tiempo               */
#ifdef XPS_HWICAP_TIMESTAMPS
#define NUM_TIMESTAMPS 32*4+2
static char *timestamp_info[] = {"Open", "Close"};
static cycles_t timestamp_cycles[NUM_TIMESTAMPS];
static unsigned int i_timestamps = 0;
#endif
```

Cuadro de Código 5.3: Variables empleadas para medir tiempo en el espacio del *kernel*.

En la función `XPS_HWICAP_open` se tomó la marca de tiempo inicial, y en la función `XPS_HWICAP_release` se tomó la marca de tiempo final. Adicionalmente esta última función imprime en el `log` del sistema los

datos de cada marca de tiempo. Esta instrucción se hizo al final con el objetivo de no afectar la medida de los tiempos. El siguiente código muestra las modificaciones descritas:

```
#ifdef XPS_HWICAP_TIMESTAMPS
    i_timestamps = 0;
    timestamp_cycles[0] = get_cycles();
#endif
```

Cuadro de Código 5.4: Modificación en la función open para tomar la marca de tiempo inicial.

```
#ifdef XPS_HWICAP_TIMESTAMPS
    timestamp_cycles[++i_timestamps] = get_cycles();
    for(indice_ts=0;indice_ts<=i_timestamps;indice_ts++)
        printk(PRINTK_LEVEL "XPS_HWICAP_TIMESTAMPS: \t t_%d = %lu\n",indice_ts,\
        (unsigned long)timestamp_cycles[indice_ts]);
#endif
```

Cuadro de Código 5.5: Modificación en la función release para tomar la marca de tiempo final.

Las marcas intermedias se implementaron como llamados `ioctl`. Su código se muestra a continuación:

```
#ifdef XPS_HWICAP_TIMESTAMPS
            timestamp_cycles[++i_timestamps] = get_cycles();
#endif
```

Cuadro de Código 5.6: Llamado al sistema ioctl para tomar marcas de tiempo intermedias.

5.1.3 Adecuación de los datos extraidos desde el espacio del *kernel*

En el `log` del sistema quedan registrados eventos como los siguientes:

```
Jan 23 14:51:31 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_0 = 3735929456
Jan 23 14:51:31 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_1 = 3736033643
Jan 23 14:51:31 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_2 = 3736172140
Jan 23 14:51:31 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_3 = 3736179181
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_4 = 3736324826
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_5 = 3736325419
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_6 = 3736460049
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_7 = 3736466110
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_8 = 3736649211
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_9 = 3736649706
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_10 = 3736784369
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_11 = 3736790490
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_12 = 3736933423
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_13 = 3736933918
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_14 = 3737068505
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_15 = 3737074627
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_16 = 3737217555
Jan 23 14:51:32 Okinawa-21 kernel: XPS_HWICAP_TIMESTAMPS:        t_17 = 3737223170
```

Cuadro de Código 5.7: Ejemplo de las impresiones que hace el *driver* sobre el registro del sistema.

Los datos numéricos representan el valor del contador de ciclos de reloj del sistema, los cuales suceden cada $2.5\,ns$. El programa de prueba fue modificado para que tome marcas de tiempo cuando inicia el programa, cuando termina el programa y antes de iniciar cada una de las lecturas y escrituras de cada uno de los frames. En total cada ejecución del programa imprime 16 marcas de tiempo mediante el llamado

`ioctl` y dos adicionales al iniciar y al finalizar el programa. El procedimiento de prueba se observa en el script mostrado en el cuadro 5.8. El procedimiento consiste en hacer que las 32 LUTs del periférico reconfigurable lleven a cabo la función OR de 6 entradas. De esta forma se medirá el tiempo que se tarda la reconfiguración del periférico para llevar a cabo esta función lógica.

```
for slice_y in {20..27}
do
   for LUT in {0..3}
   do
      ../../Ejecutables/w_escribe_una_LUT --slice_x=26 --slice_y=${slice_y} \
--tabla_de_verdad=11111111111111111111111111111111111111111111111111111111111110 --LUT=${LUT}
   done
done
```

Cuadro de Código 5.8: Script que hace que el periférico reconfigurable actúe como una operación lógica OR de 6 entradas.

Con la ejecución de este script las marcas de tiempo quedan registradas en el archivo `/var/log/syslog`, del cual pueden ser extraidos mediante el siguiente comando en línea de comandos:

```
Okinawa-22# tail -1000 /var/log/syslog | grep XPS_HWICAP_TIMESTAMPS | \
    grep -v HZ | cut -d_ -f4 | sed -e 's/=/;/' | tail -$((32*18)) > log.csv
```

Con base a estos valores se estimó en excel el tiempo promedio de lectura y de escritura de un frame. En la Tabla 5.1 se observa un resumen de las mediciones tomadas.

Proceso	**Lectura de un frame**	**Escritura de un frame**	**Lectura y Escritura**	**Ejecución de `w_escribe_una_LUT`**
Promedio [ns]	358321,9336	378282,0313	736603,9649	3312615,469
Mínimo [ns]	342265	363080	705345	3218805
Máximo [ns]	463115	466987,5	930102,5	3382350
Núm. de pruebas	128	128	128	32

Tabla 5.1: Resumen de resultados de la medición del tiempo de escritura y lectuar de un frame del FPGA.

5.2 Reconfiguración desde el *cluster*

Para llevar a cabo la reconfiguración parcial de las LUT del FPGA, se empleó inicialmente la herramienta PVM. Se adaptó el ejemplo realizado en el capítulo 2 donde cada nodo reportaba la fecha local al nodo principal y se le añadió al procesador PowerPC®la orden para que reconfigurara los LUTs como compuertas OR como se hizo en el capítulo 4. La fuente del programa que corre el PowerPC®es el siguiente:

```
/* slave program for PVM-first-try */
/* returns string consisting of machine name and local time */

#include <stdio.h>
#include <pvm3.h>
```

```
#include <time.h>
#include <string.h>
//#include "xil_types.h"
//#include "xil_assert.h"

#include "w_escribe_una_LUT.c"

//#define HWICAP_DEVICEID                 0

main()
{
time_t now;
char name[13], buf[60];
int ptid;
int sy,lut;

ptid = pvm_parent();            /* the ID of the master process */
pvm_setopt(PvmRoute, PvmRouteDirect);

gethostname(name, 64);          /* find name of machine */
now=time(NULL);                 /* get time */

strcpy(buf, name);              /* put name into string */
strcat(buf, "'s time is ");
strcat(buf, ctime(&now));       /* add time to string */

pvm_initsend(PvmDataDefault);   /* allocate message buffer */
pvm_pkstr(buf);                 /* pack string into buffer */
pvm_send(ptid, 2);              /* send buffer to master */

for(sy=20;sy<=27;sy++)
  for(lut=0;lut<=3;lut++)
    HwIcapConfiguraLUT(HWICAP_DEVICEID,/*slice_x*/26,sy,lut,\
"11111111111111111111111111111111111111111111111111111111111111110");

pvm_exit;                       /* slave is done and exits */
}
```

Cuadro de Código 5.9: Fuente del programa `helloPVM_slave_ppc.c`.

Para que este código se pueda compilar y ejecutar adecuadamente, es necesario disponer de los archivos de extensión `.h` y las librerías generadas en el cap 4. El listado completo de archivos requeridos se muestra a continuación:

En el directorio `./include`:

- `xbasic_types.h`
- `xhwicap_l.h`
- `xil_io.c`
- `xhwicap.h`
- `xil_assert.h`
- `xil_types.h`
- `xhwicap_family.h`
- `xil_assert.c`
- `xparameters.h`
- `xhwicap_i.h`
- `xil_io.h`
- `xstatus.h`

En el directorio `./lib`:

- `libxhwicap.a`
- `libxil_assert.a`
- `libxil_io.a`

Adicionalmente se modificó el archivo `w_escribe_una_LUT.c` para que actúara únicamente como una función, sin la recepción de datos que se hacía de la línea de comandos. De esta forma el programa

`hello_slave_ppc.c` lo invoca como antes lo hacía el script `set_OR.sh`. La compilación del archivo se debió modificar para que admitiera las librerías y leyera los archivos `.h`. El makefile empleado se muestra a continuación:

```
COMPILADOR := gcc

PVMBINDIR := /compartido/comun/pvm3/bin

FUENTE := helloPVM
SUF_MAESTRO := master
SUF_ESCLAVO := slave

INCLUDE_XHWICAP := -Iinclude/

all: compilar compartir
	./$(FUENTE)_$(SUF_MAESTRO)

compilar: $(FUENTE)_$(SUF_MAESTRO).c $(FUENTE)_$(SUF_ESCLAVO).c \
	  $(FUENTE)_$(SUF_ESCLAVO)_ppc.c
	ssh william@Okinawa-00 "cd `pwd` && $(COMPILADOR) $(FUENTE)_$(SUF_MAESTRO).c \
---------------o $(FUENTE)_$(SUF_MAESTRO) -lpvm3"
	ssh william@Okinawa-00 "cd `pwd` && $(COMPILADOR) $(FUENTE)_$(SUF_ESCLAVO).c \
---------------o $(FUENTE)_$(SUF_ESCLAVO) -lpvm3"
	ssh william@Okinawa-21 "cd `pwd` && $(COMPILADOR) $(FUENTE)_$(SUF_ESCLAVO)_ppc.c lib/* \
---------------o $(FUENTE)_$(SUF_ESCLAVO)_ppc -lpvm3 $(INCLUDE_XHWICAP)"

compartir:
	cp $(FUENTE)_$(SUF_ESCLAVO)      $(PVMBINDIR)/LINUX
	cp $(FUENTE)_$(SUF_ESCLAVO)_ppc  $(PVMBINDIR)/LINUXPPC/$(FUENTE)_$(SUF_ESCLAVO)

clean:
	rm -f $(FUENTE)_$(SUF_MAESTRO) $(FUENTE)_$(SUF_ESCLAVO) $(FUENTE)_$(SUF_ESCLAVO)_ppc
```

Cuadro de Código 5.10: Makefile empleado para compilar la aplicación de prueba de PVM

5.3 Conclusiones

Mediante el presente proyecto se logró identificar los alcances y algunas limitaciones del proceso de reconfiguración parcial en FPGAs a través de la implementación de una plataforma HPRC. Esto establece un precedente en la Universidad Industrial de Santander y en Colombia para trabajos enfocados en esta área. Así mismo, este trabajo permite establecer el panorama de las herramientas existentes y de las diversas aplicaciones beneficiadas, lo cual promueve el inicio de trabajos de investigación que además pueden usar como base la plataforma implementada y/o los procedimientos descritos en este documento. Las conclusiones más importantes obtenidas tras el desarrollo del presente proyecto se muestran a continuación:

- **Se implementó satisfactoriamente Linux Debian Lenny sobre el sistema de desarrollo ML507 apoyado en algunos servicios de red por parte del nodo maestro del *cluster*.**

  Durante el desarrollo del proyecto se definió la arquitectura del *cluster* y en esta organización se establecieron mecanismos para satisfacer los requerimientos del sistema operativo Linux Debian Lenny. De esta forma la tarjeta ML507 pudo iniciar su sistema operativo con el apoyo del servidor NFS y el servidor TFTP que permitieron almacenar el sistema de archivos y el *kernel* respectivamente. La elección de este sistema operativo

fue importante porque facilitó la configuración de herramientas en el *cluster* debido a que redujo un factor de heterogeneidad entre los nodos. Este proceso se encuentra documentado, paso a paso, en el Anexo A como una guía que garantiza la repetibilidad de la implementación.

- **El sistema base consume demasiados recursos y deja sólamente el 60% para la implementación de procesadores específicos.**

  Con la implementación realizada del sistema base, se suplieron los requerimientos del sistema operativo y del *bootloader* para su funcionamiento, pero este diseño requeire aproximadamente el 40% de los slices de la tarjeta. Esto propone un trabajo de investigación posterior que permita llevar a cabo una implementación que requiera menos recursos. Una alternativa más ambiciosa que consiste en implementar una arquitectura donde el sistema base sea externo, y use un ordenador con un procesador de propósito general, mientras la tarjeta ML507 actuaría como un periférico del bus PCI. De esta forma se solucionarían los problemas de interconexión con el *cluster* sin invertir gran cantidad de recursos lógicos en ello.

- **El manejo de un *cluster* heterogéneo reduce la cantidad de herramientas para cómputo paralelo que se pueden implementar pero esta situación no representa una limitante fuerte para trabajar aplicaciones reconfigurables sobre *clusters*.**

  En un principio, durante la ejecución del proyecto, no eran claras las dificultades que iba a generar la heterogeneidad del *cluster* sobre las aplicaciones que se implementen sobre este tipo de plataformas. Mediante el desarrollo del proyecto se logró identificar algunas herramientas que permiten el trabajo sobre este tipo de plataformas y se logró implementar una aplicación que hace uso de reconfiguración parcial sobre el *cluster* haciendo uso de las librerías y el *driver* para el manejo del periférico `XPS_HWICAP`. Esto demuestra que la heterogeneidad del *cluster* no se convierte en una dificultad, siempre que se utilicen las herramientas que dan soporte a este tipo de plataformas, y el acceso a los recursos particulares de cada sistema se puede llevar a cabo sin inconvenientes.

- **Entre todos los bits de configuración del FPGA, se lograron identificar aquellos que modifican el comportamiento de cada una de las LUT6 del sistema.**

  Mediante este proyecto se amplió el conocimiento sobre los bits de configuración del FPGA (esta informacion se obtuvo de la documentación del fabricante), la cual puede

llegar a ser bastante limitada en este tipo de aplicaciones. Durante el desarrollo de este trabajo se identificaron los bits que modifican la tabla de verdad de cada LUT6 del FPGA y se implementaron en C las rutinas que manipulan el comportamiento de los LUT6 según el usuario lo requiera. Es importante resaltar que este trabajo realiza un aporte en la presentación de una metodología que permite identificar los bits que modifican el comportamiento de ciertos recursos del FPGA. En futuros trabajos, esta metodología puede ser utilizada para modificar diferentes recursos lógicos del FPGA.

- **El tiempo de reconfiguración es significativamente alto y debe ser mejorado para que se amplíe la gama de aplicaciones beneficiadas con la reconfiguración parcial.**

  En promedio con la implementación actual, reconfigurar un frame (lectura y escritura) toma $736,6\,\mu s$. Este valor es significativamente alto si se estudia la configuración completa de un periférico tan sencillo como el reconfigurable de prueba o el controlador LCD; los cuales requirieron de (331 Flip-Flops, 279 LUTs) y (272 Flip-Flops, 201 LUTs) respectivamente. En la implementación se reservaron 120 CLBs, lo que representa 108 frames para cada uno. Esto significa que con los métodos implementados en el presente proyecto la reconfiguración completa de cualquiera de estos periféricos tardaría aproximadamente $79,553\,ms$ y solo se están involucrando el $0,512\%$ de los frames del FPGA.

  Este análisis sugiere que el tipo de aplicaciones para las cuales la reconfiguración parcial puede hacer un aporte significativo son aquellas en las cuales el ahorro de tiempo obtenido al implementar un procesador de propósito específico, compensa el tiempo muerto que implica la configuración del mismo en el sistema. Esto normalmente se logra cuando se implementan procesadores con muy alto índice de aceleración o algoritmos que requieren procesar grandes cantidades de datos.

- **Sobre el panorama de las herramientas de Xilinx® para manejar reconfiguración pacial.**

  Las herramientas que Xilinx® ofrece para reconfiguración parcial han venido evolucionando lentamente desde hace varios años atrás. Los FPGAs de Xilinx® han tenido la capacidad para realizar reconfiguración parcial, pero las herramientas de *software* no han ido al mismo ritmo para poder hacer uso de ella. En un principio, las metodologías denominadas *Difference Based*, *Module Based* y *Early Access* fueron empleadas con algunas limitaciones y sin soporte por parte de Xilinx® . Estas herramientas eran libres, con previa solicitud a Xilinx® por medio del XUP, y llegaron a manejar hasta los dis-

positivos Virtex-5 subfamilia LX pero no la subfamilia FX. Posteriormente, en 2009, con la salida al mercado de ISE 11.1(2009) y 12.2(2010), se ha cambiado el enfoque de estas metodologías y se han presentado dos nuevas versiones que facilitan el desarrollo de este tipo de aplicaciones; esta vez, las licencias son comerciales y dan soporte a casi todos los dispositivos de Xilinx® .

Este proyecto ha realizado un aporte hacia la comprensión de la función de los bits de configuración del FPGA `XC5VFX70TFFG1136`, el cual puede interpretarse como un avance hacia el desarrollo de una herramienta para la implementación de sistemas reconfigurables. El aporte de este proyecto está en la comprensión del trabajo que una herramienta de este tipo debe hacer, de su funcionamiento y de los requerimientos de una implementación reconfigurable o de HPRC.

5.4 Trabajo Futuro

Como trabajo futuro se proponen algunas mejoras al sistema y trabajos que complementan y fortalecen el avance que se ha realizado mediante el presente proyecto.

- **Mejoras sobre el *driver* del periférico que hace la reconfiguración**

  El *driver* que maneja el periférico que hace la reconfiguración permite manipular los registros del mismo, de igual forma, la interfaz que ofrece al espacio del usuario está basada en esta forma de acceso. Una mejora que se puede hacer sobre este tipo de sistemas tiene que ver con la percepción del periférico que se tiene en el espacio del usuario. Esta forma de ver el periférico se puede mejorar si se orienta para que pueda descargar más fácilmente un *bitstream* de reconfiguración parcial creado con las herramientas licenciadas de Xilinx® .

  Otra forma de abstracción que se puede tener es sobre los recursos lógicos y no sobre el periférico, de esta forma un usuario sin mayores conociemientos de la reconfiguración parcial o del `HWICAP` puede hacer uso de ella.

- **Mejoras sobre las rutinas que configuran una LUT**

  Reconfigurar una LUT implica la configuración de 4 frames del FPGA. Actualmente el sistema hace un proceso completo para reconfigurar cada frame que incluye la creación de un pequeño cabecero al configurar cada frame. Si se hiciera un solo cabecero para configurar los 4 frames pues mejoraría el desempeño del proceso.

  Otro aspecto que se puede mejorar está relacionado con la posibilidad de optimizar

el proceso de configuración cuando se modifica la tabla de verdad de varios LUTs del mismo slice, debido a que estos comparten los mismos 4 frames.

- **Mejoras sobre el sistema base**

  Durante el desarrollo del proyecto se implementaron los periféricos disponibles para cada uno de los recursos del sistema de desarrollo, debido a que fueron importantes como periféricos de prueba en cada una de las etapas del proceso. Como trabajo futuro se propone crear una versión del sistema base que incluya solo los periféricos necesarios, e incluso puede no incluir la unidad de punto flotante de precisión doble implementada como coprocesador del PowerPC®. Eliminando los siguientes periféricos: SysACE, controlador de la memoria IIC, la unidad de punto flotante, el controlador del LCD y los GPIO que manejan Leds, Switches, etc. se puede reducir los recursos consumidos en un 43% respecto a la implementación realizada en este trabajo. La relación de recursos usados en este trabajo y la espectativa que se tendría de una implementación reducida, se muestra en las Tablas 5.2 y 5.3.

Periférico	**LUTS**	**FF**
SysACE	94	264
Mem. IIC	401	372
APU_FPU	4803	2629
LCD	201	272
GPIOs	297	511
XHWICAP	1799	420
Per. Reconf.	279	331
Otros	5483	5527
Total	13357	10326

Tabla 5.2: Implementación Actual

Periférico	**LUTS**	**FF**
SysACE	–	–
Mem. IIC	–	–
APU_FPU	–	–
LCD	–	–
GPIOs	–	–
XHWICAP	1799	420
Per. Reconf.	279	331
Otros	5483	5527
Total	7561	6278

Tabla 5.3: Espectativa para una implementación reducida.

- **Sobre la versión de Linux que corre en la tarjeta.**

  Aunque la versión de Linux que actualmente tiene instalada la tarjeta le permite tener mayor compatibilidad con los demás nodos del *cluster*, es interesante estudiar la viabilidad de implementar un sistema más ligero únicamente con los componentes necesarios para las aplicaciones de este tipo.

- **Interpretación de otros recursos del FPGA.**

Con la metodología seguida en este proyecto es posible identificar los bits que modifican el comportamiento de otros recursos del FPGA y los bits que modifican la configuración de los recursos de interconexión al interior del mismo. Esto podría finalmente generar una herramienta que generara *bitstreams* parciales, similar a la que Xilinx® ha puesto en el mercado recientemente.

- **Estudiar aplicaciones que tengan una alta espectativa para ser aceleradas mediante un sistema como el implementado.**

  Aprovechando la plataforma implementada, se pueden estudiar los requerimientos de algunas aplicaciones en cuanto a transferencia de datos, recursos lógicos, tipos de datos y de tipos de unidades funcionales para analizar la viabilidad de implementarlos en esta plataforma, ya sea como sistema reconfigurable o como un sistema de procesamiento paralelo con procesadores de propósito específicos implementados en el FPGA.

Bibliografía

[1] Inc. Xilinx. *Development System Reference Guide*. Xilinx, 2008.

[2] Inc. Xilinx. *ML505/ML506/ML507 Evaluation Platform, User Guide*. Xilinx, 2009.

[3] Xilinx Inc. *UG191: Virtex5 FPGA Configuration user guide*, Octubre 2008.

[4] Xilinx Inc. *DS100: Virtex-5 Family Overview*, Diciembre 2008.

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

[6] Tarek El-Ghazawi, Esam El-Araby, Miaoqing Huang, Kris Gaj, Volodymyr Kindratenko, and Duncan Buell. The promise of high-performance reconfigurable computing. *Computer*, 41(2):69–76, 2008.

[7] lam-mpi frequently asked questions. `http://www.lam-mpi.org/faq/`.

[8] Xilinx open source wiki. `http://xilinx.wikidot.com/`.

[9] Xilinx university program - embedded system design flow. `http://www.xilinx.com/university/workshops/embedded-system-design-flow/index.htm`.

[10] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005.

Parte I

Anexos

Apéndice **A**

Guía para la configuración del *embedded system*.

El presente Anexo busca ser una guía para las personas que buscan repetir el procedimiento realizado durante el desarrollo de esta tesis. Este Anexo en particular, mostrará la instalación de un *cluster* de procesadores de propósito general (PPG) Intel x86 y procesadores PowerPC® implementados sobre FPGA. Las secciones que componen este Anexo indicarán detalladamente todo el proceso de implementación desde la instalación del sistema operativo hasta la configuración de las herramientas para cómputo paralelo.

A.1 Descripción del montaje físico del *cluster*

La Figura A.1 muestra el diagrama del *cluster* que se implementó. El *cluster* está compuesto por una serie de nodos conectados por una red de datos *ethernet*, para lo cual es empleó un router D-Link DIR-600 como dispositivo de red. Entre los nodos se destaca el maestro/host que permitirá administrar el *cluster* y servirá como plataforma auxiliar en el desarrollo e implementación de los *embedded system* sobre los FPGA. Además de estos nodos, se emplearon dos equipos que permitieron implementar el *hardware* sobre los FPGAs por medio de las herramientas de Xilinx® : ISE y EDK y así mismo sobre estos equipos se instalaron máquinas virtuales que se añadieron como nodos trabajadores del *cluster*.

Éste mecanismo de las máquinas virtuales permite comprobar la escalabilidad de la implementación. Finalmente se observan las tarjetas ML507, los cuales son nodos basados en FPGA, sobre los cuales es necesario implementar el *hardware* y el sistema operativo.

A.2 Instalación, configuración y personalización del sistema operativo del frontend y los nodos con PPG

Esta sección dará algunos lineamientos generales sobre la instalación del sistema operativo sobre los nodos de PPG. La Figura A.2 referencia los elementos del *cluster* involucrados en esta etapa.

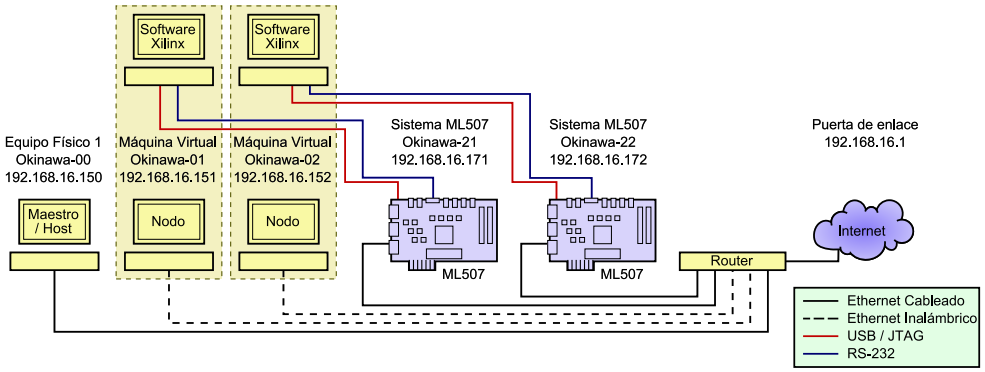


Figura A.1: Diagrama de la implementación física del sistema.

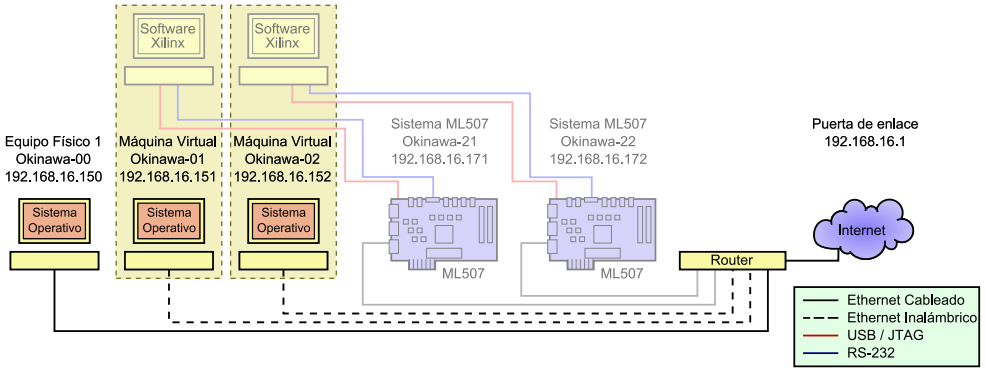


Figura A.2: Sistemas involucrados en esta etapa.

Se decidió instalar Linux en todos los nodos del sistema. Se comenzó mediante la instalación de los paquetes por defecto incluidos en el sistema base, a medida que se avanzó en la instalación se añdieron más y de esa forma se tuvo mejor control sobre los paquetes instalados. La versión instalada fue Lenny y la versión del *kernel* 2.6.26. Dentro de los parámetros asignados en la instalación, se definió el nombre de las máquinas del sistema `Okinawa-00` para el maestro y `Okinawa-XX` para los nodos; también se definió la contraseña de root y se creó un usuario llamado `usuario`. Previo a la configuración de las herramientas del *cluster* y los *embedded system*, es necesario hacer una pequeña personalización y configuración del sistema operativo.

A.2.1 Configuración del servidor DHCP en el router

La instalación del servidor DHCP, se puede realizar en uno de los nodos del *cluster* o en un dispositivo de red, si éste está en capacidad de hacerlo. Para el presente proyecto, se empleó el router que administra la red, el cuál se configuró con los siguientes parámetros:

Parámetros del servidor DHCP del router

```
Subred:                192.168.16.0
Rango de direcciones:  192.168.16.100 - 192.168.16.199
DHCP lease:            1440 minutos
Direcciones fijas:     192.168.16.100   Okinawa-CPS   00:1F:5B:3E:AB:6C
                       192.168.16.101   Sergio-Abreo  00:21:00:24:63:0D
                       192.168.16.150   Okinawa-00    00:16:CB:AE:C3:DD
                       192.168.16.151   Okinawa-01    00:0C:29:7E:BE:FD
                       192.168.16.171   Okinawa-21    00:0A:35:01:D7:4E
                       192.168.16.172   Okinawa-22    00:0A:35:01:D7:1D
```

En caso de emplear el nodo maestro como servidor DHCP, es necesario realizar previamente algunas tareas y su instalación se detalla en la sección A.4.

A.2.2 Configuración de la interfaz de red

Es importante que los nodos tengan acceso a internet, debido a que estos necesitan acceso al repositorio de la distribución para instalar las herramientas. La configuración en linux de las interfaces de red, se basa principalmente en el archivo `/etc/network/interfaces`, en el cual es posible configurar interfaces cableadas o inalámbricas. Es necesario que el nodo maestro de la red tenga una dirección estática, por lo cual se modificó el archivo de configuración así:

/etc/network/interfaces (IP Estática)

```
1
2 # The loopback network interface
3 auto lo
4 iface lo inet loopback
5
6 # Interfaz de red cableada
7 auto eth0
8 iface eth0 inet static
9 address 192.168.16.150
10 network 192.168.16.0
11 netmask 255.255.255.0
12 broadcast 192.168.16.255
13 gateway 192.168.16.1
14 dns-nameservers 192.168.19.2
15 dns-search uis.edu.co
```

Los demás nodos de la red, se configuraron mediante DHCP, para facilitar la clonación de nodos en un futuro. Su archivo de configuración es como se muestra a continuación:

/etc/network/interfaces (IP Dinámica)

```
1
2 # The loopback network interface
3 auto lo
4 iface lo inet loopback
5
6 # Interfaz de red cableada
7 auto eth0
8 iface eth0 inet dhcp
```

Para que los cambios realizados sobre la configuración de la red surjan efecto, es necesario que se reinicie el servicio `networking`, para lo cual se puede ejecutar la siguiente línea de comando.

```
Okinawa-00# /etc/init.d/networking restart
```

```
Okinawa-01# /etc/init.d/networking restart
```

y se puede verificar que se haya cumplido satisfactoriamente al hacer `ifconfig -all` y confirmar que la interfaz de red configurada esté con los parámetros deseados.

A.2.3 Configuración del repositorio en internet

Una vez configurada la interfaz de red, podemos acceder al repositorio para instalar los paquetes que sean necesarios. Se ha seleccionado el repositorio principal de `debian.org` para el sistema. Para indicar al administrador de paquetes esta configuración se debe modificar el archivo `/etc/apt/sources.list` así:

/etc/apt/sources.list

```
1
2 deb http://ftp.debian.org/debian lenny main contrib non-free
```

Para que se actualicen los cambios y se lean los paquetes disponibles, se debe ejecutar

```
Okinawa-00# apt-get update
```

A.2.4 Personalización de vim, bash, etc

Esta sección es opcional, pero puede mejorar el entorno de trabajo para la edición de archivos y acceso a los directorios en el shell. Para mejorar la versión del editor de texto vim que viene por defecto en la instalación de Debian, es posible ejecutar:

```
Okinawa-00# apt-get install vim
```

Para configurar vim para que resalte la sintaxis, se puede modificar el archivo `.vimrc` ubicado en el directorio raiz del usuario, en este caso `root`.

```
Okinawa-00# echo "syntax on" >> /root/.vimrc
```

Finalmente para activar el color al ejecutar `ls`, se puede modificar al archivo `.bashrc` agregando la línea:

/root/.bashrc

```
1
2 alias ls='ls --color=always'
```

A.3 servidor DNS y RDNS

El servidor DNS y RDNS permitirán relacionar el nombre en la red de un equipo con su IP. Es necesario instalar en el nodo maestro el sevicio que puede ser accedido desde la subred del *cluster* como se muestra en la Figura A.3.

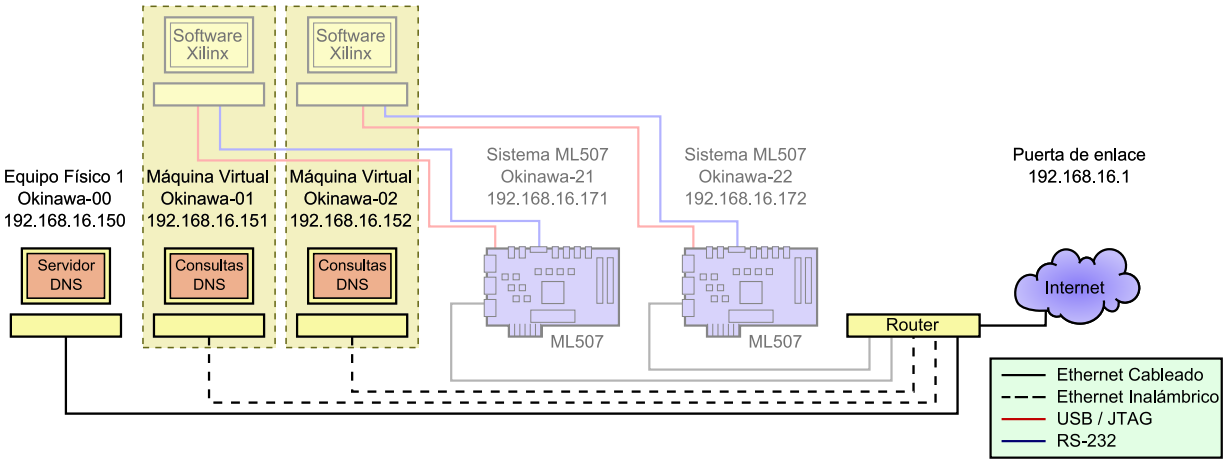


Figura A.3: Elementos involucrados en la instalación del servidor DNS y RDNS.

Inicialmente en el servidor se debe instalar el paquete `bind9`:

```
Okinawa-00# apt-get install bind9
```

se configura el archivo `named.conf.options` para que el servidor sólamente permita consultas dentro de la subred 192.168.16.0

/etc/named.conf.options

```
1
2 auth-nxdomain no;
3 listen-on-v6 { any; };
4
5 version none;
6 allow-query { 192.168.16.0/24; };
7 allow-transfer { none; };
```

Ahora debemos alimentar la base de datos del servidor de nombres de dominio para ello se crean los archivos `db.okinawa.net` y `db.16.168.192` en la carpeta `\etc\bind\`

/etc/bind/db.okinawa.net

```
1
2 $TLL$  24h
3 okinawa.net.  IN  SOA  Okinawa-00.okinawa.net root.okinawa.net (
4   2008042300  ; numero serial
5   3h          ; tiempo de refresco
6   30m         ; tiempo de reintento
7   7d          ; tiempo de "expire"
8   3h          ; negative caching ttl
9 )
10
11 ; Nameservers
12 okinawa.net.  IN  NS  192.168.16.150.
13
14 ; Hosts
15 Okinawa-00.okinawa.net.  IN  A  192.168.16.150
16 Okinawa-01.okinawa.net.  IN  A  192.168.16.151
17 Okinawa-02.okinawa.net.  IN  A  192.168.16.152
```

/etc/bind/named.conf.local

```
1
2 $TTL$  24h
3
4 16.168.192.in-addr.arpa.  IN  SOA  Okinawa-00.okinawa.net root.okinawa.net (
5   2008042300  ; numero serial
6   3h          ; tiempo de refresco
7   30m         ; tiempo de reintento
8   7d          ; tiempo de "expire"
9   3h          ; negative caching ttl
10 )
11
12 ; NameServers
13 16.168.192.in-addr.arpa.  IN  NS  192.168.16.150.
14
15 ; Hosts
16 150.16.168.192.in-addr.arpa.  IN  PTR  Okinawa-00.okinawa.net.
17 151.16.168.192.in-addr.arpa.  IN  PTR  Okinawa-01.okinawa.net.
18 152.16.168.192.in-addr.arpa.  IN  PTR  Okinawa-02.okinawa.net.
```

Posteriormente se referencian estos dos archivos creados dentro del archivo `named.conf.local` para que sean tenidos en cuenta al reiniciar el servicio DNS.

/etc/bind/named.conf.local

```
1
2 zone "okinawa.net" {
3   type master;
4   file "/etc/bind/db.okinawa.net";
5 };
6
7 zone "16.168.192.in-addr.arpa" {
8   type master;
9   file "/etc/bind/db.116.168.192";
10 };
```

En todos los equipos de la red es necesario configurar el archivo `/etc/resolv.conf`.

/etc/resolv.conf

```
1
2 search okinawa.net
3 nameserver 192.168.16.150
```

/etc/resolv.conf

```
1
2 search okinawa.net
3 nameserver 192.168.16.150
```

Ahora es necesario reiniciar el servicio DNS mediante el comando:

```
Okinawa-00# /etc/init.d/bind9 restart
```

Para comprobar el funcionamiento del servidor, podemos consultar el nombre de red de los equipos de la red y sus IP mediante el comando `host`.

```
Okinawa-00# host Okinawa-00
Okinawa-00# host Okinawa-01
Okinawa-00# host 192.168.16.150
Okinawa-00# host 192.168.16.151
```

```
Okinawa-01# host Okinawa-00
Okinawa-01# host Okinawa-01
Okinawa-01# host 192.168.16.150
Okinawa-01# host 192.168.16.151
```

En caso de que no funcione, es recomendable revisar el contenido de los siguientes archivos:

/etc/hosts

```
1
2 127.0.0.1       localhost
3 192.168.16.150  Okinawa-00.okinawa.net  Okinawa-00
4
5 # The following lines are desirable for IPv6 capable hosts
6 ::1     localhost ip6-localhost ip6-loopback
7 fe00::0 ip6-localnet
8 ff00::0 ip6-mcastprefix
9 ff02::1 ip6-allnodes
10 ff02::2 ip6-allrouters
11 ff02::3 ip6-allhosts
```

/etc/hosts.allow

```
1
2 ALL: 192.168.16.0
```

A.4 Configuración el servidor DCHP en el nodo maestro

En caso de contar con un dispositvo de red que no tiene la función de servidor DHCP, esta sección describe la instalación de un servidor en el nodo maestro. Este proceso se inicia mediante la instalación del paquete `dhcp3-server`:

```
Okinawa-00# apt-get install dhcp3-server
```

Posteriormente se editó el archivo `/etc/dhcp3/dhcp.conf` para realizar la configuración del servidor. El siguiente ejemplo muestra la configuración de un servidor, el cual es el encargado principal de asignar las direcciones en el segmento de red 192.168.16.0.

/etc/dhcp3/dhcp.conf

```
1
2 ddns-update-style none;
3
4 option domain-name "okinawa.net";
5 option domain-name-server 192.168.16.190;
6
7 default-lease-time 14400;
8 max-lease-time 14400;
9
10 authoritative;
11
12 log-facility local7;
13
14 subnet 192.168.16.0 netmask 255.255.255.0 {
15   option domain-name "okinawa.net";
16   deny unknown-clients;
17 }
18
19 group {
20   option routers 192.168.16.1;
21   next-server 192.168.16.1;
22
23   host Okinawa-00.okinawa.net {
24     hardware ethernet 00:16:CB:AE:C3:DD;
25     fixed-address 192.168.16.150;
26     option host-name "Okinawa-00";
27   }
28
29   host Okinawa-01.okinawa.net {
30     hardware ethernet 00:0C:29:7E:BE:FD;
31     fixed-address 192.168.16.151;
32     option host-name "Okinawa-01";
33   }
34
35   host Okinawa-21.okinawa.net {
36     hardware ethernet 00:0A:35:01:D7:4E;
37     fixed-address 192.168.16.171;
38     option host-name "Okinawa-21";
39   }
40
41   host Okinawa-22.okinawa.net {
42     hardware ethernet 00:0A:35:01:D7:1D;
43     fixed-address 192.168.16.172;
44     option host-name "Okinawa-22";
45   }
46 }
```

Una vez modificado el archivo de configuración, se puede reiniciar el demonio DHCP mediante el comando:

```
Okinawa-00# /etc/init.d/dhcp3-server restart
```

y se puede comprobar desde el nodo esclavo ejecutando:

```
Okinawa-01# dhclient
```

A.5 Instalación y configuración del servidor y los clientes NFS.

El primer paso fue instalar en el maestro y en el nodo los paquetes correspondientes:

```
Okinawa-00# apt-get install nfs-common nfs-kernel-server
```

```
Okinawa-01# apt-get install nfs-common
```

Luego se creó el directorio que se compartió y se le modificaron sus permisos.

```
Okinawa-00# mkdir -p /compartido/x86
Okinawa-00# mkdir -p /compartido/powerpc
Okinawa-00# chmod 777 /compartido/x86
Okinawa-00# chmod 777 /compartido/powerpc
```

Se configuró el servidor NFS editando el archivo de configuración `/etc/exports`

/etc/exports

```
1
2 /compartido/x86 192.168.16.0/255.255.255.0(sync,no_wdelay,subtree_check,rw)
3 /compartido/powerpc 192.168.16.0/255.255.255.0(sync,no_wdelay,subtree_check,rw)
```

Se reinició el servicio del servidor NFS.

```
Okinawa-00# /etc/init.d/nfs-kernel-server restart
```

Se puede verificar su funcionamiento consultando los directorios que se están exportando mediante el comando

```
Okinawa-00# exportfs
```

En el cliente, se creó la carpeta donde montó el directorio compartido y se modificó el archivo `fstab` donde se configuraron algunas propiedades de este directorio.

```
Okinawa-01# mkdir -p /compartido/x86
```

/etc/fstab

```
1
2 192.168.16.150:/compartido/x86  /compartido/x86  nfs  defaults  0  0
```

finalmente se reinició el servicio del cliente NFS

```
Okinawa-01# /etc/init.d/nfs-common restart
```

También es posible solicitar a linux que monte las unidades indicadas en el archivo `/etc/fstab` mediante el comando

```
Okinawa-01# mount -a
```

se puede verificar los sistemas de archivos montados mediante el comando

```
Okinawa-01# mount
```

A.6 Configuración de SSH y los usuarios

La configuración de ssh es necesaria para la ejecución de herramientas en el *cluster*, además facilita el entorno de trabajo al permitir el acceso remoto a los nodos sin necesidad ubicarse físicamente cerca de él. Las herramientas del *cluster*, exigen que los usuarios del *cluster* puedan iniciar sesiones remotas en los nodos del sistema sin la necesidad de *password*. Para iniciar el proceso es necesario instalar ssh en el maestro y en el nodo.

```
Okinawa-00# apt-get install ssh
```

```
Okinawa-01# apt-get install ssh
```

con esto se debe tener la posibilidad de ejecutar comandos como `ssh` o `ssh-keygen`. Ahora proseguiremos con la configuración de los usuarios, para lo cual modificamos el archivo `/etc/adduser.conf` en el maestro y en el esclavo para modificar los parámetros de los nuevos usuarios en el sistema. Específicamente se modificarán las siguientes líneas para modificar la ubicación del directorio raiz e impedir la creación de un grupo por cada usuario:

/etc/adduser.conf

```
1
2 DHOME=/compartido/home
3 USERGROUPS=no
```

/etc/adduser.conf

```
1
2 DHOME=/compartido/home
3 USERGROUPS=no
```

Ahora creamos un usuario en el host y en el esclavo, e ingresamos las opciones por defecto.

```
Okinawa-00# adduser william
```

```
Okinawa-01# adduser william
```

El siguiente paso busca crear las llaves públicas y privadas del usuario, para lo cual debemos ingresar al nodo maestro como el nuevo usuario y se ejecuta:

```
Okinawa-00# su william
william@Okinawa-00# cd
william@Okinawa-00# ssh-keygen
```

Se deben dejar las opciones por defecto y no asignar ningún *passphrase*. Este comando debe crear los archivos `id_rsa.pub` y `id_rsa` dentro del directorio `/compartido/comun/home/william/.ssh/`. El contenido de la llave pública debe ser insertado en el archivo `authorized_keys2` para que no se solicite la clave al mismo usuario.

```
william@Okinawa-00# cd .ssh
william@Okinawa-00# cat id_rsa.pub >> authorized_keys2
```

se le ajustan los permisos.

```
william@Okinawa-00# chmod 600 authorized.keys2
```

Se puede comprobar que el proceso ha funcionado si al iniciar una sesión sobre el nodo esclavo con el mismo usuario, este no solicita la clave.

```
william@Okinawa-00# ssh william@Okinawa-01
```

```
william@Okinawa-01# ssh william@Okinawa-00
```

A.7 servidor TFTP

El servidor TFTP permitirá descargar el *kernel* y los archivos de configuración durante el arranque de los FPGA. El servidor TFTP se instaló en el nodo maestro, mediante el siguiente paquete:

```
Okinawa-00# apt-get install tftpd-hpa
```

Ahora creamos el directorio que se compartirá a los demás nodos del sistema y se le dan permisos de solo lectura. El directorio `Linuxboot` es un subdirectorio del servidor donde se almacenarán el *kernel* y la configuración de cada nodo basado en FPGA.

```
Okinawa-00# mkdir -p /compartido/tftpboot/Linuxboot
Okinawa-00# chmod 755 /compartido/tftpboot
Okinawa-00# chmod 755 /compartido/tftpboot/Linuxboot
```

El archivo de configuración `/etc/default/tftpd-hpa` permite indicar el directorio que se va a compartir y el modo en el que el servidor se iniciará. Normalmente este servicio es iniciado por el servicio `inet`, pero para nuestro caso se iniciará como un servicio independiente, por lo cual se debe habilitar dicha opción en el archivo de configuración.

/etc/default/tftpd-hpa

```
1
2 RUN_DAEMON="yes"
3 OPTIONS="-l -s /compartido/tftpboot"
```

Igualmente es necesario deshabilitar el servicio TFTP en el archivo de configuración `inet`, para lo cual se edita el archivo `/etc/inetd.conf` comentando la línea relacionada con este servicio. Posteriormente se reinició el servicio del servidor TFTP.

```
Okinawa-00# /etc/init.d/tftpd-hpa restart
```

La verificación se puede realizar mediante la revisión de los procesos activos y de los puertos de escucha que abre el servicio TFTP. Para ello se pueden ejecutar los siguientes comandos:

```
Okinawa-00#   ps aux | grep tftpd
Okinawa-00#   netstat -plun | grep tftp
```

Otra forma de verificación es mediante la transferencia de un archivo de prueba a un nodo trabajador. El siguiente comando permite crear el archivo de prueba que se va a transmitir.

```
Okinawa-00# echo "Hola prueba" >> /compartido/tftpboot/Linuxboot/testfile
```

En el cliente es necesario instalar el cliente TFTP, para lo cual se ejecuta

```
Okinawa-01#   apt-get install tftp-hpa
```

Ahora el cliente debe solicitar el archivo de prueba y verificar su existencia mediante los siguientes comandos:

```
Okinawa-01#  tftp Okinawa-00 -c get Linuxboot/testfile
Okinawa-01#  cat ./testfile
```

(Introducción para la segunda guía de usuario)

Durante esta sección se instalará el sistema operativo de los *embedded system* sobre los sistemas ML507, para lo cual es necesario configurar previamente un conjunto de herramientas para el desarrollo del mismo. La Figura A.4 presenta los elementos de la implementación física involucrados durante esta sección.

A.8 Herramientas para compilación cruzada

Dentro de las herramientas necesarias para la instalación se encuentran los compiladores cruzados. Se ha seleccionado los que están contenidos en la herramienta ELDK, los cuales se pueden descargar como una imagen iso desde la página denx. Esta imagen se grabó en un DVD y se montó en el sistema Linux del host con permisos de ejecución mediante el siguiente comando:

```
Okinawa-00# mount -o exec /dev/cdrom /media/cdrom
```

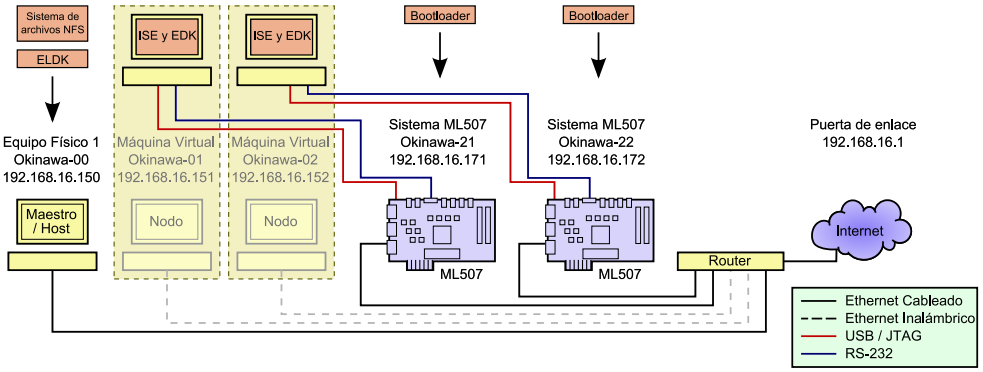


Figura A.4: Elementos involucrados en esta sección.

Ahora es posible ejecutar el script de instalación `./install` que se encuentra en el directorio raiz del DVD . Es necesario tener instalado `gcc` y los permisos de ejecución para efectuar esta tarea correctamente. Se destinó el directorio /opt/ELDK/4.2 como la raiz de la instalación del ELDK y se ha seleccionado la arquitectura PowerPC®4xx con unidad de punto flotante.

```
Okinawa-00# cd /media/cdrom
Okinawa-00# ./install -d /opt/ELDK/4.2 ppc_4xxFP
```

El siguiente paso fue ajustar las variables de entorno para los compiladores cruzados mediante el script `eldk_init` para la arquitectura ppc_4xxFP. Esto se debe hacer al inicio de cada sesión donde se empleen estos compiladores.

```
Okinawa-00# cd /media/cdrom
Okinawa-00# source eldk_init ppc_4xxFP
```

Adicionalmente los scripts `ELDK_FIXOWNER` y `ELDK_MAKEDEV` permiten ajustar los permisos y los propietarios de los archivos de desarrollo. Para ello ejecutamos las siguientes líneas:

```
Okinawa-00# /media/cdrom/ELDK_FIXOWNER  -a ppc_4xxFP
Okinawa-00# /media/cdrom/ELDK_MAKEDEV -a ppc_4xxFP
```

En los directorios `/opt/ELDK/4.2/ppc_4xxFP` y `/opt/ELDK/4.2/usr/bin` se pueden verificar los archivos que la herramienta ha instalado. Allí deben aparecer un sistema de archivos para el *embedded system* y los archivos ejecutables del compilador cruzado, respectivamente.

A.9 Instalación del sistema base en la tarjeta

Durante esta etapa se implementa el sistema de cómputo base sobre el FPGA como lo indica la Figura A.5.

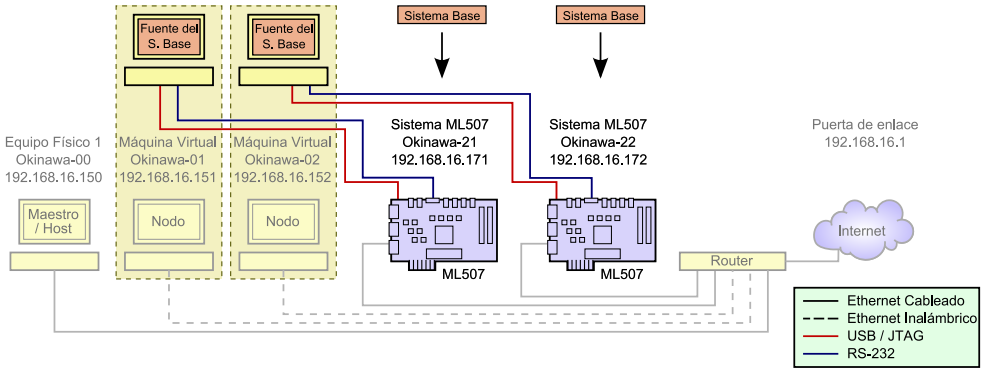


Figura A.5: Elementos involucrados en la implementación del sistema base.

El sistema base corresponde al *hardware* básico del *embedded system* que contiene el bus sobre el cual está el procesador, los periféricos y las memorias. El sistema base que se empleó se puede descargar desde el sitio del fabricante en la dirección: https://secure.xilinx.com/webreg/clickthrough.do?cid=111799. Este es un archivo comprimido que contiene un proyecto en EDK con el mapa de memoria mostrado en la tabla A.1:

Elemento	**Dirección base**	**Tamaño Rango**
Procesador PowerPC®440		
Bloque RAM	`0xFFFF0000`	64K
GPIO Leds	`0x81400000`	64K
GPIO Positions	`0x81420000`	64K
GPIO PushButtons	`0x81440000`	64K
GPIO DipSwitch	`0x81460000`	64K
IIC_EEPROM	`0x81600000`	64K
Controlador de Interrupciones	`0x81800000`	64K
MAC HW Ethernet	`0x81C00000`	64K
SysACE_CompactFlash	`0x83600000`	64K
Temporizador Watchdog	`0x83A00000`	64K
Temporizador	`0x83C00000`	64K
Puerto Serie RS-232	`0x83E00000`	64K
RAM DDR2	`0x00000000`	256MB
Memoria Flash	`0xFC000000`	32MB

Tabla A.1: Mapa de memoria del sistema base

Inicialmente solo se requiere descargar el proyecto al FPGA con las opciones por defecto, por lo tanto solo es necesario abrirlo y descargar el *bitstream* y comprobar su funcionamiento.

A.10 Instalación del Bootloader

Durante esta sección se descargará la fuente del `u-boot`, se compilará y se instalará en la memoria Flash del sistema de desarrollo ML507. La Figura A.6 muestra de forma global el proceso.

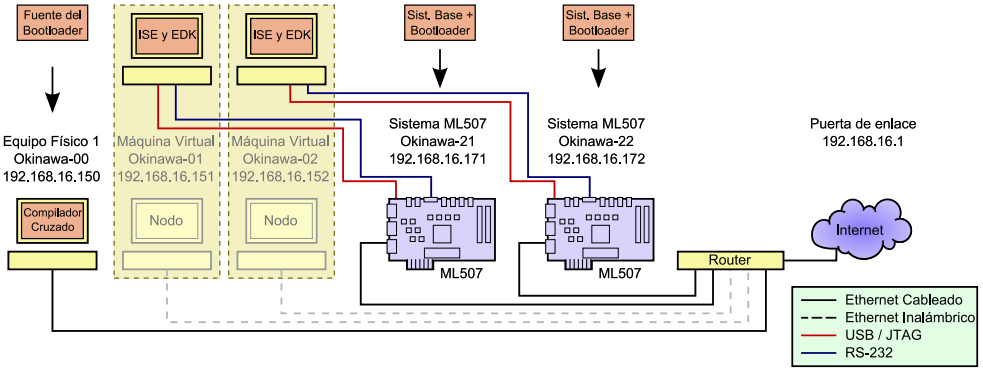


Figura A.6: Elementos involucrados en la instalación del bootloader.

Inicialmente, crearemos el directorio donde se almacenarán las fuentes del `u-boot` y del *kernel*.

```
Okinawa-00# mkdir -p /opt/SoftDev/Xilinx
Okinawa-00# cd /opt/SoftDev/Xilinx
```

Ahora se debe descargar la fuente, para lo cual se proponen tres formas: mediante el comando git que sincroniza repositorios en internet, directamente con la url de la fuente o directamente en el navegador. La fuente seleccionada es la que el grupo de trabajo de Xilinx® ha adaptado para este sistema.

Obtención de la fuente mediante git (Recomendado)

```
Okinawa-00# git clone git://git.xilinx.com/u-boot-xlnx.git
```

Obtención de la fuente mediante wget

```
Okinawa-00# wget http://git.xilinx.com/cgi-bin/gitweb.cgi?p=u-boot-xlnx
```

Obtención de la fuente mediante el navegador

ingresar a la página *http://git.xilinx.com/cgi-bin/gitweb.cgi* y luego seleccionar el *tree* del proyecto */u-boot-xlnx.git*, donde se encuentra un enlace con el texto *snapshot* el cual descarga la fuente en un archivo comprimido *u-boot-xlnx.git-HEAD.tar.gz*

En caso de usar el modo gráfico, es necesario descomprimir en la carpeta `/opt/SoftDev/Xilinx` el archivo `u-boot-xlnx.git-HEAD.tar.gz`. Para ello se puede usar el siguiente comando que permite mantener los permisos de los archivos:

```
Okinawa-00# tar xvfzp u-boot-xlnx.git-HEAD.tar.gz
```

Una vez obtenida la fuente, se adaptará la misma a las especificaciones de la tarjeta. Hay que tener en cuenta que el bootloader está hecho teniendo en cuenta el mapa de memoria del diseño de referencia que se encuentra en el sitio de Xilinx® como se explicó en la sección A.9; por lo tanto se debe confimar que el archivo `/opt/SoftDev/Xilinx/board/xilinx/ml507/xparameters.h` de la fuente corresponda con el archivo `xparameters.h` del proyecto de EDK trabajado en la sección anterior.

Adicionalmente es necesario realizar algunos cambios sobre la fuente en el archivo que permite configurar parámetros del `u-boot` como los comandos disponibles y las variables de entorno. Este archivo se encuentra en `/opt/SoftDev/Xilinx/u-boot-xlnx.git/include/configs/ml507.h` y las modificaciones fueron las siguientes:

/opt/SoftDev/Xilinx/u-boot-xlnx.git/include/configs/ml507.h

```
1
2  // Opciones para habilitar la depuración de las acciones de u-boot
3  #define DEBUG
4  #define ET_DEBUG 1
5
6  // Temporizador y comando de arranque
7  #define CONFIG_BOOTDELAY 2
8  #define CONFIG_BOOTCOMMAND "run bootnfs"
9
10 // Comandos ejecutados antes del conteo regresivo
11 #define CONFIG_PREBOOT "echo ;" \
12 "echo Bootloader configurado por W;" \
13 "echo -------------------------;" \
14 "echo ;" \
15 "echo Arranque por defecto: $(bootcmd);" \
16 "echo"
17
18 // Opciones de la línea de comandos
19 #define CONFIG_AUTO_COMPLETE
20 #define  CONFIG_CMDLINE_EDITING
21
22 // Definición de las variables de entorno
23 #define CONFIG_EXTRA_ENV_SETTINGS \
24 "serverip=192.168.16.150\0" \
25 "ipaddr=192.168.16.172\0" \
26 "bootnfs=run loadkernel;run loaddtb;setenv ramimg_addr -;run runkernel\0" \
27 "boottftp=run loadkernel;run loadramimg;run loaddtb;run runkernel\0" \
28     "loadkernel=tftp $(kernel_addr) $(bootfile)\0" \
29         "kernel_addr=0x1c00000\0" \
30         "bootfile=Linuxboot/uImage\0" \
31     "loadramimg=tftp $(ramimg_addr) $(ramfile)\0" \
32         "ramimg_addr=0x1800000\0" \
33         "ramfile=Linuxboot/ramimg.gz\0" \
34     "loaddtb=tftp $(dtb_addr) $(dtbfile)\0" \
35         "dtb_addr=0x1000000\0" \
36         "dtbfile=Linuxboot/ml507.dtb\0" \
37     "runkernel=bootm $(kernel_addr) $(ramimg_addr) $(dtb_addr)\0"
```

Una vez está lista la fuente, debemos asegurarnos de que las variables de entorno relacionadas con el compilador cruzado estén correctamente. Para ello podemos ejecutar el comando:

```
Okinawa-00# /opt/ELDK/4.2/eldk_init 4xx_FP
```

Ahora mediante el comando make compilamos el *kernel* en dos etapas:

```
Okinawa-00# make ml507_config
Okinawa-00# make
```

El primer comando se encarga de generar un archivo de configuración con las opciones de compilación que son tomadas por el segundo comando, el cual genera los archivos `u-boot`, `u-boot.srec` y `u-boot.bin`. El archivo `u-boot.srec` debe ser llevado al host que contiene EDK.

Una vez se tiene el archivo srec del `u-boot`, este debe ser llevado a la memoria flash. Para ello se debe abrir el proyecto de referencia de EDK utilizado en la sección A.9. Es importante que el sistema base sea implementado previamente mediante la opción *Device Configuration* > *Download Bitstream* mostrada en la Figura A.7.

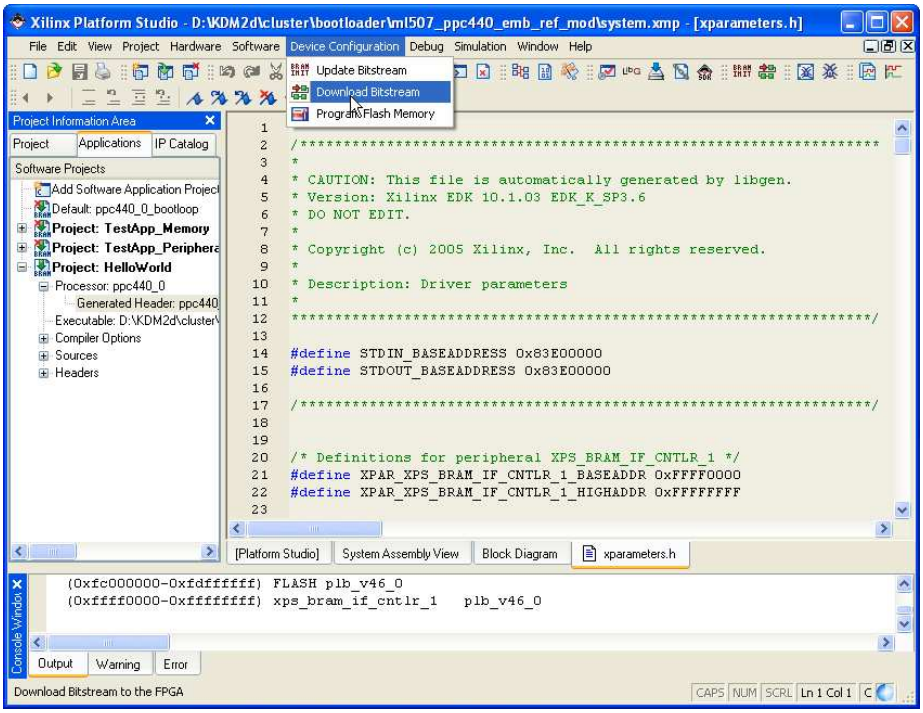


Figura A.7: Función de EDK para descargar el *bitstream* del diseño de referencia.

Ahora se debe descargar a la memoria Flash el *bootloader* mediante la opcion *Device Configuration* > *Program Flash Memory* como se muestra en la Figura A.8, donde se aprovecha el sistema base descargado en el FPGA para escribir sobre la memoria Flash el contenido del archivo *u-boot.srec*.

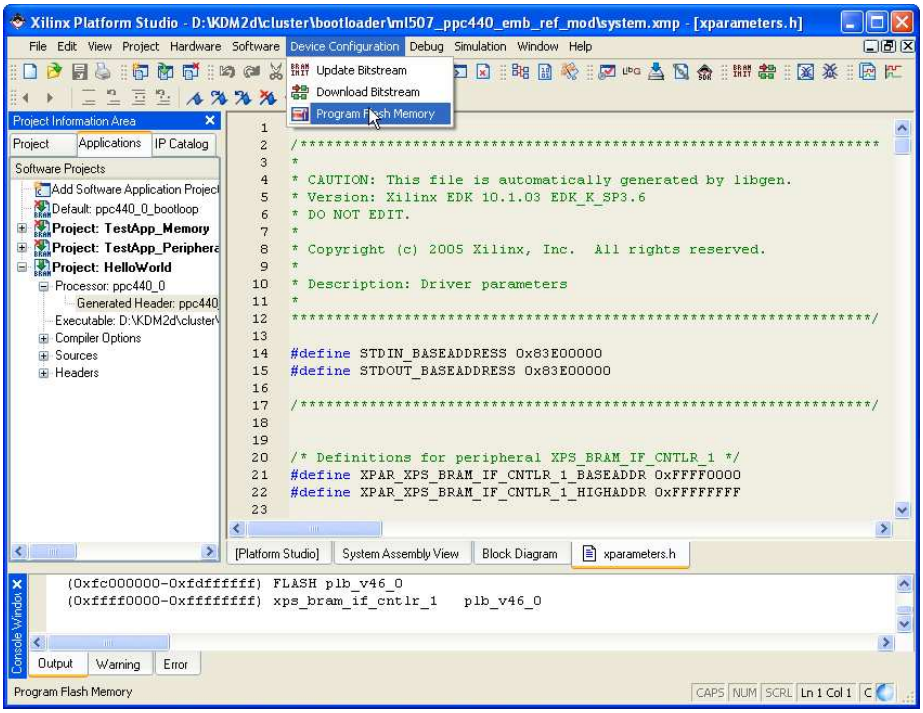


Figura A.8: Función de EDK para descargar un archivo a la memoria Flash.

Adicionalmente EDK crea una aplicación que copia el bootloader desde la flash a la RAM y lo ejecuta desde allí. Los ajustes de los parámetros realizados se muestran en la Figura A.9

El paso anterior debe haber creado un proyecto en la pestaña de *software*, el cual se debe editar para que se pueda desplegar la información por el puerto serie, para ello el archivo creado por ISE llamado `bootloader.c` se edita agregando las siguientes librerías:

bootloader.c

```
40
41 #include "xparameters.h"
42 #include "xuartns550_l.h"
```

Al iniciar el main, es necesario configurar el puerto serie RS232 mediante las siguientes instrucciones:

bootloader.c

```
95
96 XUartNs550_SetBaud(XPAR_RS232_UART_1_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, 9600);
97 XUartNs550_mSetLineControlReg(XPAR_RS232_UART_1_BASEADDR, XUN_LCR_8_DATA_BITS);
```

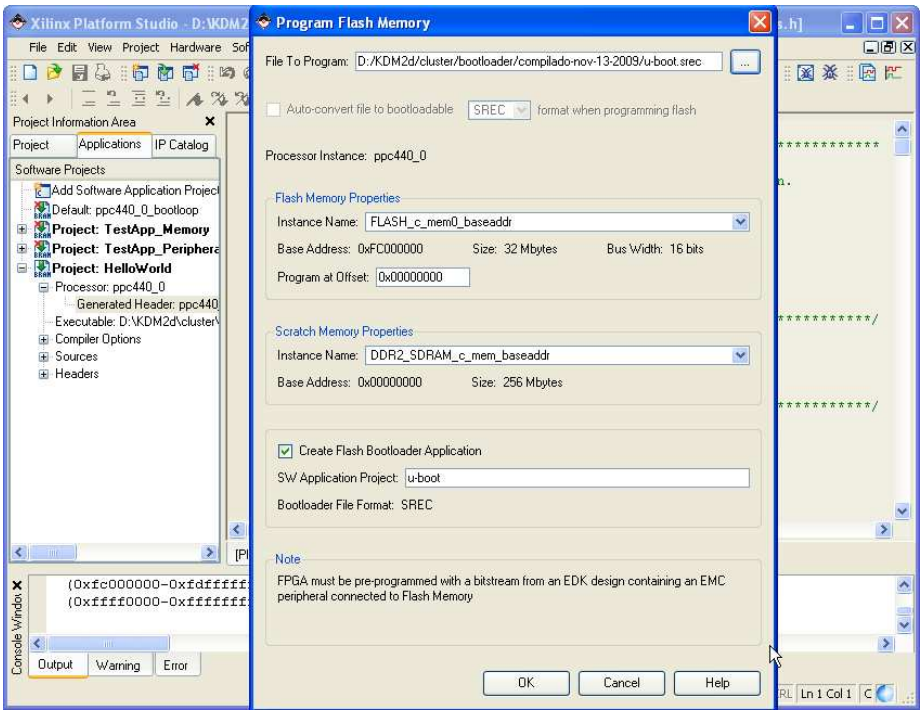


Figura A.9: Cuadro de diálogo con la configuración para descargar `u-boot` a la Flash.

Si se desea un bootloader más ágil, es posible comentar la siguiente línea que imprime algunos comentarios para la depuración de la aplicación

bootloader.c

```
48
49 #define VERBOSE
```

Para compilar las fuentes en C y descargar el sistema base, se puede ejecutar nuevamente la opción *Device Configuration* > *Download Bitstream*, con lo cual el sistema de desarrollo debería iniciar la aplicación *bootloader* que permite el inicio del `u-boot`. Antes de continuar con la compilación del *kernel* es aconsejable que el archivo de configuración generado en este proceso quede guardado en la memoria PROM de configuración del sistema de desarrollo para no tener que hacer uso de EDK cada vez que se inicie la tarjeta. Para ello se pueden seguir las instrucciones del foro de Xilinx® . Aunque las instrucciones sugieren cambiar una de las opciones con las que se genera el archivo de configuración, esto no fue necesario durante la implementación. La ejecución de este procedimiento se muestra a continuación y se inicia con la generación del archivo para la memoria PROM. Para ello se inicia la aplicación `iMPACT` y se selecciona la opción *PROM File Formater* como se muestra en la Figura A.10.

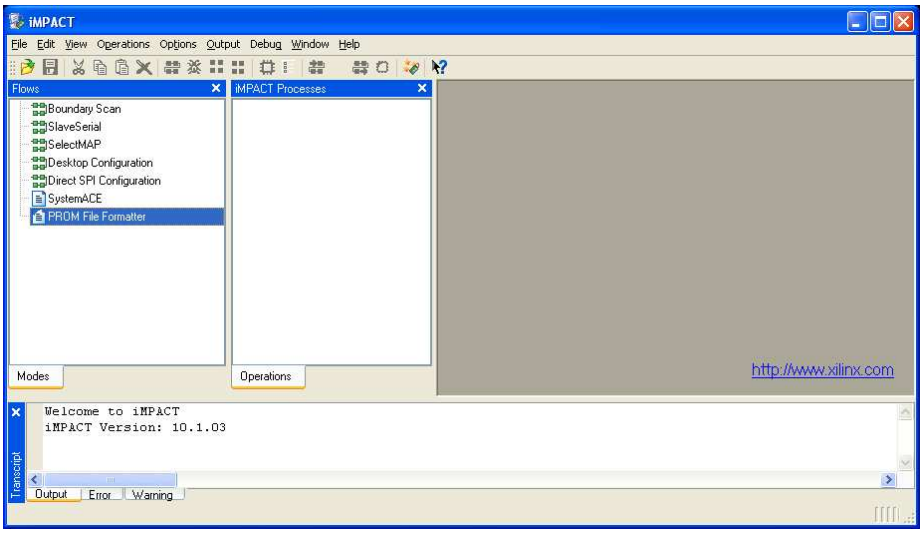


Figura A.10: Aplicación iMPACT usada para generar el archivo de configuración de la PROM y programarla.

Posteriormente se sigue el asistente para la creación del archivo de configuración de la memoria PROM. El primer paso se muestra en la Figura A.11, donde se asigna el nombre del archivo y se indica el tipo de memoria que se va a programar.

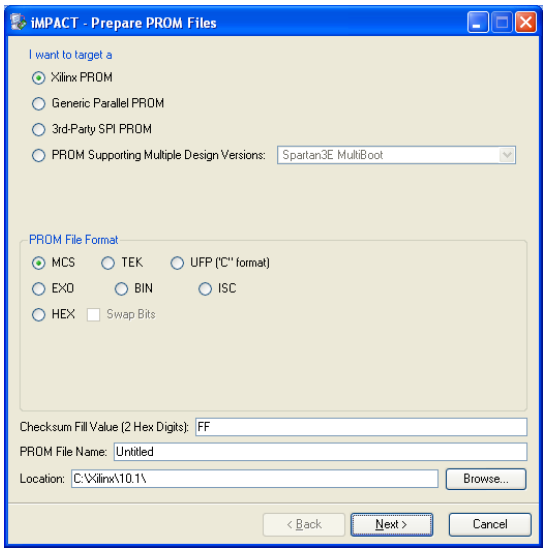


Figura A.11: Diálogo donde se especifican los parámetros generales del archivo para la memoria PROM que se desea generar.

En el siguiente paso del asistente, se indica la forma en la que está interconectada las memorias PROM.

Para el sistema de desarrollo se debe dejar como lo muestra la Figura A.12. Se ha seleccionado en modo paralelo debido a que es más rápido que el modo serial y esto se ve reflejado en el tiempo de arranque del sistema.

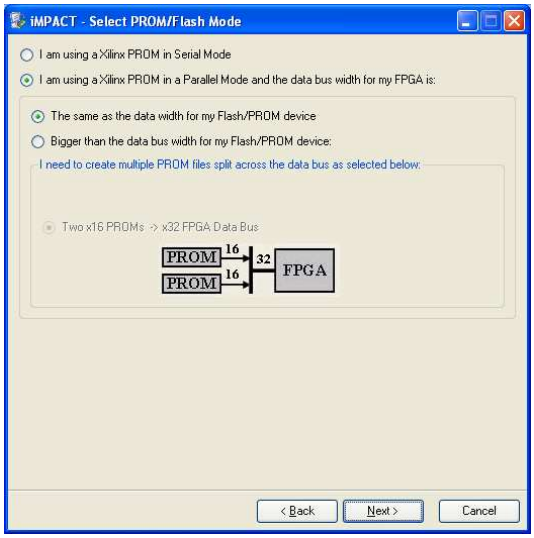


Figura A.12: Diálogo donde se especifica si el archivo se va a cargar de forma serial o paralela.

La Figura A.13 nos muestra el diálogo donde se especifican las referencias de las memorias instaladas en el sistema ML507. Y la Figura A.14 nos muestra un resumen de la información ingresada

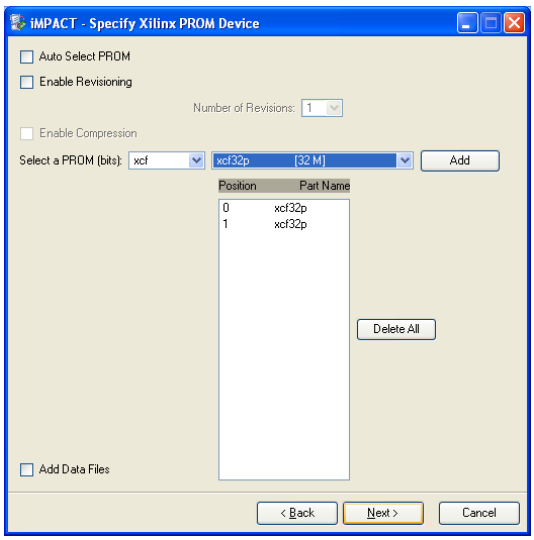


Figura A.13: Diálogo donde se definen el tipo y la cantidad de memorias PROM disponibles.

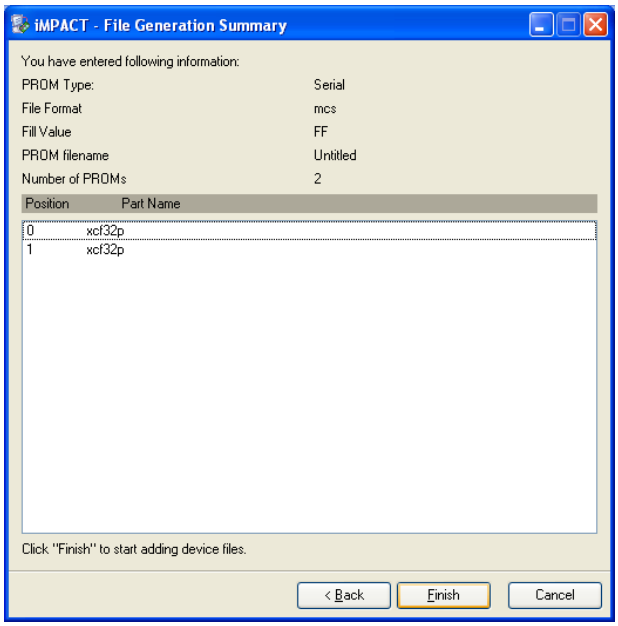


Figura A.14: Diálogo donde confirman los parámetros del archivo que se va a generar.

Al hacer click sobre *Finish*, aparece un cuadro de diálogo donde se debe seleccionar el archivo de EDK `implementation/download.bit`. Se pueden ignorar algunos warnings que pueden aparecer indicando que solo se ha empleado una memoria PROM. Finalizada esta etapa, iMPACT muestra un diagrama con las memorias PROM y el FPGA que se va a configurar como se muestra en la Figura A.15.

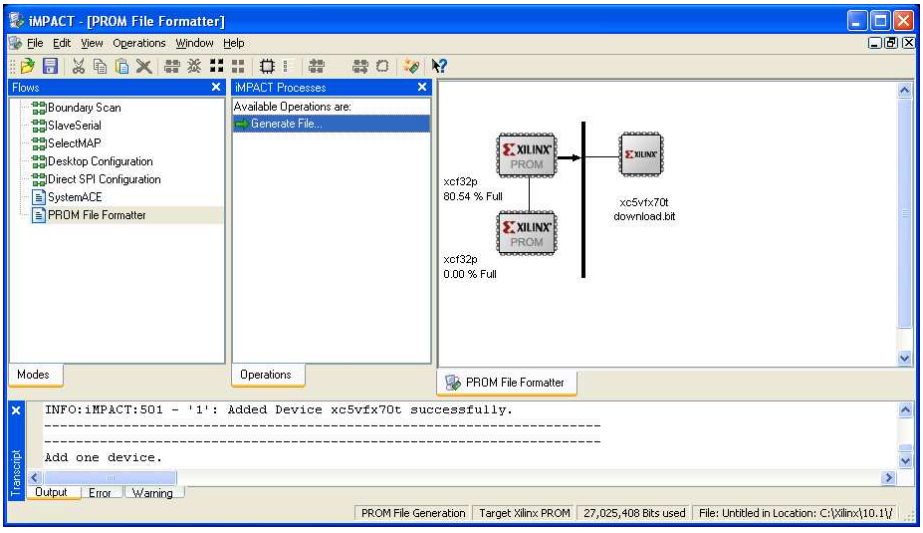


Figura A.15: En la ventana se observan las memorias PROM que se van a usar y los archivos que se cargarán en ellas.

Allí se hace doble click sobre la opción `Generate File...`, lo cual iniciará el proceso de generación del archivo .mcs; y si el proceso termina correctamente deberá enviar un mensaje como el de la Figura A.16.

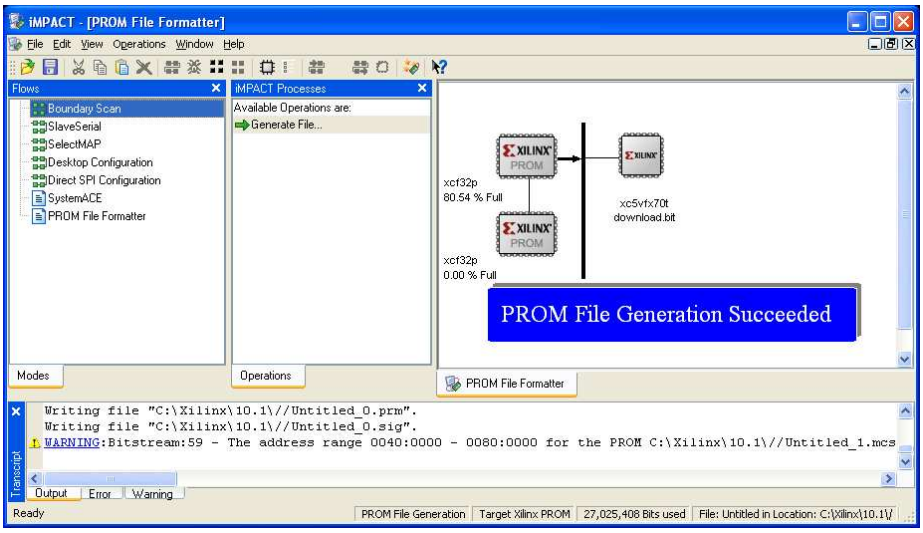


Figura A.16: Aplicación iMPACT usada para generar el archivo de configuración de la PROM y programarla.

Luego se abre la opción de `Boundary Scan` en el cual aparecen los dispositivos que están dentro de la cadena JTAG como se muestra en la Figura A.17.

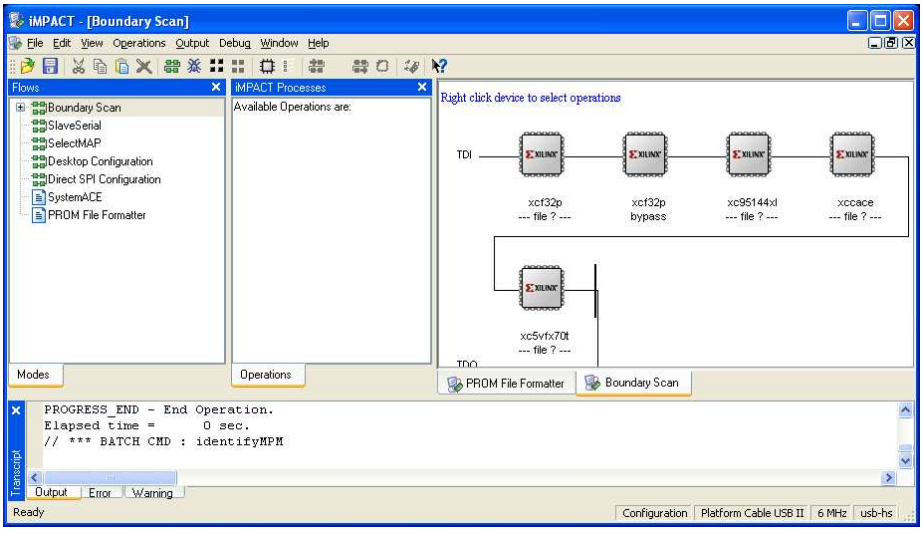


Figura A.17: Aplicación iMPACT usada para generar el archivo de configuración de la PROM y programarla.

Finalmente se asignan los archivos de configuración mcs y se configuran, asegurando que la opción `Parallel Mode` esté activa como se muestra en la Figura A.18. Adicionalmente se debe tener en cuenta que los pines de configuración del FPGA deben estar en: $M[2:0] = 100$

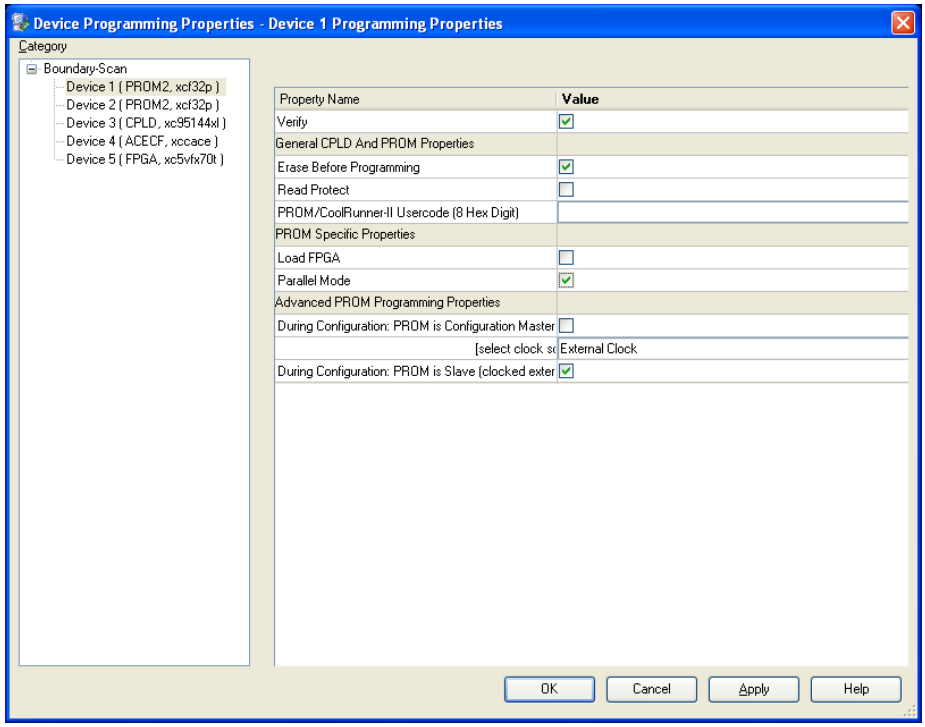


Figura A.18: Aplicación iMPACT usada para generar el archivo de configuración de la PROM y programarla.

A.11 Compilar el *kernel*

Durante esta sección se compilará el *kernel*, el cual quedará disponible en el servidor TFTP para que el bootloader pueda cargarlo. Al finalizar esta sección se podrá iniciar Embedded Linux gracias al sistema de archivos generado por el ELDK. La Figura A.19 representa las etapas de este proceso:

Para iniciar el proceso se debe obtener la fuente del *kernel*, de forma similar al `u-boot`. En esta ocasión se empleó la interfaz gráfica obteniendo el archivo `linux-2.6-xlnx.git-HEAD.tar.gz`. Posteriormente se descomprimió la fuente en el mismo directorio de trabajo de `u-boot` mediante el comando:

```
Okinawa-00# tar xvfzp linux-2.6-xlnx.git-HEAD.tar.gz
```

Ahora se compiló la fuente con las opciones por defecto para la virtex5 mediante los siguientes comandos:

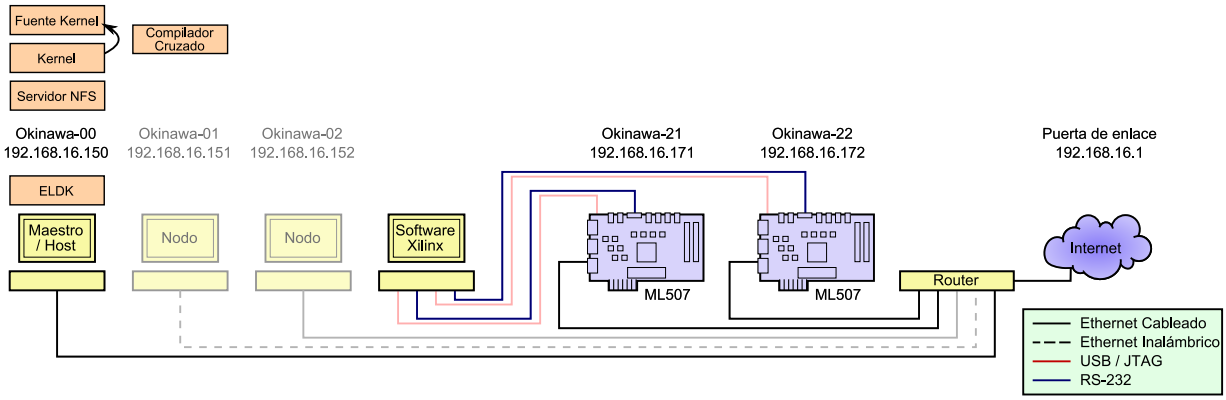


Figura A.19: Elementos involucrados en la compilación del *kernel*

```
Okinawa-00# ARCH=powerpc
Okinawa-00# make ARCH=powerpc 44x/virtex5_defconfig
Okinawa-00# make ARCH=powerpc uImage
```

Esto debe generar el archivo `uImage`, el cual es una imagen del *kernel* con un cabecero especial que emplea `u-boot`. El siguiente comando permite conocer el tipo de archivo que se generó:

```
Okinawa-00# file /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/uImage
```

El siguiente paso consiste en generar el archivo DTB modificando su fuente que es de extensión DTS, el cual se compila mediante el compilador llamado DTC. El archivo DTB permite modificar algunas opciones del *kernel* sin necesidad de recompilarlo nuevamente, por ello se van a crear dos archivos DTB uno para iniciar Embedded Linux con el sistema de archivos del ELDK y otro para iniciar Debian con el sistema de archivos que generaremos en la sección A.12. Las modificaciones realizadas sobre el archivo DTS se muestran en los siguientes códigos para cada uno de los dos sistemas de archivos que se van a emplear, donde la diferencia está en la asignación de la variable `nfsroot`.

```
Okinawa-00# mv \
/opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dts/virtex440-ml507.dts \
/opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dts/virtex440-ml507_debian.dts
```

/opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dts/virtex440-ml507_debian.dts

```
1
2 bootargs = "console=ttyS0 ip=on root=/dev/nfs " \
3            "nfsroot=192.168.16.150:/opt/debian/ppc_4xx,tcp rw"
4 local-mac-address = [ 00 0A 35 01 D7 4E ];
```

```
Okinawa-00# mv \
/opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dts/virtex440-ml507.dts \
 /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dts/virtex440-ml507_emlinux.dts
```

/opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dts/virtex440-ml507_emlinux.dts

```
1
2 bootargs = "console=ttyS0 ip=on root=/dev/nfs " \
3            "nfsroot=192.168.16.150:/opt/ELDK/4.2/ppc_4xx,tcp rw"
4 local-mac-address = [ 00 0A 35 01 D7 4E ];
```

ahora si compilar el archivo DTS para generar el archivo DTB

```
Okinawa-00# /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dtc \
              -b 0 -V 17 -R 4 -S 0x3000 -I dts -O dtb -o ml507_debian.dtb \
              -f arch/powerpc/boot/dts/virtex440-ml507_debian.dts
```

```
Okinawa-00# /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/dtc \
             -b 0 -V 17 -R 4 -S 0x3000 -I dts -O dtb -o ml507_emlinux.dtb \
             -f arch/powerpc/boot/dts/virtex440-ml507_emlinux.dts
```

copiar los archivos generados en la carpeta dentro del servidor TFTP, donde `u-boot` va a buscar sus archivos, es decir `/compartido/tftpboot/Linuxboot/`

```
Okinawa-00# cp /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/arch/powerpc/boot/uImage \
              /compartido/tftpboot/Linuxboot
Okinawa-00# cp /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/ml507_debian.dtb \
              /compartido/tftpboot/Linuxboot
Okinawa-00# cp /opt/SoftDev/Xilinx/linux-2.6-xlnx.git/ml507_emlinux.dtb \
              /compartido/tftpboot/Linuxboot
```

Teniendo en cuenta el código de `u-boot` que inicia la tarjeta busca el archivo `ml507.dtb`, entonces se debe crear este archivo copiándolo de `ml507\_emlinux.dtb`.

```
Okinawa-00# cp /compartido/tftpboot/Linuxboot/ml507_emlinux.dtb \
  /compartido/tftpboot/Linuxboot/ml507.dtb
```

Ahora el sistema está listo para iniciar Embedded Linux, por lo tanto se puede energizar la tarjeta y se debería cargar `u-boot` y posteriormente el *kernel* y el sistema de archivos, donde al final se inicia el *login* de Linux.

A.12 Creación del sistema de archivos para debian

Durante esta sección se empleará Embedded Linux para crear y configurar el sistema de archivos de Debian. El proceso se ilustra en la Figura A.20.

La creación del sistema de archivos se hace en dos etapas: La primera descarga los paquetes y crea la base de la estructura de los directorios y se realiza mediante el siguiente comando, teniendo en cuenta que se ha planeado la implementación del sistema de archivos en el directorio `/opt/debian/ppc_4xxFP/`

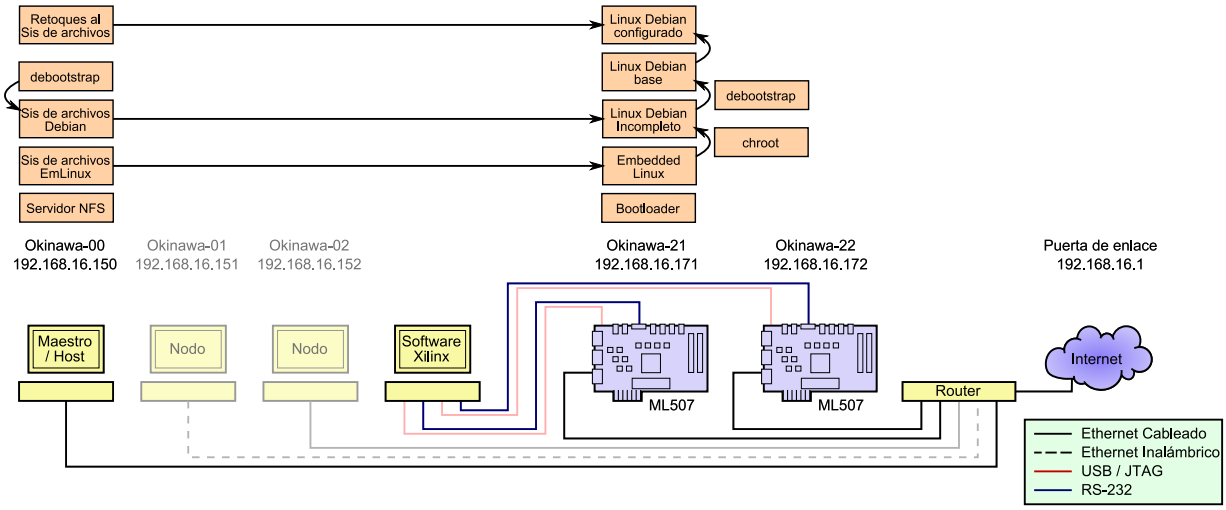


Figura A.20: Elementos involucrados en la creación del sistema de archivos para debian.

```
Okinawa-00# cp /compartido/tftpboot/Linuxboot/ml507_emlinux.dtb \
              /compartido/tftpboot/Linuxboot/ml507.dtb
```

Ahora para ejecutar la segunda parte, debemos hacerlo desde la tarjeta, para lo cual debemos iniciar Embedded Linux y montar el sistema de archivos de debian en un directorio del de Embedded Linux así:

```
192# mkdir -p /root/debian
192# mount -t proc proc /root/debian
192# mount 192.168.16.150:/opt/debian/ppc_4xxFP /root/debian/
192# mount chroot /root/debian
```

Con el último comando se ha cambiado la raiz de nuestro sistema de archivos, por lo tanto podemos ejecutar la segunda etapa del debootstrap mediante el comando:

```
192# /debootstrap/debootstrap --second-stage
```

Ahora desde el frontend, podemos modificar los archivos de configuración del sistema de desarrollo. Principalmente los siguientes:

/opt/debian/ppc_4xxFP/etc/hosts

```
1
2 127.0.0.1       localhost
3 192.168.16.172  Okinawa-22.okinawa.net  Okinawa-22
4
5 # The following lines are desirable for IPv6 capable hosts
6 ::1     localhost ip6-localhost ip6-loopback
7 fe00::0 ip6-localnet
8 ff00::0 ip6-mcastprefix
9 ff02::1 ip6-allnodes
10 ff02::2 ip6-allrouters
11 ff02::3 ip6-allhosts
```

/opt/debian/ppc_4xxFP/etc/resolv.conf

```
1
2 #domain uis.edu.co
3 search okinawa.net
4 nameserver 192.168.16.150
```

/opt/debian/ppc_4xxFP/etc/network/interfaces

```
1
2 auto lo
3 iface lo inet loopback
4
5 # Interfaz de red cableada
6 auto eth0
7 iface eth0 inet static
8 address 192.168.16.172
9 network 192.168.16.0
10 netmask 255.255.255.0
11 broadcast 192.168.16.255
12 gateway 192.168.16.1
13 dns-nameservers 192.168.19.2
14 #dns-search uis.edu.co
```

/opt/debian/ppc_4xxFP/etc/hostname

```
1
2 Okinawa-22
```

/opt/debian/ppc_4xxFP/etc/fstab

```
1
2 proc                                  /proc  proc   defaults  0  0
3 tmpfs                                 /tmp   tmpfs  defaults  0  0
4 192.168.16.150:/opt/debian/ppc_4xxFP/  /      nfs    defaults  0  0
```

Finalmente se ajusta el nombre del archivo DTB para que se inicie el sistema desde el sistema de archivos debian así:

```
Okinawa-00# mv /compartido/tftpboot/Linuxboot/ml507.dtb \
               /compartido/tftpboot/Linuxboot/ml507_emlinux.dtb
Okinawa-00# mv /compartido/tftpboot/Linuxboot/ml507_debian.dtb \
               /compartido/tftpboot/Linuxboot/ml507.dtb
```

Y finalmente se reinicia la tarjeta. Si el proceso se ha realizado correctamente, el sistema de desarrollo debe iniciar `u-boot` y posteriormente cargar el *kernel* y el sistema de archivos de debian para finalmente indicar en el terminal el login del sistema operativo.

A.13 Configuración del sistema operativo de la tarjeta

En esta sección se busca hacer la configuración del sistema operativo para que pueda ser parte del *cluster*. Inicialmente debemos configurar el repositorio del sistema operativo en el archivo `/etc/apt/sources.list`

```
deb http://ftp.debian.org/debian lenny main contrib non-free
```

Para indicar a la tarjeta ML507 cual es el servidor DNS al cual debe acceder, se debe editar el archivo `/etc/resolv.conf` de la siguiente forma

```
#domain uis.edu.co
search okinawa.net
nameserver 192.168.16.150
```

Posteriormente consultamos la lista de paquetes disponibles y la actualizamos mediante el comando

```
192# apt-get update
```

Podemos instalar algunas herramientas que pueden ser de utilidad como

```
192# apt-get install ssh vim host
```

Debido a que el sistema de desarrollo no tiene un reloj que le permita actualizar la hora, puede ser necesario la configuración de un cliente Network Time Protocol(NTP) que le permita sincronizar la hora. Para ello instalamos el paquete ntp

```
192# apt-get install ntp
```

y ahora se inicia el servicio

```
192# /etc/init.d/ntp start
```

El no tener la hora del sistema ajustada, puede traer advertencias y errores que pueden ser dificiles de rastrear o incomodos. Por ejemplo que el sistema solicite el cambio de la clave de root cada vez que se accede a él.

A.14 Configuración del directorio compartido en el *cluster*

En este momento el frontend comparte su directorio `/compartido/comun` al nodo trabajador, el cual lo monta en la misma ubicación. Dentro de este directorio se encuentra el directorio `home` que contiene las carpetas de los usuarios del *cluster*. Es necesario que la tarjeta pueda acceder al contenido del directorio y que pueda crear los mismos usuarios configurados en el frontend y en el nodo trabajador. Se inicia entonces modificando el archivo `/etc/fstab` de la tarjeta para configurar el modo en el que el sistema monta la carpeta compartida.

/etc/fstab

```
1 # <file system>                    <mount point>    <type> <options> <dump> <pass>
2   proc                                 /proc      proc   defaults      0      0
3   tmpfs                                /tmp       tmpfs  defaults      0      0
4   192.168.16.150:/opt/debian/ppc_4xxFP/ /         nfs    defaults      0      0
5   192.168.16,150:/compartido/comun/  /compartido/comun/ nfs defaults  0      0
```

Probablemente sea necesario instalar el cliente nfs llamado `nfs-common`, para lo cual se ejecuta

```
Okinawa-22# apt-get nfs-common
```

Ahora montamos el directorio configurado en el archivo `fstab` mediante el comando

```
Okinawa-22# mount -a
```

y podemos verificar que si lo hizo bien mediante el comando

```
Okinawa-22# mount
```

Ahora, antes de crear los nuevos usuarios, debemos configurar el archivo que contiene la configuración de los nuevos usuarios de la misma forma que se hizo previamente en el *cluster*. Para ello modificamos el archivo `/etc/adduser.conf`. Allí modificaremos las siguientes entradas:

/etc/fstab

```
1 DHOME=/compartido/comun/home
2 USERGROUPS=no
```

Ahora creamos los usuarios mediante el comando `adduser`, pero previamente debemos conocer el uid del usuario que vamos a crear. Para ello podemos observar el contenido del archivo `/etc/passwd` mediante el siguiente comando.

```
Okinawa-22# cat /etc/passwd | grep sergio
```

dentro de la lista de datos que se imprime, el tercero es el userid. Para este caso es 1002. Ahora si ejecutamos `adduser`.

```
Okinawa-22# adduser sergio --uid 1002
```

Esto creará el usuario sergio para el sistema de desarrollo ML507, pero el sistema no modificará los archivos ya existentes.

A.15 Configuración de otros servicios

A.15.1 Servidor DNS

Es necesario también añadir este nodo al servidor DNS, para lo cual se modificarán sus archivos de configuración.

/etc/bind/db.16.168.192

```
$TTL  24h

16.168.192.in-addr.arpa.  IN SOA  Okinawa-00.okinawa.net  root.okinawa.net  (
  2008042300  ;numero serial
  3h  ;  tiempo de refresco
  30m  ;  tiempo de reintegro
  7d  ;  tiempo de "expire"
  3h  ;  negative caching ttl
)
;  NameServers
16.168.192.in-addr.arpa.  IN NS  192.168.16.150.

;  Hosts
150.16.168.192.in-addr.arpa.  IN PTR  Okinawa-00.okinawa.net.
151.16.168.192.in-addr.arpa.  IN PTR  Okinawa-01.okinawa.net.
152.16.168.192.in-addr.arpa.  IN PTR  Okinawa-02.okinawa.net.
171.16.168.192.in-addr.arpa.  IN PTR  Okinawa-21.okinawa.net.
172.16.168.192.in-addr.arpa.  IN PTR  Okinawa-22.okinawa.net.
```

/etc/bind/db.okinawa.net

```
$TTL  24h
okinawa.net.  IN SOA  Okinawa-00.okinawa.net  root.okinawa.net (
  2008042300  ;  numero  serial
  3h  ;  tiempo de refresco
  30m  ;  tiempo de reintegro
  7d  ;  tiempo de expire
  3h  ;  negative caching ttl
)

; NameServers
okinawa.net.  IN NS  192.168.16.150.

; Hosts
Okinawa-00.okinawa.net.  IN A  192.168.16.150
Okinawa-01.okinawa.net.  IN A  192.168.16.151
Okinawa-02.okinawa.net.  IN A  192.168.16.152
Okinawa-21.okinawa.net.  IN A  192.168.16.171
Okinawa-22.okinawa.net.  IN A  192.168.16.172
```

A.16 Instalación de PVM

La instalación de PVM requiere que este sea compilado para varias arquitecturas y se iniciará con la del frontend, es decir la x86. Para ello descargamos la fuente desde el sitio `http://www.netlib.org/pvm3/`. El archivo fuente se ha ubicado en el directorio `/compartido/comun` y se ha descomprimido en el directorio `/compartido/comun/pvm3` mediante los comandos

```
Okinawa-22# cd /compartido/comun/
Okinawa-22# tar xvfz pvm3.4.6.tar.gz
```

Debido a que esta fuente será accedida por varios usuarios y equipos, se pueden modificar sus propiedades. Para este caso se darán todos los permisos mediante el siguiente comando

```
Okinawa-22# chmod 777 -R /compartido/comun/pvm3
```

luego se procedió a crear las variables de entorno que requeridas por la instalación editando el archivo `/etc/profile`. Este script se ejecuta cada vez que se inicia sesión, por lo tanto estas variables estarán disponibles en futuras ocaciones.

/etc/profile

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "‘id -u‘" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/usr/local/bin:/usr/bin:/bin:/usr/games"
fi

if [ "$PS1" ]; then
  if [ "$BASH" ]; then
    PS1='\u@\h:\w\$ '
  else
    if [ "‘id -u‘" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

export PATH

umask 022

# contenido anadido para la instalacion de PVM

PVM_ROOT=/compartido/comun/pvm3
PVM_DPATH=$PVM_ROOT/lib/pvmd
PATH=$PATH:$PVM_ROOT/lib
export PVM_ROOT PVM_DPATH PATH
```

Finalmente reiniciamos la consola en la que estamos trabajando, vamos al directorio raiz de pvm y ejecutamos `make`

```
Okinawa-22# cd $PVM_ROOT
Okinawa-22# make
```

Con se han creado las las librerías de PVM. Ahora debemos copiarlas a un sitio donde los compiladores las encuentren en cada uno de los nodos donde se piense realizar la compilación. Para ello se creó el siguiente script.

/etc/profile

```
for x in ‘cat machines‘
do
  ssh $x cp /compartido/comun/pvm3/lib/LINUX/*.a /usr/lib/
  ssh $x cp /compartido/comun/pvm3/include/*.h /usr/include/
done
```

Este script, asume la existencia de un archivo machines que lista en cada línea los nodos donde se desean copiar las librerías. Ahora se procederá a ejecutar una prueba desde uno de los usuarios creados. Para cambiar de usuario se puede ejecutar el siguiente comando:

```
Okinawa-22# su -l william
```

El primer paso para emplear PVM es configurarlo para que reconozca los nodos del sistema. Para ello abrimos la consola de la siguiente forma:

```
william@Okinawa-22# pvm
pvm>  conf
pvm>  add Okinawa-00
pvm>  add Okinawa-01
pvm>  conf
pvm>  quit
```

Ahora creamos el archivo .rhost que indicará en la carpeta del usuario, a cuales nodos es posible acceder.

/compartido/comun/home/william/.rhost

```
1 Okinawa-00 william
2 Okinawa-01 william
```

Creamos un directorio donde haremos una prueba de hola mundo que imprime el nombre de cada uno de los nodos y la hora del sistema. En ella pondremos los archivos fuente `helloPVM_master.c` y `helloPVM_slave.c`.

```
william@Okinawa-22# mkdir pruebaPVM
william@Okinawa-22# cd pruebaPVM
```

/compartido/comun/home/william/pruebaPVM/helloPVM_master.c

```
/* master program for the simple communication program */
/* which starts slaves just to get their names and the */
/* local time back */

#include <stdio.h>
#include <pvm3.h>

main()
{

  struct pvmhostinfo *hostp;
  int result, check, i, nhost, narch, stid;
  char buf[64];
  pvm_setopt(PvmRoute, PvmRouteDirect);    /* channel for communication */

  gethostname(buf, 20);        /* get name of master */
  printf("The master process runs on %s \n", buf);

  /* get and display configuration of the parallel machine */
  pvm_config( &nhost, &narch, &hostp );    /* get configuration */
  printf("I found the following hosts in your virtual machine\n");
  for (i = 0; i < nhost; i++)
  {
     printf("\t%s\n", hostp[i].hi_name);
  }

  for (i=0; i<nhost; i++)        /* spawn processes on */
  {            /* all physical machines */
     printf("Comienzo intento con el servidor numero %d\n",i);
check=pvm_spawn("helloPVM_slave", 0,PvmTaskHost,hostp[i].hi_name, 1, &stid);
   if (!check) printf("Couldn't start process on %s\n", hostp[i].hi_name);
  }
  result=0;
  while (result<nhost)
  {
     pvm_recv(-1, 2);      /* wait for reply message */
     pvm_upkstr(buf);      /* unpack message */
     printf("%s\n", buf);      /* print contents */
     result++;        /* next message */
  }

  pvm_exit;        /* we are done */
}
```

/compartido/comun/home/william/pruebaPVM/helloPVM_slave.c

```
/* slave program for PVM-first-try */
/* returns string consisting of machine name and local time */

#include <stdio.h>
#include <pvm3.h>
#include <time.h>

main()
{
  time_t now;
  char name[12], buf[60];
  int ptid;

  ptid = pvm_parent();    /* the ID of the master process */
  pvm_setopt(PvmRoute, PvmRouteDirect);

  gethostname(name, 64);    /* find name of machine */
  now=time(NULL);      /* get time */
  strcpy(buf, name);    /* put name into string */
  strcat(buf, "'s time is ");
  strcat(buf, ctime(&now));  /* add time to string */

  pvm_initsend(PvmDataDefault);  /* allocate message buffer */
  pvm_pkstr(buf);      /* pack string into buffer */
  pvm_send(ptid, 2);    /* send buffer to master */

  pvm_exit;      /* slave is done and exits */
}
```

Ahora compilaremos la fuente indicando al compilador que use la librería pvm3 mediante el siguiente comando.

```
william@Okinawa-22# cc helloPVM_master.c -o helloPVM_master -lpvm3
william@Okinawa-22# cc helloPVM_slave.c -o helloPVM_slave -lpvm3
```

Con esto se han creado los dos ejecutables, el del maestro y el de los esclavos. Ahora se debe copiar el ejecutable a la carpeta donde se buscan estos ejecutables/.

```
william@Okinawa-22# cp helloPVM_slave /compartido/comun/pvm3/bin/LINUX
```

Ahora finalmente se ejecutará el archivo helloPVM_master mediante el siguiente comando

```
william@Okinawa-22# ./helloPVM_master
```

Con lo que tendremos una salida de este tipo:

```
william@Okinawa-00:~/pruebaPVM$ ./helloPVM_master
The master process runs on Okinawa-00
I found the following hosts in your virtual machine
        Okinawa-00
        Okinawa-01
Comienzo intento con el servidor numero 0
Comienzo intento con el servidor numero 1
Okinawa-00's time is Tue Jan 26 10:58:50 2010

Okinawa-01's time is Tue Jan 26 04:53:52 2010
```

A.17 Adición del FPGA a la instalación de PVM

Para que PVM pueda distribuir aplicaciones sobre el nodo FPGA, se debe compilar la fuente para el Procesador PowerPC®. Para ello se debe iniciar por modificar el archivo `/etc/profile` de la misma forma que se hizo en el caso de los procesadores x86. Luego se instalarán algunos paquetes que son necesarios para la ejecución de PVM mediante el siguiente comando:

```
william@Okinawa-22# apt-get install m4 gfortran
```

Ahora ya es posible compilar la fuente para esta arquitectura, para lo cual nos dirigimos al directorio de la fuente y ejecutamos `make`.

```
Okinawa-22# cd /compatido/comun/pvm3
Okinawa-22# make
```

Ahora debemos copiar las librerias dinámicas generadas y los cabeceros aprovechando el script creado para los procesadores x86, pero con algunas modificaciones:

/compartido/comun/home/william/pruebaPVM/helloPVM_slave.c

```
1 for x in 'cat fpgamachines'
2 do
3         ssh $x cp /compartido/comun/pvm3/lib/LINUXPPC/*.a /usr/lib/
4         ssh $x cp /compartido/comun/pvm3/include/*.h /usr/include/
5 done
```

y ejecutarlo

```
Okinawa-22# ./copiarlibreriasanodosfpga.sh
```

Ahora procederemos a hacer un programa de prueba para comprobar el funcionamiento de la herramienta. Esto lo haremos con el usuario william y crearemos una carpeta para guardar ahi los fuentes y los ejecutables creados.

```
Okinawa-22# su -l william
Okinawa-22# mkdir pruebaPVM2
Okinawa-22# cd pruebaPVM2
```

Los programas de prueba creados fueron los siguientes:

/compartido/comun/home/william/pruebaPVM2/helloPVM_master.c

```
1  /* master program for the simple communication program */
2  /* which starts slaves just to get their names and the */
3  /* local time back */
4
5  #include <stdio.h>
6  #include <pvm3.h>
7
8  main()
9  {
10
11 struct pvmhostinfo *hostp;
12 int result, check, i, nhost, narch, stid;
13 char buf[64];
14 pvm_setopt(PvmRoute, PvmRouteDirect);    /* channel for communication */
15
16 gethostname(buf, 20);        /* get name of master */
17 printf("The master process runs on %s \n", buf);
18
19 /* get and display configuration of the parallel machine */
20 pvm_config( &nhost, &narch, &hostp );    /* get configuration */
21 printf("I found the following hosts in your virtual machine\n");
22 for (i = 0; i < nhost; i++)
23 {
24    printf("\t%s\n", hostp[i].hi_name);
25 }
26
27 for (i=0; i<nhost; i++)        /* spawn processes on */
28 {            /* all physical machines */
29    printf("Comienzo intento con el servidor numero %d\n",i);
30 check=pvm_spawn("helloPVM_slave", 0,PvmTaskHost,hostp[i].hi_name, 1, &stid);
31    if (!check) printf("Couldn't start process on %s\n", hostp[i].hi_name);
32 }
33 result=0;
34 while (result<nhost)
35 {
36    pvm_recv(-1, 2);      /* wait for reply message */
37    pvm_upkstr(buf);      /* unpack message */
38    printf("%s\n", buf);      /* print contents */
39    result++;        /* next message */
40 }
41
42 pvm_exit;        /* we are done */
43 }
44
```

/compartido/comun/home/william/pruebaPVM2/helloPVM_slave.c

```
/* slave program for PVM-first-try */
/* returns string consisting of machine name and local time */

#include <stdio.h>
#include <pvm3.h>
#include <time.h>

main()
{
time_t now;
char name[13], buf[60];
int ptid;

ptid = pvm_parent();    /* the ID of the master process */
pvm_setopt(PvmRoute, PvmRouteDirect);

gethostname(name, 64);    /* find name of machine */
now=time(NULL);      /* get time */

//printf("Mi nombre es: %s\n",name);

strcpy(buf, name);    /* put name into string */
strcat(buf, "'s time is ");
//strcat(buf, " Now...  jeje");  /* add time to string */
strcat(buf, ctime(&now));  /* add time to string */

//printf("Ya genere© elbuffer: %s\n",buf);
//printf("Voy a inicializar el buffer de envio\n");

pvm_initsend(PvmDataDefault);  /* allocate message buffer */
pvm_pkstr(buf);      /* pack string into buffer */
pvm_send(ptid, 2);    /* send buffer to master */

pvm_exit;      /* slave is done and exits */
}

```

Luego se procedió a compilar los programas para cada arquitectura

```
Okinawa-00# cc helloPVM_master.c -o helloPVM_master -lpvm3
Okinawa-00# cc helloPVM_slave.c -o helloPVM_slave -lpvm3
```

```
Okinawa-22# cc helloPVM_slave.c -o helloPVM_slave_ppc -lpvm3
```

En cada caso se ubicaron los ejecutables de los esclavos en la carpeta de ejecutables

```
Okinawa-00# cp helloPVM_slave /compartido/comun/pvm3/bin/LINUX
```

```
Okinawa-22# cp helloPVM_slave_ppc \
       /compartido/comun/pvm3/bin/LINUXPPC/helloPVM_slave
```

Es importante que los ejecutables tengan el mismo nombre pero que estén ubicados en la carpeta de su respectiva arquitectura. Finalmente desde el frontend, se puede lanzar la aplicación desde la sesión de usuario.

```
Okinawa-00# su -l william
william@Okinawa-00# ./helloPVM_master
```

El resultado debe ser como se muestra a continuación:

```
Okinawa-00# ./helloPVM_master

The master process runs on Okinawa-00
I found the following hosts in your virtual machine
        Okinawa-00
        Okinawa-02
        Okinawa-21
Comienzo intento con el servidor numero 0
Comienzo intento con el servidor numero 1
Comienzo intento con el servidor numero 2
Okinawa-00's time is Thu Jan 20 16:15:45 2011

Okinawa-02's time is Thu Jan 20 09:01:18 2011

Okinawa-21's time is Thu Jan 20 16:15:45 2011
```

A.18 Instalación de LAM-MPI

Para la instalación de LAM-MPI se realizó la compilación de la fuente de forma nativa en cada una de las arquitecturas del *cluster*, y se instaló en la ruta `/compartido/{arch}/lam`. Adicionalmente se creó un enlace simbólico llamado `/nocompartido/lam` que apunta a esta carpeta y que permite a cada nodo acceder a su respectiva instalación de LAM-MPI.

Inicialmente se descargaron las fuentes de `http://www.lam-mpi.org/beta/` la versión beta 7.1.5b1. Se descomprimió en el directorio `/compartido/x86/` y en el directorio `/compartido/powerpc/` mediante los siguientes comandos

```
Okinawa-00# mv lam-7.1.5b1.tar.gz /compartido/x86
Okinawa-00# mv lam-7.1.5b1.tar.gz /compartido/powerpc
Okinawa-00# cd  /compartido/x86
Okinawa-00# tar xvfz lam-7.1.5b1.tar.gz
```

```
Okinawa-22# cd  /compartido/powerpc
Okinawa-22# tar xvfz lam-7.1.5b1.tar.gz
```

En el maestro, de arquitectura x86 y en el FPGA se configuró LAM-MPI indicando el directorio de instalación y sustituyendo el uso de rsh por ssh.

```
Okinawa-00# cd lam-7.1.5b1
Okinawa-00# ./configure CC=cc CXX=g++ --prefix=/nocompartido/lam \
```

```
Okinawa-22# cd lam-7.1.5b1
Okinawa-22# ./configure CC=cc CXX=g++ --prefix=/nocompartido/lam \
```

Ahora se procede a compilar e instalar los ejecutables y librerias.

```
Okinawa-00# make all
Okinawa-00# make install
```

```
Okinawa-22# make all
Okinawa-22# make install
```

El siguiente paso consiste en asegurar que las variables de entorno se asignen correctamente al iniciar sesiones remotas, para eso se optó por modificar el archivo `/compartido/comun/home/william/.bashrc` adicionando al inicio las siguientes líneas:

/compartido/comun/home/william/.bashrc

```
1
2 PATH=/compartido/x86/lam/bin:$PATH
3 MANPATH=/compartido/x86/lam/man:$MANPATH
4 export PATH MANPATH
```

También es necesario configurar el archivo de arranque de LAM-MPI en los nodos. Esto se puede realizar en un archivo separado que se invoca cada vez que se inicia el sistema o en el archivo localizado en `/compartido/x86/lam/etc/lam-bhost.def`. El archivo contiene las siguientes líneas:

/compartido/x86/lam/etc/lam-bhost.def

```
1 Okinawa-00 cpu=1
2 Okinawa-01 cpu=1
3 Okinawa-22 cpu=1
```

Para probar que la configuración se hizo correctamente se pueden inicializar los demonios de LAM sobre los nodos mediante el comando `lamboot` y se puede verificar previamente el estado del sistema mediante el comando `recon`. Todo esto se debe llevar a cabo como usuario.

```
Okinawa-00# su -l william
william@Okinawa-00# recon
william@Okinawa-00# lamboot
```

Con esta configuración el sistema está listo para iniciar la prueba. Por defecto LAM-MPI no se debería ejecutar como root, por ello lo haremos como el usuario `william` y en el directorio raiz del usuario creamos un directorio para la prueba.

```
william@Okinawa-00# mkdir pruebaLAM
william@Okinawa-00# cd pruebaLAM
```

Allí se creó el programa "Hola mundo" que tiene el siguiente contenido:

/compartido/comun/home/william/pruebaLAM/hello.c

```
1 #include <mpi.h>
2 #include <stdio.h>
3
4 int main(int argc, char *argv[])
5 {
6     int rank, size;
7     MPI_Init(&argc, &argv);
8
9     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
10    MPI_Comm_size(MPI_COMM_WORLD, &size);
11
12    printf("Hello, World.  I am %d of %d\n", rank, size);
13
14    MPI_Finalize();
15    return 0;
16 }
```

Este programa se debe compilar para cada arquitectura y se puede hacer mediante los siguienes comandos:

```
william@Okinawa-00# mpicc hello.c -o hello_x86
william@Okinawa-00# ssh Okinawa-22 -n mpicc pruebaLAM/hello.c \
                                    -o pruebaLAM/hello_ppc
```

Ahora para ejecutar el programa debemos se creó el archivo `my_appfile` que determina como se desplegará la aplicación en los nodos.

/compartido/comun/home/william/pruebaLAM/my_appfile

```
1
2 -host Okinawa-00.okinawa.net /compartido/comun/home/william/pruebaLAM/hello_x86
3 -host Okinawa-01.okinawa.net /compartido/comun/home/william/pruebaLAM/hello_x86
4 -host Okinawa-22.okinawa.net /compartido/comun/home/william/pruebaLAM/hello_ppc
```

y finalmente se ejecuta mediante el siguiente comando:

```
william@Okinawa-00# mpiexec -configfile my_appfile
```

A.19 Adición de nodo x86

Es posible clonar el discoduro y modificar los siguientes archivos

/etc/network/interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
#iface eth0 inet dhcp

iface eth0 inet static
address 192.168.16.152
network 192.168.16.0
netmask 255.255.255.0
broadcast 192.168.16.255
gateway 192.168.16.1
#dns-nameservers 192.168.19.2
#dns-search uis.edu.co

```

/etc/hostname

```
Okinawa-02
```

y con eso debería ser suficiente para que el nuevo nodo inicie.

A.20 Procedimiento para añadir un nodo FPGA

Para llevar a cabo la configuración del nodo FPGA, es necesario configurar el maestro del *cluster* y además hacer uso del proyecto de EDK sobre el ordenador que tiene el *software* de Xilinx® . Se comenzará con las modificaciones a `u-boot`, para lo cual en el frontend, modificamos el archivo `ml507.h`. Crearemos una copia de toda la fuente y en el archivo `ml507.h` se especificará la dirección IP del nodo y el nombre del archivo DTB que cargará del sistema operativo.

```
Okinawa-00# cd /opt/SoftDev/Xilinx/
Okinawa-00# cp -r u-boot-xlnx.git u-boot-xlnx.git-Okinawa-21
Okinawa-00# cd u-boot-xlnx.git-Okinawa-21
```

/opt/SoftDev/Xilinx/u-boot-xlnx.git-Okinawa-21/include/configs/ml507.h

```
Copiar el archivo
```

Posteriormente debemos compilar la fuente, para ello es necesario tener las variables del compilador cruzado configuradas. El procedimiento fue el siguiente.

```
Okinawa-00# source /opt/ELDK/4.2/eldk_init ppc_4xxFP
Okinawa-00# cd /opt/SoftDev/Xilinx/u-boot-xlnx.git-Okinawa-21/
Okinawa-00# make ml507_config
Okinawa-00# make
```

Con esto hemos creado nuevamente el archivo `u-boot.srec` que contiene la información del arranque de la tarjeta que estamos configurando y este archivo deberá ser guardado en la memoria Flash de la tarjeta por medio de EDK. Este procedimiento se realiza de la misma forma que el mostrado en la sección A.10.

El siguiente paso está relacionado con la compilación del archivo DTB que cargará la tarjeta. Para ello, se realizó una copia de la fuente del *kernel*

```
Okinawa-00# cd /opt/SoftDev/Xilinx/
Okinawa-00# cp -r linux-2.6-xlnx.git linux-2.6-xlnx.git-Okinawa-21
Okinawa-00# cd linux-2.6-xlnx.git-Okinawa-21
```

Ahora se modificó el archivo DTS y se compiló para finalmente ser puesto en el directorio compartido Linuxboot `/compartido/tftpboot/Linuxboot`:

```
Okinawa-00# /opt/SoftDev/Xilinx/linux-2.6-xlnx.git-Okinawa-21/arch/powerpc/boot/dtc \
                      -b 0 -V 17 -R 4 -S 0x3000 -I dts -O dtb -o ml507-Okinawa-21.dtb \
                      -f arch/powerpc/boot/dts/virtex440-ml507.dts
Okinawa-00# cp ml507-Okinawa-21.dtb /compartido/tftpboot/Linuxboot/
```

Finalmente solo queda crear el sistema de archivos del nuevo sistema copiando el que ya existe para lo cual se ejecutó:

```
Okinawa-00# cd /opt/debian/
Okinawa-00# cp -r ppc_4xxFP ppc_4xxFP_Okinawa-21
Okinawa-00# cd /opt/SoftDev/Xilinx/linux-2.6-xlnx.git-Okinawa-21
Okinawa-00# cd /opt/SoftDev/Xilinx/u-boot-xlnx.git-Okinawa-21/
Okinawa-00# make ml507_config
Okinawa-00# make
```

Ahora se debe asegurar que el sistema exporte el nuevo sistema de archivos modificando el archivo `/etc/exports` y reiniciando el servicio

/etc/exports

```
1 Copiar el archivo
```

```
Okinawa-00# /etc/init.d/nfs-kernel-server restart
```

Ahora, antes de encender la tarjeta con el sistema operativo, se pueden modificar algunos archivos desde el maestro.

/opt/debian/ppc_4xxFP_Okinawa-21/etc/hostname

```
1 Okinawa-21
```

/opt/debian/ppc_4xxFP_Okinawa-21/etc/hosts

```
1 127.0.0.1  localhost
2 192.168.16.171  Okinawa-21.okinawa.net  Okinawa-21
3 130.89.149.226  ftp.debian.org
4 # The following lines are desirable for IPv6 capable hosts
5 ::1     localhost ip6-localhost ip6-loopback
6 fe00::0 ip6-localnet
7 ff00::0 ip6-mcastprefix
8 ff02::1 ip6-allnodes
9 ff02::2 ip6-allrouters
10 ff02::3 ip6-allhosts
```

/opt/debian/ppc_4xxFP_Okinawa-21/etc/network/interfaces

```
1 # This file describes the network interfaces available on your system
2 # and how to activate them. For more information, see interfaces(5).
3
4 # The loopback network interface
5 auto lo
6 iface lo inet loopback
7
8 # Interfaz de red cableada
9 auto eth0
10 iface eth0 inet static
11 address 192.168.16.171
12 network 192.168.16.0
13 netmask 255.255.255.0
14 broadcast 192.168.16.255
15 gateway 192.168.16.1
16 dns-nameservers 192.168.19.2
17 dns-search uis.edu.co
18
```

/opt/debian/ppc_4xxFP_Okinawa-21/etc/fstab

```
1 # /etc/fstab: static file system information.
2 #
3 # <file system> <mount point>   <type>  <options>       <dump>  <pass>
4 proc  /proc  proc  defaults  0  0
5 tmpfs  /tmp  tmpfs  defaults  0  0
6 192.168.16.150:/opt/debian/ppc_4xxFP_Okinawa-21/  /  nfs  defaults  0  0
7 192.168.16.150:/compartido/comun  /compartido/comun  nfs  defaults  0  0
8 192.168.16.150:/compartido/powerpc /compartido/powerpc nfs defaults  0  0
```

A.21 Soporte para el desarrollo de *drivers*

Dentro del proyecto va a ser necesario comprender varios aspectos del sistema operativo y se van a desarrollar algunos *drivers*, por lo cual se debe instalar las herramientas necesarias para el trabajo con estos. El procedimiento que se presenta se implementó en el nodo FPGA y en los nodos x86. Es básico para el desarrollo del *driver* tener el árbol donde estan las fuentes del *kernel* que se está corriendo, por ello este es el primer paso.

A.21.1 Implementación del árbol de fuentes sobre el nodo Okinawa-02 (x86)

En debian, basta con instalar el paquete `linux-headers-2.6-686`. Esto ajusta las fuentes del *kernel* que se tiene instalado y además descarga el compilador adecuado para este compilador. Para ello basta con ejecutar el siguiente comando

```
Okinawa-02# apt-get install linux-headers-2.6-686
```

Posteriormente se creó un módulo de prueba, para lo cual se crearon dos archivos `hello.c` y `Makefile`. El primero es la fuente del módulo, mientras que el segundo nos permite compilarlo empleando algunas funcionalidades especiales de `make`.

hello.c

```
1
2 #include<linux/init.h>
3 #include<linux/module.h>
4 MODULE_LICENSE("Dual BSD/GPL");
5
6 static int hello_init(void)
7 {
8   printk(KERN_EMERG "Hola Mundo\n");
9   return 0;
10 }
11
12 static void hello_exit(void)
13 {
14   printk(KERN_ALERT "Bye bye, Mundo\n");
15 }
16
17 module_init(hello_init);
18 module_exit(hello_exit);
```

Makefile

```
1
2 obj-m   += hello.o
3
4 all:
5   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
6
7 clean:
8   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

En este punto ya es posible compilar el módulo de prueba mediante la siguiente línea de comando

```
Okinawa-02# make
```

Igualmente si se quieren eliminar los archivos generados se puede ejecutar “hhh”

```
Okinawa-02# make clean
```

Con esto se debe haber creado el archivo hello.ko, que es el módulo en sí. Para cargar el módulo se puede ejecutar

```
Okinawa-02# insmod hello.ko
```

Para observar si el módulo está añadido al *kernel*, podemos ejecutar

```
Okinawa-02# lsmod
```

Para remover el módulo, se debe usar el siguiente comando

```
Okinawa-02# rmmod hello
```

A.21.2 Implementación del árbol de fuentes sobre el nodo Okinawa-21 (FPGA)

Aunque la distribución instalada sobre los nodos FPGA es debian, el *kernel* que está corriendo fue compilado desde su fuente mediante compiladores cruzados en el nodo maestro. Esto quiere decir que si se quiere desarrollar un módulo para este *kernel*, es necesario tener la fuente de la cual se compiló; para ello, desde el frontend, se copió el arbol de la fuente hasta un directorio compartido por NFS.

```
Okinawa-00# cp -r /opt/SoftDev/Xilinx/linux-2.6-xlnx.git-Okinawa-21 \
                /compartido/powerpc/
Okinawa-00# chmod 777 -R /compartido/powerpc/linux-2.6-xlnx.git-Okinawa-21
```

Adicionalmente es necesario crear un enlace simbólico en `/lib/modules/‘uname -r‘` que se llame `build` y que apunte a donde se copió la fuente. Esto se puede hacer desde la tarjeta mediante el comando.

```
Okinawa-21# mkdir /lib/modules/‘uname -r‘
Okinawa-21# ln -s /compartido/powerpc/linux-2.6-xlnx.git-Okinawa-21 \
                /lib/modules/‘uname -r‘/build
```

Ahora es necesario compilar el *kernel* desde la tarjeta, debido a que al interior de las fuentes existen unos ejecutables que deben ser creados para la máquina que compila el *kernel*. Para este punto estos ejecutables son para la arquitectura x86. Para compilar el *kernel* ejecutamos:

```
Okinawa-21# cd /compartido/powerpc/linux-2.6-xlnx.git-Okinawa-21
Okinawa-21# make clean
Okinawa-21# make 44x/virtex5_defconfig
Okinawa-21# make uImage
```

Con estos comandos, se ha hecho el mismo trabajo que se realizó al instalar el paquete del árbol de fuentes. Ahora se probó la instalación de la fuente mediante el mismo módulo de hola mundo que se creó para el nodo x86. Para compilarlos y añadirlos al *kernel*, se ejecutaron los mismos comandos.

```
Okinawa-21# make
Okinawa-21# insmod hello.ko
Okinawa-21# lsmod
Okinawa-21# rmmod hello
```

Script tcl para la generación del archivo bmm requerido por data2mem

El objetivo de este script es el de generar un archivo de extensión bmm creado y empleado por EDK. En él se especifica la localización de los bloques RAM (RAMB) empleados en la síntesis del periférico `xps_bram_if_cntlr_1_bram`. Esta información es empleada por el programa data2mem que se encarga de introducir el archivo binario del programa de arranque (pre-bootloader) en esta memoria. A continuación en los Cuadros de Código B.1 y B.2 se muestra el script y el archivo generado.

B.1 Archivo `crea_bmm.tcl`

```
set file_bmm [open "system_bd_LaT.bmm" w];

puts $file_bmm "// BMM_LOC annotation file.
//
// Release 10.1i - Data2MEM K.39, build 1.5.7 Nov 8, 2007
// Copyright (c) 1995-2010 Xilinx, Inc.  All rights reserved.
//
// Created on 10/06/10 10:01 am
//

//////////////////////////////////////////////////////////////////////////////
//
// Processor 'ppc440_0', ID 100, memory map.
//
//////////////////////////////////////////////////////////////////////////////

ADDRESS_MAP ppc440_0 PPC440 100

    ////////////////////////////////////////////////////////////////////////////
    //
    // Processor 'ppc440_0' address space 'xps_bram_if_cntlr_1_bram_combined'
    // 0xFFFF0000:0xFFFFFFFF (64 KB).
    //
    ////////////////////////////////////////////////////////////////////////////

    ADDRESS_SPACE xps_bram_if_cntlr_1_bram_combined RAMB32 \[0xFFFF0000:0xFFFFFFFF\]
        BUS_BLOCK"



set x [get_cells -hierarchical ramb36*]
set file_p [open "a.txt" w]; foreach i $x {puts $file_p [report_property $i]}; close $file_p
set file_p [open "a.txt" r];
set file_data [read $file_p]
```

```
close $file_p
set data [split $file_data "\n"]
set indice 60
foreach line $data {
    regexp  {LOC.*RAMB36_(.{5})}  $line matched bram_loc
    regexp  {name.*/xps_bram_if_cntlr_1_bram/(.{9})}  $line matched bram_name
    if {[info exists bram_name] == 1} {if {[info exists bram_loc] == 1} {
  puts -nonewline $file_bmm "                xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/"
        puts -nonewline $file_bmm $bram_name
        puts $file_bmm " \[[expr $indice + 3]:$indice\] PLACED = $bram_loc;"
        set indice [expr $indice - 4]
        unset bram_name
        unset bram_loc
    }}
}

puts $file_bmm "        END_BUS_BLOCK;
    END_ADDRESS_SPACE;

END_ADDRESS_MAP;"


close $file_bmm
```

Cuadro de Código B.1: Script para generar el archivo bmm

B.2 Archivo system_bd.bmm

```
// BMM LOC annotation file.
//
// Release 10.1i - Data2MEM K.39, build 1.5.7 Nov 8, 2007
// Copyright (c) 1995-2010 Xilinx, Inc.  All rights reserved.
//
// Created on 10/06/10 10:01 am
//

//////////////////////////////////////////////////////////////////////////////
//
// Processor 'ppc440_0', ID 100, memory map.
//
//////////////////////////////////////////////////////////////////////////////

ADDRESS_MAP ppc440_0 PPC440 100

    ////////////////////////////////////////////////////////////////////////
    //
    // Processor 'ppc440_0' address space 'xps_bram_if_cntlr_1_bram_combined'
    //         0xFFFF0000:0xFFFFFFFF (64 KB).
    //
    ////////////////////////////////////////////////////////////////////////

    ADDRESS_SPACE xps_bram_if_cntlr_1_bram_combined RAMB32 [0xFFFF0000:0xFFFFFFFF]
        BUS_BLOCK
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_0  [63:60] PLACED = X0Y12;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_1  [59:56] PLACED = X0Y13;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_2  [55:52] PLACED = X0Y10;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_3  [51:48] PLACED = X3Y9 ;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_4  [47:44] PLACED = X0Y11;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_5  [43:40] PLACED = X2Y8 ;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_6  [39:36] PLACED = X2Y11;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_7  [35:32] PLACED = X3Y15;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_8  [31:28] PLACED = X0Y15;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_9  [27:24] PLACED = X0Y14;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_10 [23:20] PLACED = X2Y10;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_11 [19:16] PLACED = X3Y11;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_12 [15:12] PLACED = X1Y11;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_13 [11:8] PLACED = X2Y9 ;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_14 [7:4] PLACED = X3Y14;
            xps_bram_if_cntlr_1_bram/xps_bram_if_cntlr_1_bram/ramb36_15 [3:0] PLACED = X3Y16;
        END_BUS_BLOCK;
    END_ADDRESS_SPACE;
```

```
END_ADDRESS_MAP;
```

Cuadro de Código B.2: Archivo bmm generado

Apéndice C

Script de Matlab para la interpretación de los *bitstreams*.

Mediante este script, se condensa la información inferida sobre los *bitstreams* a partir de los datasheets de Xilinx® y de la ingeniería inversa. El script está seccionado en varias subfunciones tal como lo indica el árbol de dependencias de la Figura C.1. A continuación se describe la función de cada uno de ellos:

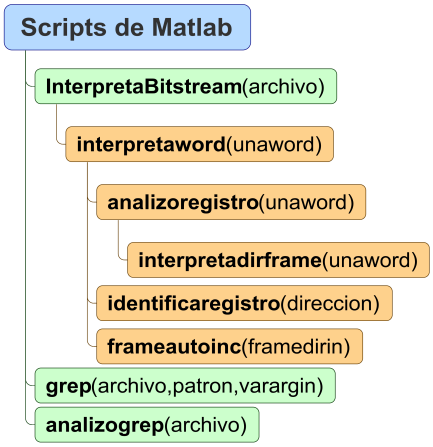


Figura C.1: Scripts de matlab para interpretar los *bitstreams*.

C.1 Función principal: InterpretaBitstream(archivo)

Éste es el script principal que hace llamado a las demás funciones creadas. Toma como argumento la ruta del *bitstream* y crea un archivo de texto llamado igual que el archivo original, pero con extensión .out, en el cual se presentan los resultados de la interpretación del *bitstream*. La Figura C.1 muestra el código fuente de este script.

```
function resultado = InterpretaBitstream(archivo)
%% Variables globales
% DEBUG define si se imprimen los mensajes de depuración
```

```
global debug;
debug = 1;
% TIPOPAQUETE define si se ha detectado anteriormente un paquete de tipo I
% o de tipo II
global tipopaquete
tipopaquete = 0;
% SID y BID son los identificadores de los archivos de entrada y salida.
global sid bid
%% Validación de los archivos de entrada
% Imprime el archivo que va a Interpretar
if (debug)
    fprintf('\nVoy a interpretar el archivo %s\n',archivo);
end
% Identificación del tipo de archivo (no se implementó)
% TIPOARCHIVO está codificado así:
%   1 - ascii
%   2 - binario
%   3 - binario (imprimir en consola los 100 primeros bytes)
tipoarchivo = 2;
%% Abrir el archivo de entrada y archivo de salida
% Abrimos el archivo de entrada
bid = fopen(archivo,'r');
% Pregunto si el archivo de entrada se abrió mal
if bid<0
    % Digo que hubo error
    if (debug)
        fprintf('Error al abrir el archivo de entrada\n');
    end
    error('El archivo no se pudo abrir.')
else
    % Digo que el archivo abrió bien
    if (debug)
        fprintf('Abrí el archivo de entrada con bid=%d\n',bid);
    end
end
% Abrimos el archivo de salida
archivosalida = strcat(archivo,'.out');
sid = fopen(archivosalida,'w+');
% Pregunto si el archivo de entrada se abrió mal
if sid<0
    % Digo que hubo error
    if (debug)
        fprintf('Error al abrir el archivo de salida\n');
    end
    error('El archivo no se pudo abrir.')
else
    % Digo que el archivo abrió bien
    if (debug)
        fprintf('Abrí el archivo de salida con sid=%d\n',sid);
    end
end
%% Acciones a realizar según el tipo de archivo.
switch (tipoarchivo)
    case 1
%% Si es un archivo de texto (ascii)...
        tline = fgetl(bid);
        while(ischar(tline))
            if(escharbit(tline))
                fprintf(sid,'%s',interpretaword(tline));
                fwrite(sid, sprintf('\n'), 'char');
            else
                fprintf(sid,'%s\t(Info)',tline);
                fwrite(sid, sprintf('\n'), 'char');
            end
            tline = fgetl(bid);
        end
    case 2
%% Si es un archivo binario...
        fprintf(sid,'Interpretación de Bitstreams de Xilinx');
        fwrite(sid, sprintf('\n'), 'char');
        fprintf(sid,'Autor:\t\tWilliam Salamanca');
        fwrite(sid, sprintf('\n'), 'char');
%%   Leemos el cabecero
        % Lectura de los primeros bytes F0 y 0F
```

```
        A = fread(bid, 13, 'uint8');
        if (~compara(A,hex2dec(['00'; '09'; '0F'; 'F0'; '0F'; 'F0'; '0F'; 'F0'; '0F'; 'F0'; ...
                '00'; '00'; '01'])))
            error('No coinciden los primeros 13 bytes')
        end
%%      Buscamos el caracter a
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Buscamos el caracter a
        if (A(length(A)-1)~=double('a'))
            error('Esperaba el caracter a y me llegó %c',A(length(A)-1))
        end
%%      Extraemos el nombre del diseno
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Extraemos el nombre del diseño
        nombrediseno = A(2:(length(A)-1));
        fprintf(sid,'%s\t\t%s','Diseño:', char(nombrediseno));
        fwrite(sid, sprintf('\n'), 'char');
%%      Buscamos el caracter b
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Buscamos el caracter b
        if (A(length(A)-1)~=double('b'))
            error('Esperaba el caracter b y me llegó %d',A(length(A)-1))
        end
%%      Extraemos la referencia del FPGA
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Extraemos el nombre del diseño
        nombreFPGA = A(2:(length(A)-1));
        fprintf(sid,'%s\t\t%s','FPGA:', char(nombreFPGA));
        fwrite(sid, sprintf('\n'), 'char');
%%      Buscamos el caracter c
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Buscamos el caracter c
        if (A(length(A)-1)~=double('c'))
            error('Esperaba el caracter c y me llegó %d',A(length(A)-1))
        end
%%      Extraemos la fecha
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Extraemos la fecha
        lafecha = A(2:(length(A)-1));
        fprintf(sid,'%s\t\t%s','Fecha:', char(lafecha));
        fwrite(sid, sprintf('\n'), 'char');
%%      Buscamos el caracter d
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Buscamos el caracter d
        if (A(length(A)-1)~=double('d'))
            error('Esperaba el caracter d y me llegó %d',A(length(A)-1))
```

```
        end
%%   Extraemos la hora
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Extraemos la hora
        lahora = A(2:(length(A)-1));
        fprintf(sid,'%s\t\t%s','Hora:', char(lahora));
        fwrite(sid, sprintf('\n'), 'char');
%%   Buscamos el caracter e
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 1, 'uint8');
        while(A(length(A))~=0)
            A = [A fread(bid, 1, 'uint8')];
        end
        % Buscamos el caracter e
        if (A(length(A)-1)~=double('e'))
            error('Esperaba el caracter e y me llegó %d',A(length(A)-1))
        end
%%   Extraemos el dato adicional...
        % Leemos hasta que encontremos un espacio
        A = fread(bid, 3, 'uint8');
%         while(A(length(A))~=0)
%             A = [A fread(bid, 1, 'uint8')];
%         end
        % Extraemos la hora
        datoadicional = A;
        fprintf(sid,'%s\t%s-%s-%s','Dato Adicional:',num2str(datoadicional(1)), ...
            num2str(datoadicional(2)),num2str(datoadicional(3)));
        fwrite(sid, sprintf('\n'), 'char');
%%   Extraemos el bitstream...
        % Leemos hasta que encontremos un espacio
        A = fread(bid, inf, 'uint8');
        elbitstream = A(1:(length(A)-0));
        for i=4:4:length(elbitstream)
%             fprintf(sid,'%s',interpretaword(tline));
            palabra = [dec2bin(elbitstream(i-3),8) dec2bin(elbitstream(i-2),8) ...
                dec2bin(elbitstream(i-1),8) dec2bin(elbitstream(i-0),8)];
            fprintf(sid,'%2.2X%2.2X%2.2X%2.2X %s',elbitstream(i-3),elbitstream(i-2),...
                elbitstream(i-1),elbitstream(i-0),interpretaword(palabra));
            fwrite(sid, sprintf('\n'), 'char');
        end
    case 3
%% Si es un archivo binario y quiero conocer los 100 primero bytes...
        A = fread(bid, 100, 'uint8');
        for i=1:100
            disp([num2str(i) ' ' dec2bin(A(i),8) ' ' char(A(i))])
        end
    otherwise
end

%% Cerrar el archivo de entrada y de salida
% Se cierra el archivo de salida
res = fclose(bid);
if (debug)
    fprintf('Cerré el archivo con bid=%d\n',bid);
end
% Se cierra el archivo de salida
res = fclose(sid);
if (debug)
    fprintf('Cerré el archivo con sid=%d\n',sid);
end

resultado = 0;
```

Cuadro de Código C.1: Script para analizar *bitstream*

C.1.1 interpretaword(unaword)

Este script está encargado de estudiar cada uno de los datos de 32 bits que componen el *bitstream*. Estos datos pueden corresponder a un comando, un dato de un paquete tipo I o un dato de un paquete tipo II. Para ello debe tener cierta información de algunas palabras leidas con anterioridad y por ello se apoya en algunas variables globales.

(posible listado de las variables globales)

Como resultado de esta función, se devuelve una cadena de caracteres que contiene la palabra recibida seguida de un texto que representa la interpretación realizada. Estos son algunos ejemplos:

```
11111111111111111111111111111111 (Dummy word)
10101010100110010101010101100110 (Sync word)
00100000000000000000000000000000 (NO OP      tipo I)
00110000000000100000000000000001 (Escritura tipo I REG=16 WBSTAR WCOUNT=1)
00000000000000000000000000000000 (Dato      tipo I REG=16 WBSTAR N=1 )
00110000000000001000000000000001 (Escritura tipo I REG=4 CMD WCOUNT=1)
00000000000000000000000000000000 (Dato      tipo I REG=4 CMD N=1 Comando NULL)
00100000000000000000000000000000 (NO OP      tipo I)
```

Cuadro de Código C.2: Script para analizar *bitstream*

Estas cadenas de texto son tomadas por InterpretaBitstream, quien las imprime en el archivo de salida. El código de esta rutina se encuentra en la Figura C.3.

```
function resultado = interpretaword(unaword)
%unaword = unaword(32:-1:1);
% WORDCOUNT define cuantas palabras de datos vienen a continuación. Es
% válido al haber detectado paquetes tipo I y II
global wordcount
% TIPOPAQUETE define si se ha detectado anteriormente un paquete de ese
% tipo.
global tipopaquete
% REGISTRO es el registro que se está direccionando por medio de un paquete
% tipo I
global registro
% SID y BID son los identificadores de los archivos de entrada y salida.
global sid bid
% FRAMEDIR es la dirección del frame que se está direccionando actualmente
global framedir

switch tipopaquete % Pregunto si anteriormente se ha detectado un paquete tipo I o II
    case 1 % Si antes se detectó un paquete tipo I entonces...
        analisisregistro = analizoregistro(unaword); % Que hace esta palabra en el registro?
        nombreregistro = identificaregistro(registro);
        analisis = [' (Dato      tipo I REG=' num2str(registro) ' ' nombreregistro ...
                                    ' N=' num2str(wordcount) ' ' analisisregistro ')'];
        wordcount = wordcount-1;
        if(wordcount == 0)
            tipopaquete = 0;
        end
    case 2 % Si antes se detectó un paquete tipo II
        if(mod(wordcount,41)==0)
            fprintf(sid,'%s TBloque=%d Top/Bot=%d Row=%d Major=%d Minor=%d',...
                '----------',framedir(1),framedir(2),framedir(3),framedir(4),framedir(5));
            fwrite(sid, sprintf('\n'), 'char');
        elseif(mod(wordcount,41)==40)
            framedir = frameautoinc(framedir);
        end
        analisis = [' (Dato      tipo II N=' num2str(wordcount) ')'];
        wordcount = wordcount-1;
        if(wordcount == 0)
            tipopaquete = 0;
        end
    otherwise
        if    (strcmp(unaword,['11111111' '11111111' '11111111' '11111111']))
```

```
            tipopaquete = 7;
            analisis = '␣(Dummy␣word)';
        elseif(strcmp(unaword,['00000000' '00000000' '00000000' '10111011']))
            tipopaquete = 0;
            analisis = '␣(Bus␣Width␣word)';
        elseif(strcmp(unaword,['10101010' '10011001' '01010101' '01100110']))
            tipopaquete = 5;
            analisis = '␣(Sync␣word)';
        elseif(strcmp(unaword,['00010001' '00100010' '00000000' '01000100']))
            tipopaquete = 0;
            analisis = '␣(8/16/32␣Bus␣width)';
        elseif(strcmp(unaword(1:3),'001'))      % Es un paquete tipo I
            registro = bin2dec(unaword(15:19));
            if(strcmp(unaword(4:5),'00'))           % Es el opcode NO OP
                analisis =  '␣(NO␣OP␣␣␣␣␣tipo␣I)';
            elseif(strcmp(unaword(4:5),'01'))           % Es una lectura
                wordcount=bin2dec(unaword(22:32));
                nombreregistro = identificaregistro(registro);
                analisis = ['␣(Lectura␣␣␣tipo␣I␣REG=' num2str(registro) '␣' nombreregistro ...
                                    '␣WCOUNT=' num2str(wordcount) ')'];
                tipopaquete = 1;
            elseif(strcmp(unaword(4:5),'10'))           % Es una escritura
                wordcount=bin2dec(unaword(22:32));
                nombreregistro = identificaregistro(registro);
                analisis = ['␣(Escritura␣tipo␣I␣REG=' num2str(registro) '␣' nombreregistro ...
                                    '␣WCOUNT=' num2str(wordcount) ')'];
                if (wordcount==0)
                    tipopaquete = 0;
                else
                    tipopaquete = 1;
                end
            elseif(strcmp(unaword(4:5),'11'))           % Es un código reservado
                analisis =  '␣(Reservado␣tipo␣I)';
            else
                analisis = '';
            end
        elseif(strcmp(unaword(1:3),'010'))      % Es un paquete tipo II
            tipopaquete = 2;
            wordcount=bin2dec(unaword(6:32));
            analisis=['␣(Cabecero␣␣tipo␣II␣para␣' num2str(bin2dec(unaword(6:32))) '␣palabras)'];
            %               TBloque Top/Bot Row Major Minor
            framedir = [    0       0     0    0     0];
        else
            tipopaquete = 0;
            analisis = '';
        end
end

resultado = [unaword analisis ];
```

Cuadro de Código C.3: Script para analizar *bitstream*

C.1.1.1 identificaregistro(direccion)

El objetivo de esta función es identificar el registro que se está escribiendo. La cadena de caracteres que devuelve corresponde al nombre del registro. Su código se presenta en la Figura C.4.

```
function nombreregistro = identificaregistro(direccion)

            switch direccion
                case 0    % CRC
                    nombreregistro = 'CRC';
                case 1    % FAR
                    nombreregistro = 'FAR';
                case 2    % FDRI
                    nombreregistro = 'FDRI';
                case 3    % FDRO
                    nombreregistro = 'FDRO';
                case 4    % CMD
                    nombreregistro = 'CMD';
```

```
                case 5    % CTL0
                    nombreregistro = 'CTL0';
                case 6    % MASK
                    nombreregistro = 'MASK';
                case 7    % STAT
                    nombreregistro = 'STAT';
                case 8    % LOUT
                    nombreregistro = 'LOUT';
                case 9    % COR0
                    nombreregistro = 'COR0';
                case 10   % MFWR
                    nombreregistro = 'MFWR';
                case 11   % CBC
                    nombreregistro = 'CBC';
                case 12   % IDCODE
                    nombreregistro = 'IDCODE';
                case 13   % AXSS
                    nombreregistro = 'AXSS';
                case 14   % COR1
                    nombreregistro = 'COR1';
                case 15   % CSOB
                    nombreregistro = 'CSOB';
                case 16   % WBSTAR
                    nombreregistro = 'WBSTAR';
                case 17   % TIMER
                    nombreregistro = 'TIMER';
                case 18   %
                    nombreregistro = 'No tiene nombre';
                case 19   %
                    nombreregistro = 'No tiene nombre';
                case 20   %
                    nombreregistro = 'No tiene nombre';
                case 21   %
                    nombreregistro = 'No tiene nombre';
                case 22   % BOOTSTS
                    nombreregistro = 'BOOTSTS';
                case 23   %
                    nombreregistro = 'No tiene nombre';
                case 24   % CTL1
                    nombreregistro = 'CTL1';
                otherwise %
                    nombreregistro = 'No tiene nombre';
            end
```

Cuadro de Código C.4: Script para analizar *bitstream*

C.1.1.2 analizoregistro(unaword)

Esta función es llamada por interpretaword cuando ha encontrado previamente un paquete tipo I. El objetivo es saber que efecto surge la escritura de un dato en un registro del ICAP. Su código se muestra en la Figura C.5.

```
function analisisregistro = analizoregistro(unaword);
%% WORDCOUNT define cuantas palabras de datos vienen a continuación. Es
%% válido al haber detectado paquetes tipo I y II
% global wordcount
%% TIPOPAQUETE define si se ha detectado anteriormente un paquete de ese
%% tipo.
% global tipopaquete
% REGISTRO es el registro que se está direccionando por medio de un paquete
% tipo I
global registro
        analisisregistro = '';
        switch registro
            case 0    % CRC
                analisisregistro = ['CRCtest=' unaword];
            case 1    % FAR
            case 2    % FDRI
                analisisregistro = ['Escribe este dato en la mem de configuración'];
```

```
            case 3    % FDRO
                analisisregistro = ['Este registro es de solo lectura'];
            case 4    % CMD
                switch bin2dec(unaword(28:32))
                    case 0 % NULL
                        analisisregistro = ['Comando NULL'];
                    case 1 % WCFG
                        analisisregistro = ['Escribe los datos de configuración ' ...
                            '(Se ejecuta previo a escribir sobre FDRI)'];
                    case 2 % MFW
                        analisisregistro = ['Usado para escribir sobre varios frames ' ...
                            'la misma info'];
                    case 3 % DGHIGH/LFRM
                        analisisregistro = ['Last Frame: No entendi el datasheet...'];
                    case 4 % RCFG
                        analisisregistro = ['Leer datos de configuración (Se ejecuta previo ' ...
                            'a una lectura por FDRO)'];
                    case 5 % START
                        analisisregistro = ['Inicia la secuencia de inicialización (Se debe ' ...
                            'haber hecho previamente CRC y DESYNC)'];
                    case 6 % RCAP
                        analisisregistro = ['Resetea la señal CAPTURE después de hacer una ' ...
                            'readbackcapture'];
                    case 7 % RCRC
                        analisisregistro = ['Resetea el registro CRC'];
                    case 8 % AGHIGH
                        analisisregistro = ['Activa la señal GHIGH_B, no entendí la ' ...
                            'explicación'];
                    case 9 % SWITCH
                        analisisregistro = ['Cambia la frecuencia de CCLK, según los bits ' ...
                            'OFSEL en COR0'];
                    case 10 % GRESTORE
                        analisisregistro = ['Pulsea la señal GRESTORE'];
                    otherwise
                        analisisregistro = ['Código inválido'];
                end
            case 5    % CTL0
            case 6    % MASK
                analisisregistro = ['Establece MASK=' unaword];
            case 7    % STAT
            case 8    % LOUT
                % IMPORTANTEE!!!!
                % Se está asumiendo que esto es una direccción de frame
                analisisregistro = interpretadirframe(unaword);
%                 analisisregistro = ['No se que signifique una escritura acá'];
            case 9    % COR0
            case 10   % MFWR
                analisisregistro = ['No se que signifique una escritura acá'];
            case 11   % CBC
                analisisregistro = ['No se que signifique una escritura acá'];
            case 12   % IDCODE
                analisisregistro = ['Se escribe el ID del dispositivo'];
            case 13   % AXSS
                analisisregistro = ['No se que signifique una escritura acá'];
            case 14   % COR1
            case 15   % CSOB
                analisisregistro = ['No se que signifique una escritura acá'];
            case 16   % WBSTAR
            case 17   % TIMER
            case 18   %
            case 19   %
            case 20   %
            case 21   %
            case 22   % BOOTSTS
            case 23   %
            case 24   % CTL1
            otherwise %
        end
```

Cuadro de Código C.5: Script para analizar *bitstream*

interpretadirframe(unaword) En caso de encontrarse que la palabra debe interpretarse como una dirección de un frame, esta función extrae el tipo de bloque, el direccionamiento de la fila, la columna y el minor address. Su código se muestra en la Figura C.6.

```
function analisisregistro = interpretadirframe(unaword)
analisisregistro = ['Tipo de Bloque = '  num2str(bin2dec(unaword(9:11)))  ...
                    ' Top/Bottom row = ' num2str(bin2dec(unaword(12)))  ...
                    ' Row Address = '    num2str(bin2dec(unaword(13:17)))...
                    ' Major Address = '  num2str(bin2dec(unaword(18:25)))...
                    ' Minor Address = '  num2str(bin2dec(unaword(26:32)))];
```

Cuadro de Código C.6: Script para analizar *bitstream*

C.1.1.3 frameautoinc(framedirin)

Debido a que los *bitstreams* generados por Xilinx® contienen por defecto la información de todos los frames en un solo paquete tipo II en el archivo de configuración, esto presume que los frames ordenados mediante unas reglas que no son claras por parte de Xilinx® pero que han sido estudiadas durante el desarrollo del proyecto. Este script permite saber cual es la siguiente dirección a partir de una dada. Su código se muestra en la Figura C.7.

```
function framedirout = frameautoinc(framedirin)
switch framedirin(1)                % TBloque
  case 0                              % CLB DSP BRAM IOB etc
    switch framedirin(2)                % Top/Bot
      case 0                              % Top (creo)
        switch framedirin(3)                % Row
          case {0, 1, 2}                      % Filas internas
            switch framedirin(4)                % Major(Columna)
              case {1,2,3,4, ...                  % CLB
                  6,7,8,9,10,11, ...
                  13,14,15,16,17,18, ...
                  20,21,22,23, ...
                  26,27,28,29, ...
                  31,32, 34,35, 37,38, ...
                  40,41,42,43, ...
                  45,46,47,48}
                if(framedirin(5)==35)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {0,24,44}                      % IOB
                if(framedirin(5)==53)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {33,36}                        % DSP
                if(framedirin(5)==27)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {5,12,19,30,39,49}             % BRAM
                if(framedirin(5)==29)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
```

```
            case {25}                          % ClockColumn
              if(framedirin(5)==3)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {50}                          % No se
              % He modificado esta comparación de 31 a 33 porque hay dos frames que están
              % descuadrando la alineación al final de cada row
              if(framedirin(5)==33)
                framedirout = [framedirin(1) framedirin(2) framedirin(3)+1 0              0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3)   framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end %switch
        case 3                             % Fila externa
          switch framedirin(4)               % Major(Columna)
            case {1,2,3,4, ...                 % CLB
                6,7,8,9,10,11, ...
                13,14,15,16,17,18, ...
                20,21,22,23, ...
                26,27,28,29, ...
                31,32, 34,35, 37,38, ...
                40,41,42,43, ...
                45,46,47,48}
              if(framedirin(5)==35)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {0,24,44}                     % IOB
              if(framedirin(5)==53)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {33,36}                       % DSP
              if(framedirin(5)==27)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {5,12,19,30,39,49}            % BRAM
              if(framedirin(5)==29)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {25}                          % ClockColumn
              if(framedirin(5)==3)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {50}                          % No se
              % He modificado esta comparación de 31 a 33 porque hay dos frames que están
              % descuadrando la alineación al final de cada row
              if(framedirin(5)==33)
                framedirout = [framedirin(1) framedirin(2)+1 0              0              0];
              else
                framedirout = [framedirin(1) framedirin(2)   framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end %switch
        otherwise
```

```
        end
      case 1                                % Bottom (creo)
        switch framedirin(3)                  % Row
          case {0, 1, 2}                        % Filas internas
            switch framedirin(4)                  % Major(Columna)
              case {1,2,3,4, ...                    % CLB
                  6,7,8,9,10,11, ...
                  13,14,15,16,17,18, ...
                  20,21,22,23, ...
                  26,27,28,29, ...
                  31,32, 34,35, 37,38, ...
                  40,41,42,43, ...
                  45,46,47,48}
                if(framedirin(5)==35)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {0,24,44}                        % IOB
                if(framedirin(5)==53)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {33,36}                          % DSP
                if(framedirin(5)==27)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {5,12,19,30,39,49}               % BRAM
                if(framedirin(5)==29)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {25}                             % ClockColumn
                if(framedirin(5)==3)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              case {50}                             % No se
                % He modificado esta comparación de 31 a 33 porque hay dos frames que están
                % descuadrando la alineación al final de cada row
                if(framedirin(5)==33)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3)+1 0              0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
              otherwise
            end
          case 3                              % Fila externa
            switch framedirin(4)                  % Major(Columna)
              case {1,2,3,4, ...                    % CLB
                  6,7,8,9,10,11, ...
                  13,14,15,16,17,18, ...
                  20,21,22,23, ...
                  26,27,28,29, ...
                  31,32, 34,35, 37,38, ...
                  40,41,42,43, ...
                  45,46,47,48}
                if(framedirin(5)==35)
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
                else
                  framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                                 framedirin(5)+1];
                end
```

```
            case {0,24,44}                    % IOB
              if(framedirin(5)==53)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {33,36}                      % DSP
              if(framedirin(5)==27)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {5,12,19,30,39,49}           % BRAM
              if(framedirin(5)==29)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {25}                         % ClockColumn
              if(framedirin(5)==3)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case {50}                         % No se
              % He modificado esta comparación de 31 a 33 porque hay dos frames que están
              % descuadrando la alineación al final de cada row
              if(framedirin(5)==33)
                framedirout = [framedirin(1)+1 0             0             0             0];
              else
                framedirout = [framedirin(1)   framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end
        otherwise
      end
    otherwise
  end
case 1 % no se...
  switch framedirin(2)              % Top/Bot
    case 0                            % Top (creo)
      switch framedirin(3)              % Row
        case {0, 1, 2}                    % Filas internas
          switch framedirin(4)              % Major(Columna)
            case {0,1,2,3,4}
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case 5                    % IOB
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2) framedirin(3)+1 0             0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3)   framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end
        case 3                            % Fila externa
          switch framedirin(4)              % Major(Columna)
            case {0,1,2,3,4}
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
```

```
            case 5                      % IOB
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2)+1 0              0              0];
              else
                framedirout = [framedirin(1) framedirin(2)   framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end
        otherwise
      end
    case 1                              % Bottom (creo)
      switch framedirin(3)                % Row
        case {0, 1, 2}                      % Filas internas
          switch framedirin(4)                % Major(Columna)
            case {0,1,2,3,4}
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case 5                      % IOB
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2) framedirin(3)+1 0              0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3)   framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end
        case 3                              % Fila externa
          switch framedirin(4)                % Major(Columna)
            case {0,1,2,3,4}
              if(framedirin(5)==127)
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4)+1 0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            case 5                      % IOB
              if(framedirin(5)==127)
                framedirout = [0              0              0              0              0];
              else
                framedirout = [framedirin(1) framedirin(2) framedirin(3) framedirin(4) ...
                               framedirin(5)+1];
              end
            otherwise
          end
        otherwise
      end
    otherwise
  end
otherwise
end

% %              TBloque Top/Bot         Row      Major    Minor
% framedirout = [    0        0       0     0        0];
```

Cuadro de Código C.7: Script para analizar *bitstream*

C.2 grep(archivo,patron,varargin)

Esta función realiza una función parcialmente similar a la que lleva a cabo grep en linux, el primér argumento es el archivo que llega de entrada, el segundo es el patrón que se buscará dentro del archivo y el tercer argumento es opcional, e indica un archivo donde se almacenan los resultados.

```
function grep(archivo,patron,varargin)
```

```
%% Verifica si hay un archivo especificado para la salida
if size(varargin,2)==1
    hayarchivodesalida = 1;
    archivodesalida = varargin{1};
else
    hayarchivodesalida = 0;
end
%% Abre los archivos necesarios
fid = fopen(archivo,'r');
if fid<0
    % Digo que hubo error
    fprintf('Error al abrir el archivo de entrada\n');
    error('El archivo no se pudo abrir.')
else
    % Digo que el archivo abrió bien
    fprintf('Abrí el archivo de entrada con fid=%d\n',fid);
end
if(hayarchivodesalida)
    sid = fopen(archivodesalida,'w+');
    if sid<0
        % Digo que hubo error
        fprintf('Error al abrir el archivo de entrada\n');
        error('El archivo no se pudo abrir.')
    else
        % Digo que el archivo abrió bien
        fprintf('Abrí el archivo de entrada con sid=%d\n',sid);
    end
end
%% Comparación del patrón
lpatron  = length(patron);
tline = fgetl(fid);
while(ischar(tline))
    ltline  = length(tline);
    if(lpatron<=ltline)
        for i=1:(ltline-lpatron)
            if(compara(patron,tline(i:(i+lpatron-1))))
                if(hayarchivodesalida)
                    fprintf(sid,'%s',tline);
                    fwrite(sid, sprintf('\n'), 'char');
                else
                    disp(tline)
                end
                break
            end
        end
    end
    tline = fgetl(fid);
end
%% Cierra los archivos
fclose(fid);
if(hayarchivodesalida)
    fclose(sid);
end
% Digo que el archivo abrió bien
fprintf('Cerré los archivos\n');
```

Cuadro de Código C.8: Script para analizar *bitstream*

C.3 analizogrep(archivo)

Debido a la gran cantidad de frames que componen un *bitstream*, esta función permite analizar el resultado que arroja grep y agrupar todos los frames cuyas direcciones son idénticas excepto en el *minor address*.

```
function analizogrep(archivo)
fid = fopen(archivo,'r');
if fid<0
    % Digo que hubo error
    fprintf('Error al abrir el archivo de entrada\n');
    error('El archivo no se pudo abrir.')
```

```
else
    % Digo que el archivo abrió bien
    fprintf('Abrí el archivo de entrada con fid=%d\n',fid);
end
sid = fopen([archivo,'.analizado'],'w+');
if sid<0
    % Digo que hubo error
    fprintf('Error al abrir el archivo de entrada\n');
    error('El archivo no se pudo abrir.')
else
    % Digo que el archivo abrió bien
    fprintf('Abrí el archivo de entrada con sid=%d\n',sid);
end

formato = ['%s ' '(Dato      tipo I REG=8 LOUT N=1 '...
           'Tipo de Bloque = '   '%d' ...
           ' Top/Bottom row = '  '%d' ...
           ' Row Address = '     '%d' ...
           ' Major Address = '   '%d' ...
           ' Minor Address = '   '%d)' ];

A = fscanf(fid,formato);

datos = zeros([length(A)/37 5]);
bits = zeros([length(A)/37 32]);
for i=1:(length(A)/37)
    datos(i,1:5)=A((i-1)*37+(33:37));
    bits(i,:)=char(A((i-1)*37+(1:32)));
end

fprintf(sid,'%s\t%s\t%s\t%s\t%s','TBloque','Top/Bot','Row','Major','Minor');
fwrite(sid, sprintf('\n'), 'char');

anterior = datos(1,:);
cabezadeserie = datos(1,:);
for i=2:size(datos,1)
    actual = datos(i,:);
    if(actual(5)~=(anterior(5)+1))
        fprintf(sid,'%d\t%d\t%d\t%d\t%d-%d',cabezadeserie(1),cabezadeserie(2),cabezadeserie(3),\
cabezadeserie(4),cabezadeserie(5),anterior(5));
        switch anterior(5)
            case 35
                fprintf(sid,'\t CLB');
            case 27
                fprintf(sid,'\tDSP');
            case 29
                fprintf(sid,'\t   BRAM');
            case 53
                fprintf(sid,'\t  IOB');
            case 3
                fprintf(sid,'\t Clock Column');
            otherwise
                fprintf(sid,'\t No se');
        end
        fwrite(sid, sprintf('\n'), 'char');
        cabezadeserie = actual;
    end
    anterior = actual;
end
fclose(fid);
fclose(sid);
```

Cuadro de Código C.9: Script para analizar *bitstream*

Apéndice D

Procedimiento para compilar el *driver* de EDK sobre el sistema operativo.

En este Anexo, se resumen los cambios que se realizaron sobre el *driver* del periférico `xps_hwicap` para que se pudiera compilar en Linux sin mayores modificaciones. Esto con el objetivo de que esta guía pueda servir de apoyo cuando se requiera realizar operaciones similares.

D.1 Modificaciones sobre los archivos del *driver* `xhwicap`

No se realizó ninguna modificación sobre los archivos del *driver* del periférico `xhwicap`. Este era el objetivo inicial, en busca de que con estas adaptaciones cualquier *driver* de EDK pueda ser traido a Linux directamente.

D.2 Modificaciones sobre los archivos de ejemplo

Sobre los archivos `xhwicap_low_level_example.c`, `xhwicap_lut.c`, `xhwicap_read_config_reg.c` y `xhwicap_read_frame_polled_example.c` se comentó la línea que indica que se va a hacer uso del `xil_printf` en vez del `printf` como se muestra en el código D.1.

```
//#define printf  xil_printf          /* A smaller footprint printf */
```

Cuadro de Código D.1: Definición que se eliminó para emplear la función `printf` estándar

Adicionalmente en algunos archivos se imprimió información relevante con el objetivo de hacer depuración, pero evidentemente este paso no es obligatorio para el funcionamiento del *driver*.

Tabla D.1: Archivos traidos desde EDK para la compilación de las aplicaciones que usan el periférico xps_hwicap

Archivo	Descripción
`ppc-asm.h`	Este archivo contiene declaraciones básicas del procesador PowerPC®y algunos macros.
`xbasic_types.c`	Contiene funciones básicas empleadas en los *driver* de Xilinx® .
`xbasic_types.h`	Contiene definiciones básicas empleadas en los *driver* de Xilinx® .
`xhwicap_family.c`	Identifica la familia de FPGA que se está trabajando. Para este caso fue 3.
`xil_assert.c`	Definición de las funciones básicas usadas en los assert de los *drivers* de Xilinx®
`xil_assert.h`	Contiene algunas definiciones sobre los assert de los *drivers* de Xilinx® .
`xil_exception.h`	Contiene funciones relacionadas con excepeciones para los *driver* de Xilinx® .
`xil_io.c`	Contiene las rutinas de entrada salida para hacer el acceso desde el procesador a los registros del periférico. Este archivo ha sido severamente modificado, sustituyendo las rutinas por métodos que permiten acceder al mapa de memoria del sistema desde el espacio del usuario por medio de el nodo `/dev/mem`.
`xil_io.h`	Contiene la declaración de los cabeceros de las funciones modificadas en `xil_io.c`. Este archivo no fue modificado.

Continua en la siguiente página

Tabla D.1 – continuación desde la página anterior

Archivo	Descripción
`xil_types.h`	Contiene la declaración de algunos tipos de datos empleados por los *drivers* de Xilinx® .
`xintc.h`	Contiene rutinas para el manejo de interrupciones con prioridades.
`xintc_l.h`	Contiene las rutinas de bajo nivel para llevar a cabo el manejo de las instrucciones.
`xio.c`	Igual que xil_io.c
`xio.h`	Igual que xil_io.h
`xpseudo_asm.h`	Incluye el archivo `xpseudo_asm_gcc.h`
`xpseudo_asm_gcc.h`	Define macros para assembler.
`xreg440.h`	Realiza algunas definiciones para ser empleadas en assembler relacionadas con la organización del procesador PowerPC®440.
`xstatus.h`	Define códigos relacionados con el estado actual de algún periférico o de alguna operación realizada sobre ellos.

D.3 `Makefile` para compilar las fuentes y los ejemplos.

Con el objetivo de compilar ordenadamente las fuentes de EDK, las del *driver*, los ejemplos y las aplicaciones de prueba, se creó el makefile mostrado en el listado D.2

```
# DESCRIPCIÓN
#
# Este makefile se encarga inicialmente de compilar la librería que manipula
# el periférico como un driver desde el espacio del usuario. Posteriormente
# es posible compilar los ejemplos que tienen compatibilidad con la tarjeta
# ML507 y que venían con el driver de xilinx. Y así mismo se pueden compilar
# las modificaciones a los ejemplos.

# Programas empleados
COMPILER := gcc
ARCHIVER := ar
```

```
# Nombres de las librerías standalone
#XTYPES := xbasic_types
XTYPES := xil_assert
#XIOLIB := xio
XIOLIB := xil_io

# Path donde se está desarrollando la librería
DIRINSTALACION := $(PWD)
DIRSRC := hwicap_v4_00_a/src
DIREXA := hwicap_v4_00_a/examples
DIROBJ := Objects
DIRLIB := LibsGen
DIREXEC := Ejecutables

vpath %.c $(DIRINSTALACION)/$(DIRSRC)
vpath %.c $(DIRINSTALACION)/xinclude

# Opciones de DEBUG
DEBUGS :=
DEBUGS += -D XLUT_DEBUG
#DEBUGS += -D DEBUG_XIL_IO


# Declaraciones de macros externas para compatibilidad
DEFINICIONMACRO := -D XPAR_OPB_HWICAP_0_DEVICE_ID=XPAR_XPS_HWICAP_0_DEVICE_ID \
                   -D XPAR_OPB_HWICAP_0_BASEADDR=XPAR_XPS_HWICAP_0_BASEADDR \
                   $(DEBUGS)
#                   -D XPAR_XHWICAP_NUM_INSTANCES=XPAR_XHWICAP_NUM_INSTANCES \



# Definición del modo de las librerías (puede ser cambiado por línea
# de comandos al invocar al make).
MODO := static

# Banderas del compilador según el modo de las librerias
ifeq ($(MODO),static)
CFLAGS := -c
else
ifeq ($(MODO),shared)
CFLAGS := -c -fPIC
endif
endif


# Archivo principal
all:  ejemplos operaLUTs
	@echo "MAKE: Info: Terminé de compilar todo todito todo!"


operaLUTs: prueba_LUTs/operaLUTs/operaLUTs prueba_LUTs/operaLUTs/operaLUTs_verbose

prueba_LUTs/operaLUTs/operaLUTs: prueba_LUTs/operaLUTs/operaLUTs.c $(DIRLIB)/lib$(XIOLIB).a \
    $(DIRLIB)/libxhwicap.a $(DIRLIB)/libxhwicap.a $(DIRLIB)/lib$(XTYPES).a $(DIREXEC)
	@echo "MAKE: Info: Linkeando operaLUTs.c y generando operaLUTs"
	@$(COMPILER) -static prueba_LUTs/operaLUTs/operaLUTs.c -I$(DIRINSTALACION)/$(DIRSRC) \
	    -I$(DIRINSTALACION)/xinclude -I$(DIRINSTALACION) -L$(DIRINSTALACION)/$(DIRLIB) \
	    -lxhwicap -l$(XIOLIB) -l$(XTYPES) -o $@

prueba_LUTs/operaLUTs/operaLUTs_verbose: prueba_LUTs/operaLUTs/operaLUTs.c \
    $(DIRLIB)/lib$(XIOLIB).a $(DIRLIB)/libxhwicap.a $(DIRLIB)/libxhwicap.a \
    $(DIRLIB)/lib$(XTYPES).a $(DIREXEC)
	@echo "MAKE: Info: Linkeando operaLUTs.c y generando operaLUTs_verbose"
	@$(COMPILER) -static -D IMPRIME_TABLA_DE_VERDAD_DE_CADA_LUT \
	    prueba_LUTs/operaLUTs/operaLUTs.c -I$(DIRINSTALACION)/$(DIRSRC) \
	    -I$(DIRINSTALACION)/xinclude -I$(DIRINSTALACION) \
	    -L$(DIRINSTALACION)/$(DIRLIB) -lxhwicap -l$(XIOLIB) -l$(XTYPES) -o $@




ejemplos:  ejemplo3 ejemplo4 ejemplo5 ejemplo6 ejemplo6.2 ejemplo7 test1 test2
```

```
        @echo "MAKE: Info: Terminé la compilación de los archivos de ejemplo"

# Listado de ejemplos de uso del driver
ejemplo1: $(DIREXEC)/xhiwcap_intr_example
ejemplo2: $(DIREXEC)/xhwicap_ff
ejemplo3: $(DIREXEC)/xhwicap_low_level_example
ejemplo4: $(DIREXEC)/xhwicap_lut
ejemplo5: $(DIREXEC)/xhwicap_read_config_reg
ejemplo6: $(DIREXEC)/xhwicap_read_frame_polled_example
ejemplo6.2: $(DIREXEC)/xhwicap_read_frame_polled_example_args
ejemplo7: $(DIREXEC)/xhwicap_testapp_example
test1: $(DIREXEC)/w_lee_y_escribe_frame
test2: $(DIREXEC)/w_escribe_una_LUT

# Enlace de los ejecutables creados en los ejemplos del driver.
$(DIREXEC)/%: $(DIRINSTALACION)/$(DIREXA)/%.c $(DIRLIB)/lib$(XIOLIB).a \
    $(DIRLIB)/libxhwicap.a $(DIRLIB)/libxhwicap.a $(DIRLIB)/lib$(XTYPES).a \
    $(DIREXEC)
        @echo "MAKE: Info: Linkeando $@.c de forma estática"
        @$(COMPILER) -static $< -I$(DIRINSTALACION)/$(DIRSRC) -I$(DIRINSTALACION)/xinclude \
            -I$(DIRINSTALACION) -L$(DIRINSTALACION)/$(DIRLIB) -lxhwicap \
            -l$(XIOLIB) -l$(XTYPES) -o $@


# Compilación de la librería de entrada/salida
$(DIRLIB)/libxil_io.a: $(DIROBJ)/xil_io.o $(DIRLIB)
        @echo "MAKE: Info: Creando la librería estática libxil_io.a"
        @$(ARCHIVER) rcs $(DIRLIB)/libxil_io.a $(DIROBJ)/xil_io.o

# Compilación de la librería de xilinx de tipos de datos
$(DIRLIB)/lib$(XTYPES).a: $(DIROBJ)/$(XTYPES).o $(DIRLIB)
        @echo "MAKE: Info: Creando la librería estática lib$(XTYPES).a"
        @$(ARCHIVER) rcs $(DIRLIB)/lib$(XTYPES).a $(DIROBJ)/$(XTYPES).o

# Compilación de la librería de entrada/salida
$(DIRLIB)/libxio.a: $(DIROBJ)/xio.o $(DIRLIB)
        @echo "MAKE: Info: Creando la librería estática libxio.a"
        @$(ARCHIVER) rcs $(DIRLIB)/libxio.a $(DIROBJ)/xio.o

# Creación de la librería del hwicap
$(DIRLIB)/libxhwicap.a: $(DIROBJ)/xhwicap.o $(DIROBJ)/xhwicap_device_read_frame.o \
    $(DIROBJ)/xhwicap_device_write_frame.o $(DIROBJ)/xhwicap_intr.o \
    $(DIROBJ)/xhwicap_selftest.o $(DIROBJ)/xhwicap_sinit.o $(DIROBJ)/xhwicap_g.o \
    $(DIROBJ)/xhwicap_srp.o $(DIRINSTALACION)/$(DIRSRC)/xhwicap_set_clb_bits_ppc_v5.o \
    $(DIRINSTALACION)/$(DIRSRC)/xhwicap_get_clb_bits_ppc_v5.o $(DIRLIB)
        @echo "MAKE: Info: Creando la librería estática libxhwicap.a"
        @$(ARCHIVER) rcs $(DIRLIB)/libxhwicap.a $(DIROBJ)/xhwicap.o \
            $(DIROBJ)/xhwicap_device_read_frame.o $(DIROBJ)/xhwicap_device_write_frame.o \
            $(DIROBJ)/xhwicap_intr.o $(DIROBJ)/xhwicap_selftest.o $(DIROBJ)/xhwicap_sinit.o \
            $(DIROBJ)/xhwicap_g.o $(DIROBJ)/xhwicap_srp.o \
            $(DIRINSTALACION)/$(DIRSRC)/xhwicap_set_clb_bits_ppc_v5.o \
            $(DIRINSTALACION)/$(DIRSRC)/xhwicap_get_clb_bits_ppc_v5.o




# Creación de los archivos objeto
$(DIROBJ)/%.o: %.c $(DIROBJ)
#%.o: %.c
        @echo "MAKE: Info: Creando el archivo $@"
        @$(COMPILER) $(CFLAGS) $(DEFINICIONMACRO) -I$(DIRINSTALACION)/xinclude \
            -L$(DIRINSTALACION)/xinclude -I$(DIRINSTALACION)    $< -o $@


# Creación de los directorios de compilación
$(DIROBJ):
        @echo "MAKE: Info: Creando el directorio $(DIROBJ)"
        @mkdir $(DIROBJ)

$(DIRLIB):
        @echo "MAKE: Info: Creando el directorio $(DIRLIB)"
        @mkdir $(DIRLIB)

$(DIREXEC):
```

```
	@echo "MAKE: Info: Creando el directorio $(DIREXEC)"
	@mkdir $(DIREXEC)


# Borra los archivos generados
clean:
	@echo "MAKE: Info: Borrando todos los archivos generados"
	@rm -f $(DIRLIB)/*.a $(DIROBJ)/*.o $(DIREXEC)/*
	@rm -r -f $(DIRLIB) $(DIROBJ) $(DIREXEC)
```

Cuadro de Código D.2: Makefile que compila las fuentes traidas de Xilinx® , los ejemplos del hwicap y los programas generados. Todo esto basado en la Figura 4.8

Apéndice E

Código de la librería `xil_io` (versión espacio del usuario).

En este Anexo se presenta el código que sustituyó la librería de Xilinx® `xil_io`. Esta librería tiene como propósito escribir y leer un dato sobre cierta posición de memoria. Desde el entorno de EDK es suficiente con el manejo de un puntero que apunte hacia esta posición de memoria; pero desde el sistema operativo es necesario solicitar el acceso a esta posición de memoria y llevar a cabo un `remap` para obtener un puntero que llegue a la posición de memoria física solicitada apartir de una dirección de memoria virtual.

E.1 Archivo `xil_io.c`

```
/*****************************************************************************
 * DESCRIPCION:                                                              *
 *                                                                           *
 * Este programa define dos rutinas para leer y escribir sobre una direccion *
 * física en el sistema. Está basado en el uso del nodo /dev/mem, y puede    *
 * ser empleado en las pruebas de un periférico, pero su desempeño será      *
 * inferior al de un driver diseñado en el espacio del kernel.               *
 *                                                                           *
 * Esta fuente está  basada en la rutina desarrollada para el manejo del LCD *
 * desde el espacio del usuario.                                             *
 *****************************************************************************/

// Entradas y salidas estandar
#include <stdio.h>

// Soporte para usleep
#include <time.h>

// stdlib....
#include <stdlib.h>

// Soporte para abrir y modificar los archivos como /dev/mem
#include <fcntl.h>


#include <sys/mman.h>

#define MAP_SIZE 4096UL
#define MAP_MASK (MAP_SIZE - 1)


#include "xil_io.h"

//#define DEBUG_XIL_IO
```

```
void Xil_Out32(u32 OutAddress,u32 Value)
{
  int memfd;
  void *mapped_base, *mapped_dev_base;
  off_t dev_base = OutAddress;
#ifdef DEBUG_XIL_IO
  printf("Xil_Out32:\t Info:\t Iniciando Xil_Out32(%p,0x%.8X)\n",OutAddress,Value);
#endif
  memfd = open("/dev/mem", O_RDWR | O_SYNC);
  if (memfd == -1) {
    printf("Can't open /dev/mem.\n");
    exit(0);
  }

// Ejecuta mmap para acceder a una dirección de memoria física desde una \
   dirección virtual. Esto mapeará una pagina pero no garantiza que la \
   posición de memoría solicitada esté al comienzo de la misma
  mapped_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, memfd, dev_base & ~MAP_MASK);

  if (mapped_base == (void *) -1) {
    printf("Can't map the memory to user space.\n");
    exit(0);
  }

// Agregando el offset, se llega a la posición de memoria que finalmente será escrita
  mapped_dev_base = mapped_base + (dev_base & MAP_MASK);

  *((unsigned long *) (mapped_dev_base )) = Value;
#ifdef DEBUG_XIL_IO
  printf("Escribí sobre %p el dato %8.8X \n",mapped_dev_base,Value);
#endif
  usleep(1000);

// unmap the memory before exiting
  if (munmap(mapped_base, MAP_SIZE) == -1) {
    printf("Can't unmap memory from user space.\n");
    exit(0);
  }

  close(memfd);

}

u32 Xil_In32(u32 InAddress)
{
  int memfd;
  void *mapped_base, *mapped_dev_base;
  off_t dev_base = InAddress;
  u32 resultado;
#ifdef DEBUG_XIL_IO
  printf("Xil_In32:\t Info:\t Iniciando Xil_In32(%.8p)\n",InAddress);
#endif
  memfd = open("/dev/mem", O_RDWR | O_SYNC);
  if (memfd == -1) {
    printf("Can't open /dev/mem.\n");
    exit(0);
  }

// Ejecuta mmap para acceder a una dirección de memoria física desde una \
   dirección virtual. Esto mapeará una pagina pero no garantiza que la \
   posición de memoría solicitada esté al comienzo de la misma
  mapped_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, memfd, dev_base & ~MAP_MASK);

  if (mapped_base == (void *) -1) {
    printf("Can't map the memory to user space.\n");
    exit(0);
  }

// Agregando el offset, se llega a la posición de memoria que finalmente será leida
  mapped_dev_base = mapped_base + (dev_base & MAP_MASK);

  resultado = *((u32 *) (mapped_dev_base ));
#ifdef DEBUG_XIL_IO
```

```
    printf("Leí de %p el dato %8.8X \n",mapped_dev_base,resultado);
#endif
    usleep(1000);


// unmap the memory before exiting
    if (munmap(mapped_base, MAP_SIZE) == -1) {
        printf("Can't unmap memory from user space.\n");
        exit(0);
    }

    close(memfd);

    return resultado;

}
```

Cuadro de Código E.1: Nueva librería xil_io: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.

Apéndice F

Programas y scripts creados durante la compilación del *driver* desde el espacio del usuario.

En este Anexo se presenta el código fuente de los programas y scripts desarrollados durante la primera etapa de la compilación en Linux del *driver* para el periférico xps_hwicap. En esta etapa se empleó la plantilla para acceder a un periférico desde el espacio del usuario sin necesidad de añadir un módulo al *kernel* de Linux.

F.1 xhwicap_read_frame_polled_example_args.c

El propósito de este programa es ampliar las opciones del programa de ejemplo del *driver* de Xilinx® llamado `xhwicap_read_frame_polled_example.c`. El listado completo de opciones se muestra en la Tabla F.1 y su código fuente se muestra en el listado F.1.

Tabla F.1: Opciones de visualización de xhwicap_read_frame_polled_example_args.c

Opción	Descripción
`--debug`	Imprime cabeceros al inicio de cada frame que muestran los parámetros de su dirección como el tipo de bloque, top, fila, major y minor. Ejemplo: `FRAME=> Block=0 Top=1 Hclk/Row=1 Major=1 Minor=20`
`--hex`	Imprime en una columna la información del frame en hexadecimal. Ejemplo: `21000303`
`--bin`	Imprime en una columna la información del frame en binario. Ejemplo:

Continua en la siguiente página

Tabla F.1 – continuación desde la página anterior

Opción	Descripción
	00100001000000000000001100000011
`--info`	Imprime en frente de cada palabra del frame su índice. Ejemplo: `Frame Word 70 -> 21000303`
`--quit_null_frame`	No imprime un frame adicional que es leido siempre que se hace la lectura de un frame. Este frame es cero en su totalidad y no contiene información.
`--top`	Indica si el frame está en la parte superior (cero) o si está en la parte inferior (uno) del FPGA.
`--block`	Indica que tipo de frame se desea leer: cero para conexión y configuración de recursos lógicos y uno para el contenido de los bloques de memoria RAM.
`--hclk`	Indica en cual fila del FPGA está ubicado el frame que se va a leer. Su numeración comienza desde la parte central hacia los extremos.
`--major`	Indica en cuál columna del FPGA está ubicado el frame que se desea leer. Cuando block es cero, su numeración es correspondiente con la de la Tabla 3.3.
`--minor`	Indica cuál es el minor address del frame que se desea leer.

F.1.1 Ejemplos de uso

Para leer el último frame de la columna 16, de la fila 2 de la parte inferior del FPGA, eliminando el frame nulo inicial, imprimiendo los datos en formato hexadecimal y binario e imprimiendo un cabecero con la información del frame:

```
192# ./Ejecutables/xhwicap_read_frame_polled_example_args \
                --quit_null_frame --hex --bin --debug --top=1 \
                --block=0 --hclk=2 --major=16 --minor=35
```

F.1.2 Código fuente

```
/*****************************************************************************
 * DESCRIPCIÓN                                                               *
 * Esta es una variación del ejemplo xhwicap_read_frame_polled_example.c     *
 * que viene con el driver de Xilinx. La modificación realizada permite      *
 * modificar la forma en la que se imprime el frame y permite acceder a un   *
 * frame en específico.                                                      *
 *****************************************************************************/

/**************************** Include Files *********************************/

#include <xparameters.h>
#include <xil_types.h>
#include <xil_assert.h>
```

```
#include <xhwicap.h>
#include <stdio.h>
#include <getopt.h>

/************************** Constant Definitions *****************************/

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define HWICAP_DEVICEID          XPAR_HWICAP_0_DEVICE_ID


/*
 * These are the parameters for reading a frame of data in
 * the slice SLICE_X0Y0
 */
#define HWICAP_EXAMPLE_TOP              1
#define HWICAP_EXAMPLE_BLOCK            0
#define HWICAP_EXAMPLE_HCLK             1
#define HWICAP_EXAMPLE_MAJOR            1
#define HWICAP_EXAMPLE_MINOR            20

//#define DEBUG_ARGS_INPUT
//#define DEBUG_DESCRIPTION

//#define printf  xil_printf    /* A smaller footprint printf */


/************************** Function Prototypes ******************************/
int HwIcapReadFramePolledExample(u16 DeviceId,int frame_top,int frame_block,int frame_hclk,\
    int frame_major,int frame_minor,int debug,int hex,int bin,int info,int quit_null_frame);

/************************** Variable Definitions *****************************/
static XHwIcap HwIcap;



/* Función auxiliar para imprimir en binario*/
char *  int2bin(u32 dec, char * bin) {
    int j;
//    char salida[32];
    for(j=0;j<32;j++) {
        bin[31-j]=(dec%2)?'1':'0';
        dec = dec/2;
    }
    return bin;
}

/* Función principal donde se reciben los argumentos y se llama la función del ejemplo. */
int main(int argc, char **argv)
{
        int Status;
// Lee los argumentos de entrada
    char* desc;
    int c;
    int digit_optind = 0;
// Variables que se pueden recibir por línea de comandos
    int frame_top   = HWICAP_EXAMPLE_TOP  ;
    int frame_block = HWICAP_EXAMPLE_BLOCK;
    int frame_hclk  = HWICAP_EXAMPLE_HCLK ;
    int frame_major = HWICAP_EXAMPLE_MAJOR;
    int frame_minor = HWICAP_EXAMPLE_MINOR;
    int debug       = 0;
    int hex         = 0;
    int bin         = 0;
    int info        = 0;
    int quit_null_frame  = 0;

    unsigned long direccion = 0;
    int cantidad = 0;
    static struct option long_options[] = {
        {"top", required_argument, 0, 0},
```

```
        {"block", required_argument, 0, 0},
        {"hclk", required_argument, 0, 0},
        {"major", required_argument, 0, 0},
        {"minor", required_argument, 0, 0},
        {"debug", no_argument, 0, 0},
        {"hex", no_argument, 0, 0},
        {"bin", no_argument, 0, 0},
        {"info", no_argument, 0, 0},
        {"quit_null_frame", no_argument, 0, 0},
        {NULL, 0, NULL, 0}
    };
    int option_index = 0;
    while ((c = getopt_long(argc, argv, "",
                 long_options, &option_index)) != -1) {
        int this_option_optind = optind ? optind : 1;
        switch (c) {
        case 0:
#ifdef DEBUG_ARGS_INPUT
            printf ("option %s", long_options[option_index].name);
#endif
            if(strcmp("top",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_top = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Top Field: %d",frame_top);
#endif
                }
            if(strcmp("block",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_block = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    switch (frame_top) {
                    case 0:
                        desc="Interconnect and block configuration";
                        break;
                    case 1:
                        desc="Block RAM contents";
                        break;
                    default:
                        desc="no se...";
                    }
                    printf ("Block Field: %d (%s)",frame_block,desc);
#endif
                }
            if(strcmp("hclk",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_hclk = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("hclk Field: %d",frame_hclk);
#endif
                }
            if(strcmp("major",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_major = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Major Field: %d",frame_major);
#endif
                }
            if(strcmp("minor",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
```

```
#endif
                    frame_minor = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Minor Field: %d",frame_minor);
#endif
                }
            if(strcmp("debug",long_options[option_index].name)==0)
              debug=1;
            if(strcmp("hex",long_options[option_index].name)==0)
              hex=1;
            if(strcmp("bin",long_options[option_index].name)==0)
              bin=1;
            if(strcmp("info",long_options[option_index].name)==0)
              info=1;
            if(strcmp("quit_null_frame",long_options[option_index].name)==0)
              quit_null_frame=42;
#ifdef DEBUG_DESCRIPTION
              printf ("\n");
#endif
            break;
        case '?':
            break;
        default:
            printf ("?? getopt returned character code 0%o ??\n", c);
        }
    }

/* Llamado a la función del ejemplo. */
        Status = HwIcapReadFramePolledExample(HWICAP_DEVICEID,frame_top,frame_block,\
            frame_hclk,frame_major,frame_minor,debug,hex,bin,info,quit_null_frame);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        return XST_SUCCESS;

}


/* Función de ejemplo del driver de Xilinx*/
int HwIcapReadFramePolledExample(u16 DeviceId,int frame_top,int frame_block,int frame_hclk,\
    int frame_major,int frame_minor,int debug,int hex,int bin, int info,int quit_null_frame)
{
        int Status;
        u32 Index;
        XHwIcap_Config *CfgPtr;
        u32  FrameData[XHI_NUM_WORDS_FRAME_INCL_NULL_FRAME];

        char dummychar[33];

        /*
         * Initialize the HwIcap instance.
         */
        CfgPtr = XHwIcap_LookupConfig(DeviceId);
        if (CfgPtr == NULL) {
                return XST_FAILURE;
        }

        Status = XHwIcap_CfgInitialize(&HwIcap, CfgPtr, CfgPtr->BaseAddress);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        /*
         * Perform a self-test to ensure that the hardware was built correctly.
         */
        Status = XHwIcap_SelfTest(&HwIcap);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        /*
         * Read the Frame
         */
```

```
        #if ((XHI_FAMILY == XHI_DEV_FAMILY_V4) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V5) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V6))

        Status = XHwIcap_DeviceReadFrame(&HwIcap,
                                         frame_top,
                                         frame_block,
                                         frame_hclk,
                                         frame_major,
                                         frame_minor,
                                         (u32 *) &FrameData[0]);

        if (Status != XST_SUCCESS) {
                printf("Failed to Read Frame: %d \n\r", Status);
                return XST_FAILURE;
        }
        #endif

        /*
         * Print Frame contents
         */
        dummychar[32]=0;
        if (debug)
                printf("FRAME=> Block=%d Top=%d Hclk/Row=%d Major=%d Minor=%d\n",frame_block,\
                    frame_top,frame_hclk,frame_major,frame_minor);
        for (Index = quit_null_frame;
                Index <= (XHI_NUM_FRAME_WORDS << 1) ; Index++) {
                if (info)
                        printf("Frame Word %d -> \t",
                                Index );
                if (hex)
                        printf(                    "%8.8X ",
                                FrameData[Index] );
                if (bin)
                        printf(                                "%s",
                                                int2bin(FrameData[Index],&dummychar[0]));
                if ((info+hex+bin)!=0)
                        printf("\n");

        }

        return  XST_SUCCESS;
}
```

Cuadro de Código F.1: Código fuente del programa `xhwicap_read_frame_polled_example_args.c`.

F.2 leer_frames_de_una_columna.sh

Este script usa el programa `xhwicap_read_frame_polled_example_args` para leer e imprimir en consola los frames de una columna de CLBs. Para seleccionar una columna en particular se puede editar su código fuente en el cabecero, así mismo se pueden modificar las opciones de visualización. Un ejemplo de uso se muestra a continuación y su código fuente se muestra en el listado F.2.

F.2.1 Ejemplo de uso

```
192# cd scripts_de_prueba
192# ./leer_frames_de_una_columna.sh
```

F.2.2 Código fuente

```
# DESCRIPCIÓN
#
# Este script hace uso de la función xhwicap_read_frame_polled_example_args
```

```
# para leer todos los frames de una columna definida por los siguientes
# parámetros:
#
# TOP: pregunta si la fila leida está ubicada en la parte superior o inferior
#      del FPGA
TOP=1
# BLOCK:
BLOCK=0
# HCLK: Hace referencia al número de la fila siendo cero la más cercana al
#       centro del FPGA
HCLK=2
# MAJOR: Mediante este parámetro se especifica la columna dentro de la fila
MAJOR=16
# Adicionalmente mediante las siguientes opciones se controla la forma en
# la que la función imprime la información.
OPCIONES="--quit_null_frame --hex --bin --debug"

if [ -x ../Ejecutables/xhwicap_read_frame_polled_example_args ]; then
  for ((minor=0;minor<36;minor++))
  do
    ../Ejecutables/xhwicap_read_frame_polled_example_args ${OPCIONES} --top=${TOP} \
                  --block=${BLOCK} --hclk=${HCLK} --major=${MAJOR} --minor=${minor}
  done
else
  echo "ERROR: Probablemente no ha ejecutado satisfactoriamente make"
fi
```

Cuadro de Código F.2: Código fuente del script `leer_frames_de_una_columna.sh`.

F.3 leer_toda_la_configuracion.sh

Este script hace un intento de leer todo el archivo de configuración del FPGA mientras el sistema base corre el sistema operativo. Esta tarea no siempre se logra satisfactoriamente debido a que la lectura de algunos frames hace que el sistema se bloquee. Este script puede ser modificado para reiniciar la lectura en el frame que se bloqueó, pero debido a que no era fundamental para el desarrollo del proyecto, esto no se hizo. Su código se muestra en el listado F.3 y un ejemplo de uso se muestra a continuación.

F.3.1 Ejemplo de uso

```
192# cd scripts_de_prueba
192# ./leer_toda_la_configuracion.sh
```

F.3.2 Código fuente

```
#!/bin/bash
#set -x
#
# DESCRIPCIÓN
#
# Mediante este script se pretende leer todo el archivo de configuración del
# FPGA y para ello se recorren las direcciones de la misma forma en la que se
# organizan los frames en el bitstream que genera ISE.

# Vector que contiene la información de la organización del FPGA
num_minors=(54 36 36 36 36 30 36 36 36 36 36 36 30 36 36 36 36 36 36 30 36 36 36 36 54 4 \
            36 36 36 36 30 36 36 28 36 36 28 36 36 30 36 36 36 36 54 36 36 36 36 30 32)

#OPCIONES_VISUALIZACION= --debug --hex --bin --info
#OPCIONES_VISUALIZACION="--debug --bin --quit_null_frame"
OPCIONES_VISUALIZACION="--debug --bin --hex --quit_null_frame "

if [ -x ../Ejecutables/xhwicap_read_frame_polled_example_args ]; then
```

```
  echo "Info: Encontrado el archivo ejecutable xhwicap_read_frame_polled_example_args."
else
  echo "ERROR: Probablemente no ha ejecutado satisfactoriamente make"
  exit
fi

# El for del tipo de bloque
for block in {0..1}
do
   # El for de si es top o bottom del FPGA
   for top in {0..1}
   do
      # El for de la fila dentro del FPGA
      for hclk in {0..3}
      do
         # El for de la columna
         if [ ${block} -eq 0 ] ; then
            max_major=51
         else
            max_major=6
         fi
         for ((major=0;major<$max_major;major++))
         #major in {0..${max_major}}
         #for major in {0..50}
         do
            # El for del campo menor
            if [ ${block} -eq 0 ] ; then
               max_minor=${num_minors[${major}]}
            else
               max_minor=128
            fi
            for ((minor=0;minor<${max_minor};minor++))
            #for minor in {0..1}
            do
#              echo "Indices: ${block} ${top} ${hclk} ${major} ${minor}"
               ../Ejecutables/xhwicap_read_frame_polled_example_args --top=${top} \
                                 --block=${block} --hclk=${hclk} --major=${major} \
                                 --minor=${minor} ${OPCIONES_VISUALIZACION}
            done
         done
      done
   done
done
```

Cuadro de Código F.3: Código fuente del script `leer_toda_la_configuracion.sh`.

F.4 w_lee_y_escribe_frame.c

Este programa lee un frame, invierte uno de sus bits y lo vuelve a escribir en la configuración del FPGA. Como salida arroja el frame que leyó y el que escribió. Tiene las mismas opciones de visualización que el programa `xhwicap_read_frame_polled_example_args.c` y los mismos argumentos para identificar el frame que se va a modificar. Pero adicionalmente tiene otros dos argumentos que le permiten identificar la palabra al interior del frame y el bit que modificará. La lista completa de las opciones se encuentra en la Tabla F.2. Adicionalmente su código fuente se encuentra en F.4 y algunos ejemplos de uso en F.4.1.

Tabla F.2: Opciones de visualización de xhwicap_read_frame_polled_example_args.c

Opción	Descripción
`--debug`	Imprime cabeceros al inicio de cada frame que muestran los parámetros de su dirección como el tipo de bloque, top, fila, major y minor. Ejemplo: `FRAME=> Block=0 Top=1 Hclk/Row=1 Major=1 Minor=20`
`--hex`	Imprime en una columna la información del frame en hexadecimal. Ejemplo: `21000303`
`--bin`	Imprime en una columna la información del frame en binario. Ejemplo: `00100001000000000000001100000011`
`--info`	Imprime en frente de cada palabra del frame su índice. Ejemplo: `Frame Word 70 -> 21000303`
`--quit_null_frame`	No imprime un frame adicional que es leido siempre que se hace la lectura de un frame. Este frame es cero en su totalidad y no contiene información.
`--top`	Indica si el frame está en la parte superior (cero) o si está en la parte inferior (uno) del FPGA.
`--block`	Indica que tipo de frame se desea leer: cero para conexión y configuración de recursos lógicos y uno para el contenido de los bloques de memoria RAM.
`--hclk`	Indica en cual fila del FPGA está ubicado el frame que se va a leer. Su numeración comienza desde la parte central hacia los extremos.
`--major`	Indica en cuál columna del FPGA está ubicado el frame que se desea leer. Cuando block es cero, su numeración es correspondiente con la de la Tabla 3.3.
`--minor`	Indica cuál es el minor address del frame que se desea leer.
`--word`	De las 41 palabras del frame, este argumento permite seleccionar una de ellas. Puede tomar valores entre 0 y 40
`--bit`	Indica cuál de los 32 bits de la palabra se va a invertir.

F.4.1 Ejemplos de uso

Para invertir el comportamiento de la LUT A del slice izquierdo ubicado en la columna 16 de la fila 2 de la parte inferior del FPGA ante la entrada binaria `''111111''`

```
192# ./Ejecutables/w_lee_y_escribe_frame \
                --quit_null_frame --hex --bin --debug --top=1 \
                --block=0 --hclk=2 --major=16 --minor=32 --word=0 --bit=0
```

Para invertir el comportamiento de la LUT A del slice izquierdo ubicado en la columna 16 de la fila 2 de la parte inferior del FPGA ante la entrada binaria ''000000'', sin imprimir nada en pantalla:

```
192# ./Ejecutables/w_lee_y_escribe_frame \
                --quit_null_frame --top=1 --block=0 --hclk=2 --major=16 \
                --minor=35 --word=0 --bit=15
```

F.4.2 Código fuente

```
/*****************************************************************************
 * DESCRIPCION                                                               *
 *                                                                           *
 * Este programa busca invierte uno de los bits de la configuración actual   *
 * del FPGA. Para ello emplea las rutinas de lectura y escritura de frames.  *
 * El bit exacto es dado por la dirección completa del frame, el indice de   *
 * la palabra al interior del frame y el bit dentro de esa palabra.          *
 *****************************************************************************/

/***************************** Include Files *********************************/

#include <xparameters.h>
#include <xil_types.h>
#include <xil_assert.h>
#include <xhwicap.h>
#include <stdio.h>
#include <getopt.h>

/************************** Constant Definitions *****************************/

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define HWICAP_DEVICEID         XPAR_HWICAP_0_DEVICE_ID


/*
 * These are the parameters for reading a frame of data in
 * the slice SLICE_X0Y0
 */
#define HWICAP_EXAMPLE_TOP              1
#define HWICAP_EXAMPLE_BLOCK            0
#define HWICAP_EXAMPLE_HCLK             1
#define HWICAP_EXAMPLE_MAJOR            1
#define HWICAP_EXAMPLE_MINOR            20

//#define DEBUG_ARGS_INPUT
//#define DEBUG_DESCRIPTION

//#define printf  xil_printf    /* A smaller footprint printf */


/************************** Function Prototypes ******************************/
int HwIcapLeeyEscribeFrame(u16 DeviceId,int frame_top,int frame_block,int frame_hclk,\
    int frame_major,int frame_minor,int word, int bit,int debug,int hex,int bin,int info,\
    int quit_null_frame);

/************************** Variable Definitions *****************************/
static XHwIcap HwIcap;
```

```
/* Función auxiliar para imprimir en binario*/
char *  int2bin(u32 dec, char * bin) {
    int j;
//    char salida[32];
    for(j=0;j<32;j++) {
        bin[31-j]=(dec%2)?'1':'0';
        dec = dec/2;
    }
    return bin;
}

/* Función principal donde se reciben los argumentos y se llama la función del ejemplo*/
int main(int argc, char **argv)
{
        int Status;
// Lee los argumentos de entrada para ver cual es la direccion
// que va a leer.
    char* desc;
    int c;
    int digit_optind = 0;
// Variables que se pueden recibir por línea de comandos
    int frame_top   = HWICAP_EXAMPLE_TOP  ;
    int frame_block = HWICAP_EXAMPLE_BLOCK;
    int frame_hclk  = HWICAP_EXAMPLE_HCLK ;
    int frame_major = HWICAP_EXAMPLE_MAJOR;
    int frame_minor = HWICAP_EXAMPLE_MINOR;
    int word        = 0;
    int bit         = 0;
    int debug       = 0;
    int hex         = 0;
    int bin         = 0;
    int info        = 0;
    int quit_null_frame  = 0;

    unsigned long direccion = 0;
    int cantidad = 0;
    static struct option long_options[] = {
        {"top", required_argument, 0, 0},
        {"block", required_argument, 0, 0},
        {"hclk", required_argument, 0, 0},
        {"major", required_argument, 0, 0},
        {"minor", required_argument, 0, 0},
        {"word", required_argument, 0, 0},
        {"bit", required_argument, 0, 0},
        {"debug", no_argument, 0, 0},
        {"hex", no_argument, 0, 0},
        {"bin", no_argument, 0, 0},
        {"info", no_argument, 0, 0},
        {"quit_null_frame", no_argument, 0, 0},
        {NULL, 0, NULL, 0}
    };
    int option_index = 0;
    while ((c = getopt_long(argc, argv, "",
                    long_options, &option_index)) != -1) {
        int this_option_optind = optind ? optind : 1;
        switch (c) {
        case 0:
#ifdef DEBUG_ARGS_INPUT
            printf ("option %s", long_options[option_index].name);
#endif
            if(strcmp("top",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_top = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Top Field: %d",frame_top);
#endif
                }
            if(strcmp("block",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
```

```
#endif
                    frame_block = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    switch (frame_top) {
                    case 0:
                        desc="Interconnect and block configuration";
                        break;
                    case 1:
                        desc="Block RAM contents";
                        break;
                    default:
                        desc="no se...";
                    }
                    printf ("Block Field: %d (%s)",frame_block,desc);
#endif
                }
            if(strcmp("hclk",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_hclk = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("hclk Field: %d",frame_hclk);
#endif
                }
            if(strcmp("major",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_major = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Major Field: %d",frame_major);
#endif
                }
            if(strcmp("minor",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    frame_minor = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Minor Field: %d",frame_minor);
#endif
                }
            if(strcmp("word",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    word = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Word Field: %d",word);
#endif
                }
             if(strcmp("bit",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    bit = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Bit Field: %d",bit);
#endif
                }
             if(strcmp("debug",long_options[option_index].name)==0)
              debug=1;
            if(strcmp("hex",long_options[option_index].name)==0)
              hex=1;
            if(strcmp("bin",long_options[option_index].name)==0)
              bin=1;
            if(strcmp("info",long_options[option_index].name)==0)
              info=1;
```

```
            if(strcmp("quit_null_frame",long_options[option_index].name)==0)
              quit_null_frame=42;
#ifdef DEBUG_DESCRIPTION
              printf ("\n");
#endif
            break;
        case '?':
            break;
        default:
            printf ("?? getopt returned character code 0%o ??\n", c);
        }
    }


        /*
         * Run the HwIcap example, specify the Device ID generated in
         * xparameters.h
         */
        Status = HwIcapLeeyEscribeFrame(HWICAP_DEVICEID,frame_top,frame_block,frame_hclk,\
           frame_major,frame_minor,word,bit,debug,hex,bin,info,quit_null_frame);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        return XST_SUCCESS;

}

/* Función del ejemplo modificada porque permite leer y escribir */
int HwIcapLeeyEscribeFrame(u16 DeviceId,int frame_top,int frame_block,int frame_hclk,\
    int frame_major,int frame_minor,int word, int bit,int debug,int hex,int bin, int info,\
    int quit_null_frame)
{
        int Status;
        u32 Index;
        XHwIcap_Config *CfgPtr;
        u32  FrameData[XHI_NUM_WORDS_FRAME_INCL_NULL_FRAME];
        u32 mascara = 0;

        char dummychar[33];

        /*
         * Initialize the HwIcap instance.
         */
        CfgPtr = XHwIcap_LookupConfig(DeviceId);
        if (CfgPtr == NULL) {
                return XST_FAILURE;
        }

        Status = XHwIcap_CfgInitialize(&HwIcap, CfgPtr, CfgPtr->BaseAddress);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        /*
         * Perform a self-test to ensure that the hardware was built correctly.
         */
        Status = XHwIcap_SelfTest(&HwIcap);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        /*
         * Read the Frame
         */
        #if ((XHI_FAMILY == XHI_DEV_FAMILY_V4) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V5) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V6))

        Status = XHwIcap_DeviceReadFrame(&HwIcap,
                                         frame_top,
                                         frame_block,
                                         frame_hclk,
                                         frame_major,
```

```
                                        frame_minor,
                                        (u32 *) &FrameData[0]);

        if (Status != XST_SUCCESS) {
                printf("Failed to Read Frame: %d \n\r", Status);
                return XST_FAILURE;
        }
        #endif

        /*
         * Print Frame contents
         */
        dummychar[32]=0;
        if (debug)
                printf("FRAME=> Block=%d Top=%d Hclk/Row=%d Major=%d Minor=%d\n",frame_block,\
                    frame_top,frame_hclk,frame_major,frame_minor);
        for (Index = quit_null_frame;
                Index <= (XHI_NUM_FRAME_WORDS << 1) ; Index++) {
                if (info)
                        printf("Frame Word %d -> \t",
                                Index );
                if (hex)
                        printf(                         "%8.8X ",
                                FrameData[Index] );
                if (bin)
                        printf(                                 "%s",
                                                int2bin(FrameData[Index],&dummychar[0]));
                if ((info+hex+bin)!=0)
                        printf("\n");
        }


        /*
         * Impresion de los bits que creo que son de interés
         */

/*       dummychar[32]=0;
         Index = 2;
         printf("\nWords de interés:\nFrameData[quit_null_frame+1]=");
         printf("%8.8X = ",FrameData[quit_null_frame+1]);
         printf("%s\n",int2bin(FrameData[quit_null_frame+1],&dummychar[0]));
         printf("FrameData[quit_null_frame+2]=");
         printf("%8.8X = ",FrameData[quit_null_frame+2]);
         printf("%s\n\n",int2bin(FrameData[quit_null_frame+2],&dummychar[0]));
*/
        /*
         * Modificaciones al Bitstream
         */
         mascara = 1 << bit;
         FrameData[quit_null_frame+word] = mascara ^ FrameData[quit_null_frame+word];
        /*
         * Write the same frame
         */

        #if ((XHI_FAMILY == XHI_DEV_FAMILY_V4) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V5) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V6))

        Status = XHwIcap_DeviceWriteFrame(&HwIcap,
                                        frame_top,
                                        frame_block,
                                        frame_hclk,
                                        frame_major,
                                        frame_minor,
                                        (u32 *) &FrameData[0]);

        if (Status != XST_SUCCESS) {
                printf("Failed to Write Frame: %d \n\r", Status);
                return XST_FAILURE;
        }
        #endif


        /*
```

```
         * Read the Frame
         */
        #if ((XHI_FAMILY == XHI_DEV_FAMILY_V4) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V5) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V6))

        Status = XHwIcap_DeviceReadFrame(&HwIcap,
                                         frame_top,
                                         frame_block,
                                         frame_hclk,
                                         frame_major,
                                         frame_minor,
                                         (u32 *) &FrameData[0]);

        if (Status != XST_SUCCESS) {
                printf("Failed to Read Frame: %d \n\r", Status);
                return XST_FAILURE;
        }
        #endif

        /*
         * Print Frame contents
         */
        dummychar[32]=0;
        if (debug)
                printf("FRAME=> Block=%d Top=%d Hclk/Row=%d Major=%d Minor=%d\n",frame_block,\
                    frame_top,frame_hclk,frame_major,frame_minor);
        for (Index = quit_null_frame;
                Index <= (XHI_NUM_FRAME_WORDS << 1) ; Index++) {
                if (info)
                        printf("Frame Word %d -> \t",
                                Index );
                if (hex)
                        printf(                         "%8.8X ",
                                FrameData[Index] );
                if (bin)
                        printf(                                 "%s",
                                                int2bin(FrameData[Index],&dummychar[0]));
                if ((info+hex+bin)!=0)
                        printf("\n");

        }


        return  XST_SUCCESS;
}
```

Cuadro de Código F.4: Código fuente del programa `w_lee_y_escribe_frame.c`.

F.5 w_cambia_cada_bit_de_una_columna.sh

Este script explora el efecto de invertir cada uno de los bits de los frames que componen una columna. Una vez se cambia un bit, se retorna a su estado original para no causar algún daño al dispositivo. El uso de este script debe hacerse precavidamente teniendo en cuenta que no se tiene el conocimiento del efecto que tiene sobre el circuito cada uno de los bits de los frames. Su forma de uso se presenta a continuación, y su código fuente en el listado F.5

F.5.1 Ejemplo de uso

```
192# cd scripts_de_prueba
192# ./w_cambia_cada_bit_de_una_columna.sh
```

F.5.2 Código fuente

```
# DESCRIPCIÓN
#
# Este script emplea el programa w_lee_y_escribe_frame para modificar cada uno
# de los frames que componen una columna de CLBs, con el objetivo de encontrar
# cual de sus bits contiene la información de la tabla de verdad de cada uno
# de los LUTs del mismo.
#
# Advertencia:
#
# Este script modifica indiscriminadamente los bits del frame, por lo tanto
# podría llegar a causar daño al dispositivo.
#

if [ -x ../Ejecutables/w_lee_y_escribe_frame ]; then
  echo "Info: Encontrado el archivo ejecutable xhwicap_read_frame_polled_example_args."
else
  echo "ERROR: Probablemente no ha ejecutado satisfactoriamente make"
  exit
fi


for ((minor=0;minor<36;minor++))
do
  for ((word=0;word<41;word++))
  do
    for ((bit=0;bit<32;bit++))
    do
      echo "Modifiqué el bit ${bit} del word ${word} del minor ${minor}"
      ../Ejecutables/w_lee_y_escribe_frame --quit_null_frame --info --hex --bin  --top=1 \
            --block=0 --hclk=3 --major=1 --minor=${minor} --word=${word} --bit=${bit}
      sleep 5
      echo "Deshice el cambio."
      ../Ejecutables/w_lee_y_escribe_frame --quit_null_frame --info --hex --bin  --top=1 \
            --block=0 --hclk=3 --major=1 --minor=${minor} --word=${word} --bit=${bit}
      sleep 5
    done
  done
done
```

Cuadro de Código F.5: Código fuente del script `w_cambia_cada_bit_de_una_columna.sh`.

F.6 operaLUTs.c

Este programa tiene como objetivo verificar la tabla de verdad de las 32 LUTs que tiene implementadas el periférico reconfigurable de prueba, Para ello recorre todas las posibles combinaciones de la entrada y recolecta la información de la salida. Su ejecución no requiere parámetros de entrada y para el proyecto se ejecutó de forma permanente durante las pruebas de reconfiguración para ver en tiempo real el estado de las LUTs del periférico. Su código se muestra en el listado F.6 y su forma de uso a continuación.

F.6.1 Ejemplo de uso

```
192# ./prueba_LUTs/operaLUTs/operaLUTs
```

F.6.2 Código fuente

```
/*****************************************************************************
 * DESCRIPCIÓN:
 *
 * Esta fuente busca hacer uso del periférico de prueba, el cual tiene 6     *
 * registros de entrada, mediante los cuales opera 32 LUTs y su resultado es *
```

```
 * arrojado en la séptima posición de memoria                                *
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "xil_io.h"

// Esta es la baseaddress del periférico de prueba
#define ANDS_BASE_ADDRESS 0x814A0000

// Descomentar esta línea para imprimir la tabla de verdad de cada una de las LUTs. Si se deja
// comentada, lo único que devuelve el programa es la cantidad de LUTs que se han mantenido en
// su estado original.
//#define IMPRIME_TABLA_DE_VERDAD_DE_CADA_LUT

int main() {
  unsigned int j,k,mascara,dato;
  unsigned int datosleidos[64];
  unsigned int resultado1[32], resultado2[32];
  unsigned int contador;
  // Probar todas las combinaciones
  for(k=0;k<(64);k++) {
    for(j=0;j<6;j++) {
      mascara = 1 << j ;
      if(k & mascara)
        dato = 0xFFFFFFFF;
      else
        dato = 0x00000000;
      Xil_Out32(ANDS_BASE_ADDRESS+j*4,dato);
    }
    datosleidos[k]=Xil_In32(ANDS_BASE_ADDRESS+6*4);
  }
#ifdef IMPRIME_TABLA_DE_VERDAD_DE_CADA_LUT
  printf("Resultados:\n");
#endif
  // Borra el vector de resultados
  for(j=0;j<32;j++)
    resultado1[j] = 0;
  for(j=0;j<32;j++)
    resultado2[j] = 0;
  // Reorganiza los datos leidos
  for(k=0;k<32;k++) {
    for(j=0;j<32;j++) {
      mascara = 1 << j;
      if(datosleidos[k] & mascara)
        resultado1[j] += (1 << k);
    }
  }
   for(k=32;k<64;k++) {
    for(j=0;j<32;j++) {
      mascara = 1 << j;
      if(datosleidos[k] & mascara)
        resultado2[j] += (1 << (k-32));
    }
  }
  // Imprime los resultados
  contador = 0;
  for(j=0;j<32;j++) {
#ifdef IMPRIME_TABLA_DE_VERDAD_DE_CADA_LUT
    printf("LUT%2.2d : %8.8X%8.8X ",j,resultado2[j],resultado1[j]);
#endif
    if     (resultado2[j]==0x80000000 && resultado1[j]==0x00000000) {
#ifdef IMPRIME_TABLA_DE_VERDAD_DE_CADA_LUT
      printf("Compuerta AND de 6 entradas\n");
#endif
      contador++;
    }
#ifdef IMPRIME_TABLA_DE_VERDAD_DE_CADA_LUT
    else if(resultado2[j]==0x7FFFFFFF && resultado1[j]==0xFFFFFFFF)
      printf("Compuerta NAND de 6 entradas\n");
    else if(resultado2[j]==0xFFFFFFFF && resultado1[j]==0xFFFFFFFE)
```

```
      printf("Compuerta OR de 6 entradas\n");
    else if(resultado2[j]==0x00000000 && resultado1[j]==0x00000001)
      printf("Compuerta NOR de 6 entradas\n");
    else if(resultado2[j]==0x69969669 && resultado1[j]==0x96696996)
      printf("Compuerta XOR de 6 entradas\n");
    else if(resultado2[j]==0x96696996 && resultado1[j]==0x69969669)
      printf("Compuerta XNOR de 6 entradas\n");
    else if(resultado2[j]==0x00000000 && resultado1[j]==0x00000000)
      printf("Constante en cero\n");
    else if(resultado2[j]==0xFFFFFFFF && resultado1[j]==0xFFFFFFFF)
      printf("Constante en uno\n");
    else
      printf("\n");
#endif
  }
  printf("%d",contador);
}
```

Cuadro de Código F.6: Código fuente del programa `operaLUTs.c`.

F.7 w_escribe_una_LUT.c

Este programa modifica los frames necesarios para configurar el comportamiento de una LUT del FPGA. Toma como argumentos las coordenadas del slice, la identificación del LUT ($A = 0$, $B = 1$, $C = 2$ o $D = 3$) y la nueva tabla de verdad de la LUT. En el listado F.7 se encuentra el código fuente y en la siguiente sección se encuentran algunos ejemplos de uso.

F.7.1 Ejemplos de uso

Configurar como una compuerta OR la LUT que opera sobre los bits de mayor peso de los datos.

```
192# ./Ejecutables/w_escribe_una_LUT \
                  --slice_x=26 --slice_y=20 --LUT=0 \
--tabla_de_verdad=11111111111111111111111111111111111111111111111111111111111110
```

Introducir un valor específico sobre la tabla de verdad que modifica los bits de menos peso del periférico reconfigurable.

```
192# ./Ejecutables/w_escribe_una_LUT \
                  --slice_x=26 --slice_y=27 --LUT=3 \
--tabla_de_verdad=11111110110111001011101010011000011101100101010000110010000010000
```

F.7.2 Código Fuente

```
/*****************************************************************************
 * DESCRIPCION                                                               *
 *                                                                           *
 * Este programa busca invierte uno de los bits de la configuración actual   *
 * del FPGA. Para ello emplea las rutinas de lectura y escritura de frames.  *
 * El bit exacto es dado por la dirección completa del frame, el indice de   *
 * la palabra al interior del frame y el bit dentro de esa palabra.          *
 *****************************************************************************/

/*************************** Include Files ***********************************/

#include <xparameters.h>
#include <xil_types.h>
```

```c
#include <xil_assert.h>
#include <xhwicap.h>
#include <stdio.h>
#include <getopt.h>
#include <string.h>

/************************** Constant Definitions *****************************/

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define HWICAP_DEVICEID         XPAR_HWICAP_0_DEVICE_ID


//#define DEBUG_ARGS_INPUT
//#define DEBUG_DESCRIPTION
//#define DEBUG_CONFIGURACION_LUT

//#define printf  xil_printf    /* A smaller footprint printf */


/************************** Function Prototypes ******************************/
int HwIcapConfiguraLUT(u16 DeviceId,int slice_x,int slice_y,int LUT,char* tabla_de_verdad);

/************************** Variable Definitions *****************************/
static XHwIcap HwIcap;

/******* Funciones para el calculo de los campos de la dir del frame ********/
int calcula_frame_top(int slice_y) { return ((int) (slice_y)/40 + 1) % 2; }
int calcula_frame_hclk(int slice_y) {
    int frame_hclk  = ((int) slice_y/20);
    if (frame_hclk>3)
        frame_hclk = frame_hclk - 4;
    else
        frame_hclk = 3 - frame_hclk;
    return frame_hclk;
}

int calcula_frame_major(int slice_x) {
    int majors[38] = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, \
                      26, 27, 28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 42, 43, 45, 46, 47, 48};
    return majors[(int)(slice_x/2)];
}
int calcula_word(int slice_y, int LUT) { return 2*(slice_y%10) + ((int) LUT/2);}
int calcula_frame_minor(int slice_x, int Entrada) {
    int phi[8] = {3, 2, 0, 1, 2, 3, 1, 0};
    return 26 + 6*((slice_x+1)%2) + phi[Entrada%8];
}
int calcula_bit(int Entrada, int LUT) {return 15 - ((int) Entrada/4) + 16*(LUT%2);}

/* Función principal donde se reciben los argumentos y se llama la función del ejemplo*/
int main(int argc, char **argv)
{
    int Status;
    int c,valor,j,p;
    int slice_x             = 0;
    int slice_y             = 0;
    int LUT                 = 0;
    char tabla_de_verdad[64] = "00000000000000000000000000000000000000000000000000000000000000000";

    static struct option long_options[] = {
        {"slice_x", required_argument, 0, 0},
        {"slice_y", required_argument, 0, 0},
        {"LUT", required_argument, 0, 0},
        {"tabla_de_verdad", required_argument, 0, 0},
        {NULL, 0, NULL, 0}
    };
    int option_index = 0;
    while ((c = getopt_long(argc, argv, "",
                long_options, &option_index)) != -1) {
        int this_option_optind = optind ? optind : 1;
        switch (c) {
```

```
        case 0:
#ifdef DEBUG_ARGS_INPUT
            printf ("option %s", long_options[option_index].name);
#endif
            if(strcmp("slice_x",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    slice_x = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Recibí slice_x: %d",slice_x);
#endif
                }
            if(strcmp("slice_y",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    slice_y = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Block Field: %d",slice_y);
#endif
                }
            if(strcmp("LUT",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    LUT = (int) atoi(optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("LUT: %d (%c)",LUT,(char)('A'+LUT));
#endif
                }
            if(strcmp("tabla_de_verdad",long_options[option_index].name)==0)
                if (optarg) {
#ifdef DEBUG_ARGS_INPUT
                    printf (" with arg %s\n", optarg);
#endif
                    switch(strlen((char *)optarg)) {
                        case 64:
                        strcpy(tabla_de_verdad,optarg);
#ifdef DEBUG_DESCRIPTION
                    printf ("Tabla de verdad en Binario: %s",tabla_de_verdad);
#endif
                            break;
                        case 16:
                            // Conversión hexadecimal a binario
                            for(j=0;j<16;j++) {
                                if (optarg[j] >= '0' && optarg[j] <= '9')
                                    valor = (optarg[j] - '0');
                                else if (optarg[j] >= 'A' && optarg[j] <= 'F')
                                    valor = (optarg[j] - 'A' + 10);
                                else if (optarg[j] >= 'a' && optarg[j] <= 'f')
                                    valor = (optarg[j] - 'a' + 10);
                                else
                                    break;
                                for (p=0;p<4;p++) {
                                    tabla_de_verdad[4*j+3-p] = (valor%2)?'1':'0';
                                    valor = valor/2;
                                }
                            }
#ifdef DEBUG_DESCRIPTION
                    printf ("Tabla de verdad en Hexadecimal: %s",optarg);
#endif
                            break;
                         default:
                            printf("La tabla de verdad se debe introducir en hexa o en bin\n");
                            return -1;
                    }
                }
#ifdef DEBUG_DESCRIPTION
                printf ("\n");
#endif
```

```
            break;
        case '?':
            break;
        default:
            printf ("?? getopt returned character code 0%o ??\n", c);
        }
    }


        /*
         * Run the HwIcap example, specify the Device ID generated in
         * xparameters.h
         */
        Status = HwIcapConfiguraLUT(HWICAP_DEVICEID, slice_x ,slice_y ,LUT, tabla_de_verdad);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        return XST_SUCCESS;

}

/* Función del ejemplo modificada porque permite leer y escribir */
int HwIcapConfiguraLUT(u16 DeviceId,int slice_x,int slice_y,int LUT,char* tabla_de_verdad)
{
        int Status;
        u32 Index;
        XHwIcap_Config *CfgPtr;
        u32  FrameData[XHI_NUM_WORDS_FRAME_INCL_NULL_FRAME];
        u32 mascaraAND;
        u32 mascaraOR ;

        char dummychar[33];


        int frame_major;
        int frame_top;
        int frame_hclk;
        int frame_block;
        int frame_minor;
        int word;
        int bit;
        int frame_minor_test;
        int k;
        int Entrada;

#ifdef DEBUG_CONFIGURACION_LUT
        printf("Entré a la función principal y tengo lo siguiente:\n");
        printf("slice_x = %d\n",slice_x);
        printf("slice_y = %d\n",slice_y);
        printf("LUT = %d\n",LUT);
        printf("tabla_de_verdad = %s\n",tabla_de_verdad);
#endif

        /*
         * Initialize the HwIcap instance.
         */
        CfgPtr = XHwIcap_LookupConfig(DeviceId);
        if (CfgPtr == NULL) {
                return XST_FAILURE;
        }

        Status = XHwIcap_CfgInitialize(&HwIcap, CfgPtr, CfgPtr->BaseAddress);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        /*
         * Perform a self-test to ensure that the hardware was built correctly.
         */
        Status = XHwIcap_SelfTest(&HwIcap);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
```

```
        /*
         * Calcula los parámetros para saber cuales frames debe leer
         */

        frame_top   = calcula_frame_top(slice_y);
        frame_block = 0;
        frame_hclk  = calcula_frame_hclk(slice_y);
        frame_major = calcula_frame_major(slice_x);
        // La palabra siempre es la misma para una LUT
        word = calcula_word(slice_y,LUT);
#ifdef DEBUG_CONFIGURACION_LUT
        printf("Frame_top = %d\n",frame_top);
        printf("Frame_block = %d\n",frame_block);
        printf("Frame_hclk = %d\n",frame_hclk);
        printf("frame_major = %d\n",frame_major);
        printf("word = %d\n",word);
#endif

        /*
         * For que interará en los 4 frames que deben ser modificados
         */
        for(k=0; k<4; k++) {
            frame_minor = 26 + 6*((slice_x+1)%2) + k;
#ifdef DEBUG_CONFIGURACION_LUT
            printf("Modificaciones sobre el frame con minor = %d\n",frame_minor);
#endif

            /*
             * Read the Frame
             */
#if ((XHI_FAMILY == XHI_DEV_FAMILY_V4) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V5) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V6))

            Status = XHwIcap_DeviceReadFrame(&HwIcap,
                                             frame_top,
                                             frame_block,
                                             frame_hclk,
                                             frame_major,
                                             frame_minor,
                                             (u32 *) &FrameData[0]);

            if (Status != XST_SUCCESS) {
                printf("Failed to Read Frame: %d \n\r", Status);
                return XST_FAILURE;
            }
#endif

            /*
             * Modificaciones al frame
             */
            mascaraAND = 0xFFFFFFFF;
            mascaraOR  = 0x00000000;
            for(Entrada=0;Entrada<64;Entrada++) {
                frame_minor_test = calcula_frame_minor(slice_x,Entrada);
                if (frame_minor_test == frame_minor) {
                    bit = calcula_bit(Entrada,LUT);
                    if (tabla_de_verdad[63-Entrada]=='0')
                        mascaraAND ^= 1 << bit;
                    else
                        mascaraOR  ^= 1 << bit;
                }
            }
#ifdef DEBUG_CONFIGURACION_LUT
            printf("MascaraAND = %8.8x\n",mascaraAND);
            printf("MascaraOR  = %8.8x\n",mascaraOR);
            printf("Valor del FrameData[42+%d] = %8.8x\n", word, FrameData[42+word]);
#endif
            FrameData[42+word] &= mascaraAND ;
            FrameData[42+word] |= mascaraOR  ;
#ifdef DEBUG_CONFIGURACION_LUT
            printf("Valor del FrameData[42+%d] = %8.8x\n", word, FrameData[42+word]);
#endif
```

```
            /*
             * Write the same frame
             */

#if ((XHI_FAMILY == XHI_DEV_FAMILY_V4) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V5) || \
                (XHI_FAMILY == XHI_DEV_FAMILY_V6))

            Status = XHwIcap_DeviceWriteFrame(&HwIcap,
                                              frame_top,
                                              frame_block,
                                              frame_hclk,
                                              frame_major,
                                              frame_minor,
                                              (u32 *) &FrameData[0]);

            if (Status != XST_SUCCESS) {
                printf("Failed to Write Frame: %d \n\r", Status);
                return XST_FAILURE;
            }
#endif

        }

        return  XST_SUCCESS;
}
```

Cuadro de Código F.7: Código fuente del programa `w_escribe_una_LUT.c`.

Apéndice G

Código de la librería `xil_io` (versión espacio del *kernel*).

En este Anexo se presenta el código que sustituyó la librería de Xilinx® `xil_io` para el *driver* desde el espacio del *kernel*. Esta librería tiene como propósito escribir y leer un dato sobre cierta posición de memoria. Esta librería requiere que antes de ser invocada se haya ejecutado el llamado al sistema `open`.

G.1 Archivo `xil_io.c`

```
/*****************************************************************************
 * DESCRIPCION:                                                              *
 *                                                                           *
 * Este programa define dos rutinas para leer y escribir sobre una direccion *
 * física en el sistema. Está basado en el uso del nodo /dev/mem, y puede    *
 * ser empleado en las pruebas de un periférico, pero su desempeño será      *
 * inferior al de un driver diseñado en el espacio del kernel.               *
 *                                                                           *
 * Esta fuente está  basada en la rutina desarrollada para el manejo del LCD *
 * desde el espacio del usuario.                                             *
 *****************************************************************************/

// Entradas y salidas estandar
#include <stdio.h>

// Soporte para usleep
#include <time.h>

// stdlib....
#include <stdlib.h>

// Soporte para abrir y modificar los archivos como /dev/mem
#include <fcntl.h>


#include "xil_io.h"
#include "../xparameters.h"

//#define DEBUG_XIL_IO

extern size_t filedesc;

void Xil_Out32(u32 OutAddress,u32 Value)
{
//    size_t filedesc = open("/dev/Periferico_XPS_HWICAP",O_RDWR);
    int offset;

#ifdef DEBUG_XIL_IO
```

```
    printf("Xil_Out32:\t Info:\t Iniciando Xil_Out32(%p,0x%.8X)\n",OutAddress,Value);
#endif

    /* Validación de la dirección que se solicita escribir                    */
    offset = OutAddress - XPAR_XPS_HWICAP_0_BASEADDR;
    if(offset<0 || offset>0x0000FFFF) {
        printf("ERROR: La dirección que se solicitó escribir no está en el rango del periférico XPS_HWICAP\n");
        exit(EXIT_FAILURE);
    }

    /* En caso de no poder abrirlo, entonces se termina el programa           */
//    if(filedesc < 0) {
//        printf("No se pudo abrir el nodo /dev/Periferico_XPS_HWICAP\n");
//        exit(EXIT_FAILURE);
//    }

    /* Llamado al sistema lseek para ajustar el puntero del archivo           */
    if(lseek(filedesc,offset,SEEK_SET)<0) {
        printf("Hubo un error en lseek 1\n");
        exit(EXIT_FAILURE);
    }

    /* Llamado al sistema write para escribir los datos en el registro        */
    if(write(filedesc,&Value,4)<0) {
        printf("Hubo un error en write 1\n");
        exit(EXIT_FAILURE);
    }

#ifdef DEBUG_XIL_IO
    printf("Escribí sobre %p el dato %8.8X \n",OutAddress,Value);
#endif

    /* Llamado al sistema close                                               */
//    if(close(filedesc) < 0) {
//        printf("No se pudo cerrar el nodo /dev/Periferico_XPS_HWICAP\n");
//        exit(EXIT_FAILURE);
//    }
}

u32 Xil_In32(u32 InAddress)
{
//    size_t filedesc = open("/dev/Periferico_XPS_HWICAP",O_RDWR);
    int offset;
    u32 Value;

#ifdef DEBUG_XIL_IO
    printf("Xil_In32:\t Info:\t Iniciando Xil_In32(%.8p)\n",InAddress);
#endif

    /* Validación de la dirección que se solicita escribir                    */
    offset = InAddress - XPAR_XPS_HWICAP_0_BASEADDR;
    if(offset<0 || offset>0x0000FFFF) {
        printf("ERROR: La dirección que se solicitó leer no está en el rango del periférico XPS_HWICAP\n");
        exit(EXIT_FAILURE);
    }

    /* En caso de no poder abrirlo, entonces se termina el programa           */
//    if(filedesc < 0) {
//        printf("No se pudo abrir el nodo /dev/Periferico_XPS_HWICAP\n");
//        exit(EXIT_FAILURE);
//    }

    /* Llamado al sistema lseek para ajustar el puntero del archivo           */
    if(lseek(filedesc,offset,SEEK_SET)<0) {
        printf("Hubo un error en lseek 1\n");
        exit(EXIT_FAILURE);
    }

    /* Llamado al sistema read para leer los datos del registro               */
    if(read(filedesc,&Value,4)<0) {
        printf("Hubo un error en read\n");
        exit(EXIT_FAILURE);
    }
```

```
#ifdef DEBUG_XIL_IO
    printf("Leí de %X el dato %8.8X \n",InAddress,Value);
#endif

    /* Llamado al sistema close                                          */
//    if(close(filedesc) < 0) {
//        printf("No se pudo cerrar el nodo /dev/Periferico_XPS_HWICAP\n");
//        exit(EXIT_FAILURE);
//    }
    return Value;
}
```

Cuadro de Código G.1: Nueva librería xil_io: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.

Apéndice H

Plantilla del *driver*.

Esta plantilla aparece como el resultado de los conceptos aprendidos sobre diseño de módulos del *kernel*, más específicamente sobre *drivers* tipo `char`. El objetivo de esta plantilla, es que fácilmente se pueda manipular un periférico haciendo pequeñas adaptaciones que se enumerarán a continuación.

El desarrollo de esta plantilla concentra gran parte de los conceptos aprendidos de la lectura del texto [10]. Aunque en el texto se basan en el *driver* `scull`, ese *driver* no maneja ningún dispositivo *hardware*, únicamente memoria. Debido a esto, al *driver* se le ha implementado un mecanismo que restringe al periférico para que solo pueda ser accedido por un proceso a la vez. Así si un proceso desea mantener reservado el recurso solo debe abrir el nodo en el sistema de archivos.

En el código fuente se ha manejado la palabra clave PERIFERICO en letras mayúsculas para que pueda ser sustituido por el nombre del periférico que se está diseñando, así toman significado los nombres de las funciones, las variables y hasta el id con el que queda registrado en el *kernel*. Para adaptar un periférico para un periférico en particular, por ejemplo uno llamado `leds`, se deben seguir los siguientes pasos:

Nombre de las fuentes: Los archivos principales del *driver*, se llaman `periferico.c` y `periferico.h`, por ello se recomienda cambiarlos por un nombre más coherente con el dispositivo que se va a manejar. Puede ejecutar el siguiente comando para hacerlo:

```
192# rename -v 's/periferico/leds/' periferico.*
```

Cambio en el objetivo del Makefile: El makefile del directorio del *driver* asume que la fuente se sigue llamando periférico.c, por lo tanto se debe modificar la primera línea así:

```
obj-m   += leds.o
```

Modificaciones en `periferico.c`: Se requieren dos modificaciones para este archivo. Inicialmente hay que cambiar el nombre de la línea donde se incluye el archivo `periférico.h`, para que quede:

```
#include"leds.h"
```

El segundo cambio está relacionado con el nombre clave PERIFERICO, el cual puede ser sustituido mediante vim con la siguiente orden:

```
:%s/PERIFERICO/LEDS/g
```

Modificaciones en `periferico.h`: En este archivo existen varias definiciones importantes que deben ser ajustadas coherentemente con los datos del periférico. Ellos son:

PERIFERICO_NUMERO_REGISTROS indica la cantidad de registros que tiene el periférico.

PERIFERICO_BASEADDRESS es la dirección base del periférico.

PERIFERICO_NAME es el nombre con el que queda registrado el periférico.

PERIFERICO_MAGIC este debe ser ajustado a un valor diferente en cada periférico porque este valor es usado para generar los números de las opciones del llamado al sistema ioctl.

Los datos que tiene por defecto la plantilla están ajustados para el periférico que maneja los leds de la tarjeta. Igualmente se debe hacer la sustitución del nombre PERIFERICO.

```
:%s/PERIFERICO/LEDS/g
```

Modificaciones en `montar.sh`: Este es un script que facilita el montaje del módulo en el *kernel* ya que consulta el major y el minor asignados al periférico y crea los nodos en el sistema de archivos para que pueda ser accedido por los procesos. Allí se debe modificar los valores de las variables `module` y `device`. Así mismo la línea donde se asigna el major con el valor que tiene la variable `device`. Un ejemplo del archivo modificado es el siguiente:

```
#!/bin/sh
# set -x
module="leds"
device="Periferico_LEDS"
mode="664"

rm -f /dev/${device}
# invoke insmod with all arguments we got
# and use a pathname, as newer modutils don't look in . by default
/sbin/insmod ./$module.ko $* || exit 1

# remove stale nodes
rm -f /dev/${device}
# major=$(awk "\$2==\"$module\" {print \$1}" /proc/devices)
major=$(cat /proc/devices | awk \
'{if($2=="Periferico_LEDS") print $1 ; else print pailas}' | grep -v pailas)
mknod /dev/${device} c $major 0

```

A continuación se presenta el código fuente de los archivos de la plantilla:

H.1 Archivo periferico.h

```
/* Librería con las definiciones del periférico */

#ifndef _PERIFERICO_H_
#define _PERIFERICO_H_

#include<linux/ioctl.h>

/* Estructura del dispositivo*/
struct PERIFERICO_dev {
        unsigned int baseaddress;
        unsigned int numero_de_registros;
        unsigned int *intercambio_de_datos;
        char nombre_periferico[25];
        struct semaphore sem;
        struct cdev cdev;
};

/* Datos del periférico LCD                                              */
#define PERIFERICO_NUMERO_REGISTROS 4
#define PERIFERICO_BASEADDRESS 0x81400000
#define PERIFERICO_NAME "Periferico_PERIFERICO"


/* Comandos del ioctl */
#define PERIFERICO_MAGIC '&'
#define COMANDO_0 _IO(PERIFERICO_MAGIC,1)
#define COMANDO_1 _IO(PERIFERICO_MAGIC,2)


#endif
```

Cuadro de Código H.1: Nueva librería xil_io: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.

H.2 Archivo periferico.c

```
/*************************************************************************
 * Driver para un PERIFERICO. Este archivo puede servir como plantilla   *
```

```
 * para otros dispositivos a los que se les quiera implementar un driver.     *
 ******************************************************************************/

/***********************
 * Librerías generales *
 ***********************/

/* Soporte para establecer las funciones de inicialización y finalización     *
 * mediante los macros module_init module_exit                                */
#include<linux/init.h>

/* Obligatorio para cualquier módulo                                          */
#include<linux/module.h>

/* Definiciones generales y la mayoría de los macros que interactuan con el   *
 * kernel                                                                     */
#include<linux/sched.h>

/* Funciones relacionadas con la escritura drivers, acá estan las funciones   *
 * para registrar los numeros de dispositivos y los cdev                      */
#include<linux/fs.h>

/* Permite instanciar y manipular los cdev                                    */
#include<linux/cdev.h>

/* ?? */
#include<linux/slab.h>

/* Soporta copy_to_user y copy_from_user                                      */
#include<asm/uaccess.h>

/* Soporte para manejar los puertos                                           */
#include<linux/ioport.h>

/* Soporte para las funciones inx outx                                        */
#include<asm/io.h>

/* Soporte para las barreras                                                  */
#include<asm/system.h>

/* Soporte para msleep                                                        */
#include<linux/delay.h>

/****************************
 * Librerías personalizadas *
 ****************************/

/* Definiciones para el PERIFERICO                                            */
#include "periferico.h"

/****************************
 * Librerías personalizadas *
 ****************************/

/* Cantidad de caracteres que se copian cada vez que se llama las funciones   *
 * PERIFERICO_write o PERIFERICO_read.                                        */
#define PERIFERICO_MEMSPACE_SIZE PERIFERICO_NUMERO_REGISTROS*4 //n reg * 4 bytes

/* Habilita el cambio de endianness en la escritura y lectura                 */
#define cambio_endianness

/* Habilita la impresión de mensajes de depuración mediante printk            */
#define depura

/* Nivel del printk por defecto                                               */
#define PRINTK_LEVEL KERN_INFO

/* Puntero devuelto por ioremap                                               */
static void * PERIFERICO_iomem_pointer;

/* Licencia bajo la cual se desarrolló el driver                              */
MODULE_LICENSE("Dual_BSD/GPL");

/******************************************
```

```
 * Parámetros al momento de cargar el módulo *
 *********************************************/
static char *whom   = "Mundo";
static int  howmany = 1;
module_param(whom,     charp, S_IRUGO);
module_param(howmany, int,   S_IRUGO);

/************************
 * Parámetros del driver *
 ************************/

/* Nombre que aparecerá en el /proc/devices                                  */
static char *nombre = PERIFERICO_NAME;
/* El primer minor number es cero                                            */
static unsigned int firstminor = 0;
/* Indica cuantos devices PERIFERICO existen, para reservar igual cantidad de *
 * minor numbers.                                                            */
static unsigned int count = 1;

/* Dato de tipo dev_t que se usa en las funciones PERIFERICO_init y          *
 * PERIFERICO_exit y agrupa los major y minor numbers asignados al           *
 * dispositivo PERIFERICO                                                    */
static dev_t mydev;
/* Indica si el driver ha sido inicializado, en otras palabras, si alguna vez *
 * se ha ejecutado la función open.                                          */
static char bandera;


/* Función change_endianness: Tiene como objetivo cambiar el endianness a un  *
 * dato de 32 bits tipo u32.                                                 */
u32 change_endianness(u32 dato)
{
    u32 valret=0;
    *((u8 *) &valret)+3) = *((u8 *)&dato)+0);
    *((u8 *) &valret)+2) = *((u8 *)&dato)+1);
    *((u8 *) &valret)+1) = *((u8 *)&dato)+2);
    *((u8 *) &valret)+0) = *((u8 *)&dato)+3);
    return valret;
}




/* Función PERIFERICO_llseek:                                                */
loff_t PERIFERICO_llseek (struct file *puntero_file, loff_t off, int whence)
{
    /* Variable que almacena el valor a devolver (nuevo puntero)             */
    loff_t retval;
    /* Conserva un puntero a la estructura de tipo PERIFERICO_dev            */
    struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_llseek: Estoy ejecutando la función PERIFERICO_llseek\
#endif
    switch(whence){
        case 0: /* SEEK_SET  */
                retval = off;
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_llseek: Estoy ejecutando SEEK_SET\n");
#endif
                break;
        case 1: /* SEEK_CUR  */
                retval = puntero_file->f_pos + off;
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_llseek: Estoy ejecutando SEEK_CUR\n");
#endif
                break;
        case 2: /* SEEK_END  */
                retval = el_PERIFERICO->numero_de_registros*4 + off;
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_llseek: Estoy ejecutando SEEK_END\n");
#endif
                break;

        default: /* can't happen  */
```

```
                return -EINVAL;
        }
        if (retval < 0) return -EINVAL;
        puntero_file->f_pos = retval;
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_llseek: Salí bien de la función PERIFERICO_llseek\n");
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_llseek: El valor devuelto es: %d\n",(int) retval);
#endif
        return retval;
}




/* Función PERIFERICO_read: El kernel solicita leer "tamano" bytes al driver. *
 * El driver responde con la cantidad de bytes leidos o cero si se llegó al   *
 * final del archivo.                                                          */
ssize_t PERIFERICO_read (struct file *puntero_file, char __user *puntero_usuario, \
size_t tamano, loff_t * puntero_offset)
{
    /* Variable que almacena el valor a devolver (Bytes leidos)               */
    ssize_t retval;
    /* Conserva un puntero a la estructura de tipo PERIFERICO_dev             */
    struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
    /* Tamaño del buffer calcluado según memoria de caracteres                */
    int cantidad_de_registros = el_PERIFERICO->numero_de_registros;

    unsigned int leido;
    /* Variable para el for */
    int k;


#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Estoy ejecutando la función PERIFERICO_read\n");
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Tamano = %d\n",tamano);
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: El contenido de puntero_offset es: %lu\n",(long unsi
#endif
    /* Dejamos en tamano el menor entre tamano y PERIFERICO_MEMSPACE_SIZE     *
     * porque el driver no piensa leer mas que eso (él es perezoso)           */
    if(tamano>PERIFERICO_MEMSPACE_SIZE)
        tamano = PERIFERICO_MEMSPACE_SIZE;

    /* Verificamos si se puede sobrepasar el tamaño máximo del buffer y       *
     * limitamos la cantidad de datos leidos en caso de sobrepasarnos         */
    if((*puntero_offset + tamano)>=cantidad_de_registros*4) { // Si sobrepasa límite...
        retval = cantidad_de_registros*4 - *puntero_offset;   // Leeremos lo que podemos
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Lo que nos \
piden sobrepasa la cantidad de registros del PERIFERICO, así que solo leemos \
lo que podemos hasta llegar al final. retval = %d\n",retval);
#endif
    } else {                                  // Si no lo sobrepasa...
        retval = tamano;                      // Leemos lo que nos piden
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: La cantidad \
de datos solicitados no excede el numero de registros del PERIFERICO, así que \
leeremos los datos solicitados. retval = %d\n",retval);
#endif
    }

#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Voy a leer %d \
datos\n", retval);
#endif

    if(retval > 0) {
        for(k=0; k<retval; k=k+4) {
        /* Estas barreras son necesarias para evitar optimizaciones del       *
         * compilador al darse cuenta que se está escribiendo sobre el mismo  *
         * registro                                                            */
            barrier();
            mb();
```

```
        /* Entre cada una de las escrituras es necesario esperar un tiempo    */
            msleep(1);

        /* Ahora si se envía el caracter al PERIFERICO                        */
            leido=ioread32(PERIFERICO_iomem_pointer + (*puntero_offset + k));
#ifdef cambio_endianness
            leido = change_endianness(leido);
#endif
#ifdef depura
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Voy a \
leer la palabra número %d\r\n",k/4);
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Lei el \
valor 0 to 31 = %8.8X\r\n",leido);
#endif
       /* Se actualiza la estructura de datos que contiene el valor de los    *
        * registros                                                           */
            *(el_PERIFERICO->intercambio_de_datos + (*puntero_offset + k/4))=leido;
        }

        /* Se realiza el traspaso de la información desde el driver el        *
         * usuario. En caso de salir mal entonces se envía el código del      *
         * error.                                                             */
        if(copy_to_user(puntero_usuario, el_PERIFERICO->intercambio_de_datos + (long unsigned int)*puntero_offset
            retval = -EFAULT;
        else {
#ifdef depura
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Se \
leyeron bien los datos\n");
#endif
            /* En caso de que todo salga bien entonces se actualiza el        *
             * puntero con los datos leidos.                                  */
            *puntero_offset += retval;
        }
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_read: Al salir de \
la función, *puntero_offset = %lu\n\n",(long unsigned int)*puntero_offset);
#endif
    }
    return retval;
}




/* Función PERIFERICO_write:                                                  *
 *                  El kernel solicita escribir tamano bytes al driver. La    *
 * función efectua la lectura y devuelve la cantidad de bytes que pudo leer.  *
 * Además actualiza el puntero del usuario puntero3 según los bytes leidos.   */
ssize_t PERIFERICO_write (struct file *puntero_file, const char __user *puntero_usuario, \
size_t tamano, loff_t * puntero_offset)
{
    /* Variable para el for */
    int k;
    /* Variable que almacena lo que se va a escribir en el reg del PERIFERICO */
    u32 una_palabra;
    /* Variable que almacena el valor a devolver (Bytes escritos)             */
    ssize_t retval;
    /* Conserva un puntero a la estructura de tipo PERIFERICO_dev             */
    struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Estoy ejecutando \
la función PERIFERICO_write\n");
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Tamano = %d\n", \
tamano);
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: El contenido de \
puntero_offset es: %lu\n",(long unsigned int)*puntero_offset);
#endif
    /* tamano es la cantidad de bytes que se desean escribir y                *
     * PERIFERICO_MEMSPACE_SIZE es la cantidad de registros de 32 bits que se *
     * pueden escribir                                                        */
    if(tamano>PERIFERICO_MEMSPACE_SIZE)
        retval = PERIFERICO_MEMSPACE_SIZE;
    else
```

```
        retval = tamano;

    /* Se realiza el traspaso de la información hacia el driver el usuario.    *      * En caso de salir mal entonces */
    if(copy_from_user(el_PERIFERICO->intercambio_de_datos + (long unsigned int)*puntero_offset, puntero_usuario,
        return -EFAULT;

#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Se trajeron %d \
bytes desde el espacio del usuario al del PERIFERICO. \n", retval);
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: El valor del \
puntero virtual del I/O Port as I/O Memory base es: %p\n",                  \
PERIFERICO_iomem_pointer);
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Con este \
puntero se harán todas las escrituras al PERIFERICO\n");
#endif

   /* Ahora se enviará caracter por caracter la información al PERIFERICO     */
    for(k=0; k<retval; k=k+4) {

#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Voy a \
escribir la palabra número %d\r\n",k/4);
#endif
        /* Estas barreras son necesarias para evitar optimizaciones del       *
         * compilador al darse cuenta que se está escribiendo sobre el mismo  *
         * registro                                                           */
        barrier();
        mb();

        /* Descomentar cuando hay que esperar un tiempo entre cada escritura  */
        //msleep(1);

        /* Ahora si se envía la palabra al PERIFERICO                         */
        una_palabra = *(el_PERIFERICO->intercambio_de_datos + (*puntero_offset + k/4));
        /* Explicación de porque hay que cambiar el endianness al dato ??     */
#ifdef cambio_endianness
        una_palabra = change_endianness(una_palabra);
#endif
        iowrite32(una_palabra,PERIFERICO_iomem_pointer + *puntero_offset + k);

#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Escribi en \
la direccion = %X\r\n",(unsigned int)PERIFERICO_iomem_pointer + k);
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Escribi el \
valor 0 to 31 = 0x%08X\r\n",change_endianness(una_palabra));
/*        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Escribi el \
valor 0 to 31 = 0x%8.8X\r\n",una_palabra); */
#endif
    }

#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: Se escribieron bien los datos en el PERIFERICO.\n"
#endif

    /* En caso de que todo salga bien entonces se actualiza el puntero con    *
     * los datos leidos.                                                      */
    *puntero_offset += retval;

#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_write: El contenido de puntero_offset al final de la funci
#endif
    return retval;
}









/* Función PERIFERICO_ioctl:                                                  */
int PERIFERICO_ioctl (struct inode *inode1, struct file *puntero1, unsigned int comando, unsigned long parametro)
```

```
{
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: Estoy ejecutando la función PERIFERICO_ioctl\n"
#endif
    switch(comando) {
        case COMANDO_0 :
#ifdef depura
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: El comando fue COMANDO_0\n");
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: Pongo el periférico como salida\n");
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: Escribo 0x00000000 en slv_reg1\n");
#endif
            iowrite32(0x00000000,PERIFERICO_iomem_pointer + 4);
            break;

        case COMANDO_1 :
#ifdef depura
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: El comando fue COMANDO_1\n");
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: Pongo el periférico como entrada\n");
            printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_ioctl: Escribo 0xFFFFFFFF en slv_reg1\n");
#endif
            iowrite32(0xFFFFFFFF,PERIFERICO_iomem_pointer + 4);
            break;

        default:
            return -ENOTTY;
    }
    return 0;
}




/* Función PERIFERICO_open:                                                 *
 *                  Esta función se llama cuando algún proceso del sistema  *
 * operativo abre el nodo /dev/PERIFERICO como si fuera un archivo.         */
int PERIFERICO_open (struct inode *puntero_inode, struct file *puntero_file)
{
    /* Este puntero almacenará el puntero al PERIFERICO_dev que devolverá la  *
     * función container_of y posteriormente se almacena en el puntero_file   *
     * para que pueda ser usado por las demás funciones del módulo            */
    struct PERIFERICO_dev *el_PERIFERICO;
    /* Almacenará el número de registros del PERIFERICO                       */
    int tamano;
    /* Variable del for                                                       */
    int k;
#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_open: Estoy ejecutando \
la función PERIFERICO_open\n");
#endif
    /* Esta función busca cual struct PERIFERICO_dev contiene                 *
     * puntero_inode->i_cdev, el cual es de tipo cdev.                        */
    el_PERIFERICO = container_of(puntero_inode->i_cdev,struct PERIFERICO_dev,cdev);
    /* Verificamos que seamos los únicos que han abierto el PERIFERICO. Si no *
     * debemos retornar un error que haga que el llamado al sistema se repita */
    if(down_interruptible(&el_PERIFERICO->sem))
        return -ERESTARTSYS;
    /* El puntero_file llega a todas las funciones, por ello se aprovecha el  *
     * campo private_data para almacenar el puntero                           */
    puntero_file->private_data = el_PERIFERICO;
    /* Pregunta si alguna vez ha sido invocado el open para saber si debe     *
     * asignar los parámetros del periférico                                  */
    if(bandera==0) {
        el_PERIFERICO->baseaddress = PERIFERICO_BASEADDRESS;
        el_PERIFERICO->numero_de_registros = PERIFERICO_NUMERO_REGISTROS;
        strcpy(el_PERIFERICO->nombre_periferico, PERIFERICO_NAME);
        tamano = el_PERIFERICO->numero_de_registros;
        el_PERIFERICO->intercambio_de_datos = kmalloc(tamano*sizeof(unsigned int),GFP_KERNEL);
        for(k=0;k<tamano;k++)
            el_PERIFERICO->intercambio_de_datos[k] = 0;
        bandera=1;
#ifdef depura
```

```
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_open: El mensaje \
guardado en el_PERIFERICO->nombre_periferico es: %s\n",                    \
el_PERIFERICO->nombre_periferico );
#endif
    } else {
#ifdef depura
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_open: Ya estaba    \
inicializado :)\n");
#endif
    }
    /* Reserva del puerto... esto se refleja inmediatamente en /proc/ioports  */
    if(!request_region(PERIFERICO_BASEADDRESS,4,PERIFERICO_NAME))
        printk(PRINTK_LEVEL "PERIFERICO_open: No se pudo hacer exitosamente el \
 request del puerto PERIFERICO :(\n");

    /* Se mapea la dirección física del puerto en la porción de memoria tipo  *
     * I/O. El tamaño de la reserva es de 64 bytes                             */
    PERIFERICO_iomem_pointer=ioremap(PERIFERICO_BASEADDRESS,0x00000040);

    return 0;
}







/* Función PERIFERICO_release: Esta función libera lo que la función          *
 * PERIFERICO_open ha reservado.                                               */
int PERIFERICO_release (struct inode *inode1, struct file *puntero_file)
{
    struct PERIFERICO_dev *el_PERIFERICO = puntero_file->private_data;
#ifdef depura
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_release: Estoy \
ejecutando la función PERIFERICO_release\n");
#endif
    /* liberacion de la memoria I/O sobre la que se mapeo el puerto           */
    iounmap(PERIFERICO_iomem_pointer);

    /* Liberación del puerto reservado                                        */
    release_region(PERIFERICO_BASEADDRESS,4);

    /* Libera el semáforo                                                     */
    up(&el_PERIFERICO->sem);
    return 0;
}







/* En esta estructura se definen las operaciones que se van implementar en el *
 * driver                                                                      */
struct file_operations PERIFERICO_fops = {
        .owner    = THIS_MODULE,
        .llseek   = PERIFERICO_llseek,
        .read     = PERIFERICO_read,
        .write    = PERIFERICO_write,
        .ioctl    = PERIFERICO_ioctl,
        .open     = PERIFERICO_open,
        .release = PERIFERICO_release
};









/* Esta estructura representa los dispositivos PERIFERICO implementados. Para *
 * este caso fue solo uno. La memoria que emplea este dispositivo es          *
```

```
 * reservada en PERIFERICO_init                                          */
struct PERIFERICO_dev *my_PERIFERICO_dev;









/* Función PERIFERICO_init: Esta función es llamada cuando se hace insmod y  *
 * reserva los números major y minor y registrar el dispositivo.            */
static int PERIFERICO_init(void)
{
    /* Variable usada para recibir el código de error de la función cdev_init */
    int err;
#ifdef depura
    /* Variable del for que repite el mensaje de saludo.                     */
    int k;
#endif
    /* Variable que recibe el valor de la función alloc_chrdev_region        */
    int result;

    /* Reserva dinámica de los números del driver.                           */
    result = alloc_chrdev_region(&mydev,firstminor,count,nombre);
#ifdef depura
    if(result==0)
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_init: Los números \
reservados para el driver fueron:\n Major: %d\n Minor: %d\n", MAJOR(mydev), \
MINOR(mydev));
    else
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_init: Hubo un error \
y los números no se reservaron. El error fue: %d\n",result);
#endif

    /* Reserva memoria en el espacio del kernel para PERIFERICO_dev, es      *
     * decir, que inicializa el puntero my_PERIFERICO_dev                    */
    my_PERIFERICO_dev = kmalloc(sizeof(struct PERIFERICO_dev), GFP_KERNEL);
#ifdef depura
    if(my_PERIFERICO_dev==NULL)
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_init: Pailas, tocó \
reiniciar, no pude reservar memoria\n");
    else
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_init: Congratulations, \
sos un duro, reservaste memoria para my_PERIFERICO_dev, no tocó reiniciar, se \
reservaron %d Bytes\n",(int) sizeof(struct PERIFERICO_dev));
#endif
    /* Inicialización del semaforo del periferico                            */
    init_MUTEX(&my_PERIFERICO_dev->sem);
    /* Método para reservar la memoria para el cdev. Este método permite que *
     * el cdev sea parte de nuestra estructura pesonalizada my_PERIFERICO    */
    cdev_init(&my_PERIFERICO_dev->cdev,&PERIFERICO_fops);
    /* Registro del driver en el kernel. Este método no se debe llamar hasta *
     * no tener todo listo para atender cualquier llamado                    */
    err = cdev_add(&my_PERIFERICO_dev->cdev, mydev, 1);
    /* Inicializa algunos campos de la estructura my_PERIFERICO_dev          */
    my_PERIFERICO_dev->cdev.owner = THIS_MODULE;
    my_PERIFERICO_dev->cdev.ops = &PERIFERICO_fops;

#ifdef depura
    if(err<0)
        printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_init: Hubo un error \
al registrar el dispositivo. El error fue: %d\n",err);
    /* Mensaje de saludo para demostrar el uso de los parámetros             */
    for (k=0;k<howmany;k++)
        printk( "PERIFERICO_hello: PERIFERICO_init: Hola %s\n",whom);
    /* Prueba que permite saber cual proceso lo invoca a uno                 */
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_init: He sido invocado \
por:[%i] %s\n",current->pid,current->comm);
#endif

    /* Se pone la bandera a cero para indicar que el driver nunca ha sido    *
     * inicializado                                                          */
    bandera=0;
```

```
    return result;
}

/* Función PERIFERICO_exit:                                                 *
                    Esta función es invocada cuando se ejecuta rmmod y debe  *
 * liberar y deshacer lo que la función PERIFERICO_init ha reservado        */
static void PERIFERICO_exit(void)
{
    /* Desregistra los números major y minor que le fueron asignados de forma *
     * dinámica*/
    unregister_chrdev_region(mydev,count);
    /* Remueve el dispositivo del kernel                                    */
    cdev_del(&my_PERIFERICO_dev->cdev);
    /* Libera el espacio de memoria de los registros del PERIFERICO         */
    if(&my_PERIFERICO_dev->intercambio_de_datos != NULL && bandera==1)
        kfree(my_PERIFERICO_dev->intercambio_de_datos);
    /* Libera la memoria reservada para el struct PERIFERICO_dev            */
    kfree(my_PERIFERICO_dev);
#ifdef depura
    /* Mensaje de despedida                                                 */
    printk(PRINTK_LEVEL "PERIFERICO_hello: PERIFERICO_exit: Bye bye, Mundo\n");
    /* Prueba que permite saber cual proceso lo invoca a uno                */
    printk(PRINTK_LEVEL "PERIFERICO_depura: PERIFERICO_exit: He sido invocado por:[%i] %s\n",current->pid,current
#endif
}

/* Indica cuales funciones son las que se deben ejecutar al ejecutar insmod y *  * rmmod.
*/
module_init(PERIFERICO_init);
module_exit(PERIFERICO_exit);

/* Información adicional sobre el módulo.                                   */
MODULE_AUTHOR("William Salamanca y Sergio Abreo.");
MODULE_ALIAS("PERIFERICO_template");
MODULE_DESCRIPTION("Driver que maneja un PERIFERICO y que sirve como plantilla\
 para controladores de otro tipo de periféricos.");
```

Cuadro de Código H.2: Nueva librería `xil_io`: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.

H.3 Archivo `montar.sh`

```
#!/bin/sh
# set -x
module="periferico"
device="Periferico_PERIFERICO"
mode="666"

rm -f /dev/${device}
# invoke insmod with all arguments we got
# and use a pathname, as newer modutils don't look in . by default
/sbin/insmod ./$module.ko $* || exit 1

# remove stale nodes
rm -f /dev/${device}
# major=$(awk "\$2==\"$module\" {print \$1}" /proc/devices)
major=$(cat /proc/devices | awk '{if($2=="Periferico_PERIFERICO") print $1 ; else print pailas}' | grep -v pailas
mknod /dev/${device} c $major 0
#mknod /dev/${device}1 c $major 1
#mknod /dev/${device}2 c $major 2
#mknod /dev/${device}3 c $major 3

# give appropriate group/permissions, and change the group.
# Not all distributions have staff, some have "wheel" instead.
#group="staff"
#grep -q '^staff:' /etc/group || group="wheel"
#chgrp $group /dev/${device}[0-3]
chmod $mode  /dev/${device}
```

Cuadro de Código H.3: Nueva librería `xil_io`: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.

Módulo del *kernel* (Adaptación de la plantilla).

El presente Anexo muestra el código fuente del módulo que se añadió al *kernel* para realizar las escrituras y lecturas de los registros del preriférico XPS_HWICAP.

I.1 Archivo `XPS_HWICAP.h`

```
/* Librería con las definiciones del periférico */

#ifndef _XPS_HWICAP_H_
#define _XPS_HWICAP_H_

#include<linux/ioctl.h>

/* Estructura del dispositivo*/
struct XPS_HWICAP_dev {
        unsigned int baseaddress;
        unsigned int numero_de_registros;
        unsigned int *intercambio_de_datos;
        char nombre_periferico[25];
        struct semaphore sem;
        struct cdev cdev;
};

/* Datos del periférico LCD                                              */
//#define XPS_HWICAP_NUMERO_REGISTROS 0x46
#define XPS_HWICAP_NUMERO_REGISTROS 0x47
#define XPS_HWICAP_BASEADDRESS 0x81480000
#define XPS_HWICAP_NAME "Periferico_XPS_HWICAP"


/* Comandos del ioctl */
#define XPS_HWICAP_MAGIC 'W'
#define COMANDO_0 _IO(XPS_HWICAP_MAGIC,1)
#define COMANDO_1 _IO(XPS_HWICAP_MAGIC,2)


#endif
```

Cuadro de Código I.1: Nueva librería `xil_io`: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.

I.2 Archivo `XPS_HWICAP.c`

```
/*****************************************************************************
 * Driver para un XPS_HWICAP. Este archivo puede servir como plantilla       *
 * para otros dispositivos a los que se les quiera implementar un driver.    *
 *****************************************************************************/
```

```
/**********************
 * Librerías generales *
 **********************/

/* Soporte para establecer las funciones de inicialización y finalización     *
 * mediante los macros module_init module_exit                                */
#include<linux/init.h>

/* Obligatorio para cualquier módulo                                          */
#include<linux/module.h>

/* Definiciones generales y la mayoría de los macros que interactuan con el   *
 * kernel                                                                     */
#include<linux/sched.h>

/* Funciones relacionadas con la escritura drivers, acá estan las funciones   *
 * para registrar los numeros de dispositivos y los cdev                      */
#include<linux/fs.h>

/* Permite instanciar y manipular los cdev                                    */
#include<linux/cdev.h>

/* ?? */
#include<linux/slab.h>

/* Soporta copy_to_user y copy_from_user                                      */
#include<asm/uaccess.h>

/* Soporte para manejar los puertos                                           */
#include<linux/ioport.h>

/* Soporte para las funciones inx outx                                        */
#include<asm/io.h>

/* Soporte para las barreras                                                  */
#include<asm/system.h>

/* Soporte para msleep                                                        */
#include<linux/delay.h>

/***************************
 * Librerías personalizadas *
 ***************************/

/* Definiciones para el XPS_HWICAP                                            */
#include "XPS_HWICAP.h"

/***************************
 * Librerías personalizadas *
 ***************************/

/* Cantidad de caracteres que se copian cada vez que se llama las funciones   *
 * XPS_HWICAP_write o XPS_HWICAP_read.                                        */
#define XPS_HWICAP_MEMSPACE_SIZE XPS_HWICAP_NUMERO_REGISTROS*4 //n reg * 4 bytes

/* Habilita el cambio de endianness en la escritura y lectura                 */
#define cambio_endianness

/* Habilita la impresión de mensajes de depuración mediante printk            */
//#define depura

/* Nivel del printk por defecto                                               */
#define PRINTK_LEVEL KERN_INFO

/* Puntero devuelto por ioremap                                               */
static void * XPS_HWICAP_iomem_pointer;

/* Licencia bajo la cual se desarrolló el driver                              */
MODULE_LICENSE("Dual_BSD/GPL");

/*****************************************
 * Parámetros al momento de cargar el módulo *
 *****************************************/
```

```
static char *whom   = "Mundo";
static int  howmany = 1;
module_param(whom,     charp, S_IRUGO);
module_param(howmany, int,   S_IRUGO);

/************************
 * Parámetros del driver *
 ************************/

/* Nombre que aparecerá en el /proc/devices                                  */
static char *nombre = XPS_HWICAP_NAME;
/* El primer minor number es cero                                            */
static unsigned int firstminor = 0;
/* Indica cuantos devices XPS_HWICAP existen, para reservar igual cantidad de *
 * minor numbers.                                                            */
static unsigned int count = 1;

/* Dato de tipo dev_t que se usa en las funciones XPS_HWICAP_init y          *
 * XPS_HWICAP_exit y agrupa los major y minor numbers asignados al           *
 * dispositivo XPS_HWICAP                                                    */
static dev_t mydev;
/* Indica si el driver ha sido inicializado, en otras palabras, si alguna vez *
 * se ha ejecutado la función open.                                          */
static char bandera;


/* Función change_endianness: Tiene como objetivo cambiar el endianness a un  *
 * dato de 32 bits tipo u32.                                                 */
u32 change_endianness(u32 dato)
{
    u32 valret=0;
    *(((u8 *) &valret)+3) = *(((u8 *)&dato)+0);
    *(((u8 *) &valret)+2) = *(((u8 *)&dato)+1);
    *(((u8 *) &valret)+1) = *(((u8 *)&dato)+2);
    *(((u8 *) &valret)+0) = *(((u8 *)&dato)+3);
    return valret;
}



/* Función XPS_HWICAP_llseek:                                                */
loff_t XPS_HWICAP_llseek (struct file *puntero_file, loff_t off, int whence)
{
    /* Variable que almacena el valor a devolver (nuevo puntero)             */
    loff_t retval;
    /* Conserva un puntero a la estructura de tipo XPS_HWICAP_dev            */
    struct XPS_HWICAP_dev *el_XPS_HWICAP = puntero_file->private_data;
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_llseek: Estoy ejecutando la función XPS_HWICAP_llseek\
#endif
    switch(whence){
        case 0: /* SEEK_SET  */
                retval = off;
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_llseek: Estoy ejecutando SEEK_SET\n");
#endif
                break;
        case 1: /* SEEK_CUR  */
                retval = puntero_file->f_pos + off;
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_llseek: Estoy ejecutando SEEK_CUR\n");
#endif
                break;
        case 2: /* SEEK_END  */
                retval = el_XPS_HWICAP->numero_de_registros*4 + off;
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_llseek: Estoy ejecutando SEEK_END\n");
#endif
                break;

        default: /* can't happen  */
                return -EINVAL;
        }
```

```
        if (retval < 0) return -EINVAL;
        puntero_file->f_pos = retval;
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_llseek: Salí bien de la función XPS_HWICAP_llseek\n");
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_llseek: El valor devuelto es: %d\n",(int) retval);
#endif
        return retval;
}




/* Función XPS_HWICAP_read: El kernel solicita leer "tamano" bytes al driver. *
 * El driver responde con la cantidad de bytes leidos o cero si se llegó al   *
 * final del archivo.                                                         */
ssize_t XPS_HWICAP_read (struct file *puntero_file, char __user *puntero_usuario, \
size_t tamano, loff_t * puntero_offset)
{
    /* Variable que almacena el valor a devolver (Bytes leidos)               */
    ssize_t retval;
    /* Conserva un puntero a la estructura de tipo XPS_HWICAP_dev             */
    struct XPS_HWICAP_dev *el_XPS_HWICAP = puntero_file->private_data;
    /* Tamaño del buffer calcluado según memoria de caracteres                */
    int cantidad_de_registros = el_XPS_HWICAP->numero_de_registros;

    unsigned int leido;
    /* Variable para el for */
    int k;


#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Estoy ejecutando la función XPS_HWICAP_read\n");
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Tamano = %d\n",tamano);
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: El contenido de puntero_offset es: %lu\n",(long unsi
#endif
    /* Dejamos en tamano el menor entre tamano y XPS_HWICAP_MEMSPACE_SIZE     *
     * porque el driver no piensa leer mas que eso (él es perezoso)           */
    if(tamano>XPS_HWICAP_MEMSPACE_SIZE)
        tamano = XPS_HWICAP_MEMSPACE_SIZE;

    /* Verificamos si se puede sobrepasar el tamaño máximo del buffer y       *
     * limitamos la cantidad de datos leidos en caso de sobrepasarnos         */
    if((*puntero_offset + tamano)>=cantidad_de_registros*4) { // Si sobrepasa límite...
        retval = cantidad_de_registros*4 - *puntero_offset;   // Leeremos lo que podemos
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Lo que nos \
piden sobrepasa la cantidad de registros del XPS_HWICAP, así que solo leemos \
lo que podemos hasta llegar al final. retval = %d\n",retval);
#endif
    } else {                                    // Si no lo sobrepasa...
        retval = tamano;                        // Leemos lo que nos piden
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: La cantidad \
de datos solicitados no excede el numero de registros del XPS_HWICAP, así que \
leeremos los datos solicitados. retval = %d\n",retval);
#endif
    }

#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Voy a leer %d \
datos\n", retval);
#endif

    if(retval > 0) {
        for(k=0; k<retval; k=k+4) {
        /* Estas barreras son necesarias para evitar optimizaciones del       *
         * compilador al darse cuenta que se está escribiendo sobre el mismo  *
         * registro                                                           */
            barrier();
            mb();

        /* Ahora si se envía el caracter al XPS_HWICAP                        */
            leido=ioread32(XPS_HWICAP_iomem_pointer + (*puntero_offset + k));
```

```
#ifdef cambio_endianness
            leido = change_endianness(leido);
#endif
#ifdef depura
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Voy a \
leer la palabra número %d\r\n",k/4);
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Lei el \
valor 0 to 31 = %8.8X\r\n",leido);
#endif
        /* Se actualiza la estructura de datos que contiene el valor de los    *
         * registros                                                           */
            *(el_XPS_HWICAP->intercambio_de_datos + (*puntero_offset + k/4))=leido;
         }

         /* Se realiza el traspaso de la información desde el driver el        *
          * usuario. En caso de salir mal entonces se envía el código del      *
          * error.                                                             */
         if(copy_to_user(puntero_usuario, el_XPS_HWICAP->intercambio_de_datos + (long unsigned int)*puntero_offset
            retval = -EFAULT;
         else {
#ifdef depura
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Se \
leyeron bien los datos\n");
#endif
            /* En caso de que todo salga bien entonces se actualiza el        *
             * puntero con los datos leidos.                                  */
            *puntero_offset += retval;
         }
#ifdef depura
         printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_read: Al salir de \
la función, *puntero_offset = %lu\n\n",(long unsigned int)*puntero_offset);
#endif
     }
     return retval;
}




/* Función XPS_HWICAP_write:                                                  *
 *                  El kernel solicita escribir tamano bytes al driver. La    *
 * función efectua la lectura y devuelve la cantidad de bytes que pudo leer.  *
 * Además actualiza el puntero del usuario puntero3 según los bytes leidos.   */
ssize_t XPS_HWICAP_write (struct file *puntero_file, const char __user *puntero_usuario, \
size_t tamano, loff_t * puntero_offset)
{
     /* Variable para el for */
     int k;
     /* Variable que almacena lo que se va a escribir en el reg del XPS_HWICAP */
     u32 una_palabra;
     /* Variable que almacena el valor a devolver (Bytes escritos)            */
     ssize_t retval;
     /* Conserva un puntero a la estructura de tipo XPS_HWICAP_dev            */
     struct XPS_HWICAP_dev *el_XPS_HWICAP = puntero_file->private_data;
#ifdef depura
     printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Estoy ejecutando \
la función XPS_HWICAP_write\n");
     printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Tamano = %d\n", \
tamano);
     printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: El contenido de \
puntero_offset es: %lu\n",(long unsigned int)*puntero_offset);
#endif
     /* tamano es la cantidad de bytes que se desean escribir y               *
      * XPS_HWICAP_MEMSPACE_SIZE es la cantidad de registros de 32 bits que se *
      * pueden escribir                                                       */
     if(tamano>XPS_HWICAP_MEMSPACE_SIZE)
         retval = XPS_HWICAP_MEMSPACE_SIZE;
     else
         retval = tamano;

     /* Se realiza el traspaso de la información hacia el driver el usuario.   *      * En caso de salir mal entor
*/
     if(copy_from_user(el_XPS_HWICAP->intercambio_de_datos + (long unsigned int)*puntero_offset, puntero_usuario,
```

```
        return -EFAULT;

#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Se trajeron %d \
bytes desde el espacio del usuario al del XPS_HWICAP. \n", retval);
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: El valor del \
puntero virtual del I/O Port as I/O Memory base es: %p\n",                  \
XPS_HWICAP_iomem_pointer);
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Con este \
puntero se harán todas las escrituras al XPS_HWICAP\n");
#endif

   /* Ahora se enviará caracter por caracter la información al XPS_HWICAP     */
    for(k=0; k<retval; k=k+4) {

#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Voy a \
escribir la palabra número %d\r\n",k/4);
#endif
        /* Estas barreras son necesarias para evitar optimizaciones del       *
         * compilador al darse cuenta que se está escribiendo sobre el mismo  *
         * registro                                                           */
        barrier();
        mb();

        /* Ahora si se envía la palabra al XPS_HWICAP                         */
        una_palabra = *(el_XPS_HWICAP->intercambio_de_datos + (*puntero_offset + k/4));
        /* Explicación de porque hay que cambiar el endianness al dato ??     */
#ifdef cambio_endianness
        una_palabra = change_endianness(una_palabra);
#endif
        iowrite32(una_palabra,XPS_HWICAP_iomem_pointer + *puntero_offset + k);

#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Escribi en \
la direccion = %X\r\n",(unsigned int)XPS_HWICAP_iomem_pointer + k);
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Escribi el \
valor 0 to 31 = 0x%08X\r\n",change_endianness(una_palabra));
/*        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Escribi el \
valor 0 to 31 = 0x%8.8X\r\n",una_palabra); */
#endif
    }

#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: Se escribieron bien los datos en el XPS_HWICAP.\n"
#endif

    /* En caso de que todo salga bien entonces se actualiza el puntero con    *
     * los datos leidos.                                                      */
    *puntero_offset += retval;

#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_write: El contenido de puntero_offset al final de la funci
#endif
    return retval;
}







/* Función XPS_HWICAP_ioctl:                                                  */
int XPS_HWICAP_ioctl (struct inode *inode1, struct file *puntero1, unsigned int comando, unsigned long parametro)
{
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: Estoy ejecutando la función XPS_HWICAP_ioctl\n"
#endif
    switch(comando) {
        case COMANDO_0 :
#ifdef depura
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: El comando fue COMANDO_0\n");
```

```
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: Pongo el periférico como salida\n");
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: Escribo 0x00000000 en slv_reg1\n");
#endif
            iowrite32(0x00000000,XPS_HWICAP_iomem_pointer + 4);
            break;

        case COMANDO_1 :
#ifdef depura
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: El comando fue COMANDO_1\n");
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: Pongo el periférico como entrada\n");
            printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_ioctl: Escribo 0xFFFFFFFF en slv_reg1\n");
#endif
            iowrite32(0xFFFFFFFF,XPS_HWICAP_iomem_pointer + 4);
            break;

        default:
            return -ENOTTY;
    }
    return 0;
}




/* Función XPS_HWICAP_open:                                                 *
 *                  Esta función se llama cuando algún proceso del sistema  *
 * operativo abre el nodo /dev/XPS_HWICAP como si fuera un archivo.         */
int XPS_HWICAP_open (struct inode *puntero_inode, struct file *puntero_file)
{
    /* Este puntero almacenará el puntero al XPS_HWICAP_dev que devolverá la  *
     * función container_of y posteriormente se almacena en el puntero_file   *
     * para que pueda ser usado por las demás funciones del módulo            */
    struct XPS_HWICAP_dev *el_XPS_HWICAP;
    /* Almacenará el número de registros del XPS_HWICAP                       */
    int tamano;
    /* Variable del for                                                       */
    int k;
#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_open: Estoy ejecutando \
la función XPS_HWICAP_open\n");
#endif
    /* Esta función busca cual struct XPS_HWICAP_dev contiene                 *
     * puntero_inode->i_cdev, el cual es de tipo cdev.                        */
    el_XPS_HWICAP = container_of(puntero_inode->i_cdev,struct XPS_HWICAP_dev,cdev);
    /* Verificamos que seamos los únicos que han abierto el XPS_HWICAP. Si no *
     * debemos retornar un error que haga que el llamado al sistema se repita */
    if(down_interruptible(&el_XPS_HWICAP->sem))
        return -ERESTARTSYS;
    /* El puntero_file llega a todas las funciones, por ello se aprovecha el  *
     * campo private_data para almacenar el puntero                           */
    puntero_file->private_data = el_XPS_HWICAP;
    /* Pregunta si alguna vez ha sido invocado el open para saber si debe     *
     * asignar los parámetros del periférico                                  */
    if(bandera==0) {
        el_XPS_HWICAP->baseaddress = XPS_HWICAP_BASEADDRESS;
        el_XPS_HWICAP->numero_de_registros = XPS_HWICAP_NUMERO_REGISTROS;
        strcpy(el_XPS_HWICAP->nombre_periferico, XPS_HWICAP_NAME);
        tamano = el_XPS_HWICAP->numero_de_registros;
        el_XPS_HWICAP->intercambio_de_datos = kmalloc(tamano*sizeof(unsigned int),GFP_KERNEL);
        for(k=0;k<tamano;k++)
            el_XPS_HWICAP->intercambio_de_datos[k] = 0;
        bandera=1;
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_open: El mensaje \
guardado en el_XPS_HWICAP->nombre_periferico es: %s\n",                      \
el_XPS_HWICAP->nombre_periferico );
#endif
    } else {
#ifdef depura
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_open: Ya estaba    \
inicializado :)\n");
```

```
#endif
    }
    /* Reserva del puerto... esto se refleja inmediatamente en /proc/ioports  */
    if(!request_region(XPS_HWICAP_BASEADDRESS,4,XPS_HWICAP_NAME))
        printk(PRINTK_LEVEL "XPS_HWICAP_open: No se pudo hacer exitosamente el\
 request del puerto XPS_HWICAP :(\n");

    /* Se mapea la dirección física del puerto en la porción de memoria tipo  *
     * I/O. El tamaño de la reserva es de 64 bytes                             */
    XPS_HWICAP_iomem_pointer=ioremap(XPS_HWICAP_BASEADDRESS,0x00000040);

    return 0;
}







/* Función XPS_HWICAP_release: Esta función libera lo que la función          *
 * XPS_HWICAP_open ha reservado.                                              */
int XPS_HWICAP_release (struct inode *inode1, struct file *puntero_file)
{
    struct XPS_HWICAP_dev *el_XPS_HWICAP = puntero_file->private_data;
#ifdef depura
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_release: Estoy \
ejecutando la función XPS_HWICAP_release\n");
#endif
    /* liberacion de la memoria I/O sobre la que se mapeo el puerto           */
    iounmap(XPS_HWICAP_iomem_pointer);

    /* Liberación del puerto reservado                                        */
    release_region(XPS_HWICAP_BASEADDRESS,4);

    /* Libera el semáforo                                                     */
    up(&el_XPS_HWICAP->sem);
    return 0;
}







/* En esta estructura se definen las operaciones que se van implementar en el *
 * driver                                                                     */
struct file_operations XPS_HWICAP_fops = {
        .owner    = THIS_MODULE,
        .llseek   = XPS_HWICAP_llseek,
        .read     = XPS_HWICAP_read,
        .write    = XPS_HWICAP_write,
        .ioctl    = XPS_HWICAP_ioctl,
        .open     = XPS_HWICAP_open,
        .release = XPS_HWICAP_release
};







/* Esta estructura representa los dispositivos XPS_HWICAP implementados. Para *
 * este caso fue solo uno. La memoria que emplea este dispositivo es          *
 * reservada en XPS_HWICAP_init                                               */
struct XPS_HWICAP_dev *my_XPS_HWICAP_dev;
```

```
/* Función XPS_HWICAP_init: Esta función es llamada cuando se hace insmod y   *
 * reserva los números major y minor y registrar el dispositivo.              */
static int XPS_HWICAP_init(void)
{
    /* Variable usada para recibir el código de error de la función cdev_init */
    int err;
#ifdef depura
    /* Variable del for que repite el mensaje de saludo.                      */
    int k;
#endif
    /* Variable que recibe el valor de la función alloc_chrdev_region         */
    int result;

    /* Reserva dinámica de los números del driver.                            */
    result = alloc_chrdev_region(&mydev,firstminor,count,nombre);
#ifdef depura
    if(result==0)
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_init: Los números \
reservados para el driver fueron:\n Major: %d\n Minor: %d\n", MAJOR(mydev), \
MINOR(mydev));
    else
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_init: Hubo un error \
y los números no se reservaron. El error fue: %d\n",result);
#endif

    /* Reserva memoria en el espacio del kernel para XPS_HWICAP_dev, es       *
     * decir, que inicializa el puntero my_XPS_HWICAP_dev                     */
    my_XPS_HWICAP_dev = kmalloc(sizeof(struct XPS_HWICAP_dev), GFP_KERNEL);
#ifdef depura
    if(my_XPS_HWICAP_dev==NULL)
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_init: Pailas, tocó \
reiniciar, no pude reservar memoria\n");
    else
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_init: Congratulations, \
sos un duro, reservaste memoria para my_XPS_HWICAP_dev, no tocó reiniciar, se \
reservaron %d Bytes\n",(int) sizeof(struct XPS_HWICAP_dev));
#endif
    /* Inicialización del semaforo del periferico                             */
    init_MUTEX(&my_XPS_HWICAP_dev->sem);
    /* Método para reservar la memoria para el cdev. Este método permite que  *
     * el cdev sea parte de nuestra estructura pesonalizada my_XPS_HWICAP     */
    cdev_init(&my_XPS_HWICAP_dev->cdev,&XPS_HWICAP_fops);
    /* Registro del driver en el kernel. Este método no se debe llamar hasta  *
     * no tener todo listo para atender cualquier llamado                     */
    err = cdev_add(&my_XPS_HWICAP_dev->cdev, mydev, 1);
    /* Inicializa algunos campos de la estructura my_XPS_HWICAP_dev           */
    my_XPS_HWICAP_dev->cdev.owner = THIS_MODULE;
    my_XPS_HWICAP_dev->cdev.ops = &XPS_HWICAP_fops;

#ifdef depura
    if(err<0)
        printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_init: Hubo un error\
 al registrar el dispositivo. El error fue: %d\n",err);
    /* Mensaje de saludo para demostrar el uso de los parámetros              */
    for (k=0;k<howmany;k++)
        printk( "XPS_HWICAP_hello: XPS_HWICAP_init: Hola %s\n",whom);
    /* Prueba que permite saber cual proceso lo invoca a uno                  */
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_init: He sido invocado \
por:[%i] %s\n",current->pid,current->comm);
#endif

    /* Se pone la bandera a cero para indicar que el driver nunca ha sido     *
     * inicializado                                                           */
    bandera=0;
    return result;
}

/* Función XPS_HWICAP_exit:                                                   *
                    Esta función es invocada cuando se ejecuta rmmod y debe  *
 * liberar y deshacer lo que la función XPS_HWICAP_init ha reservado          */
static void XPS_HWICAP_exit(void)
{
```

```
    /* Desregistra los números major y minor que le fueron asignados de forma *
     * dinámica*/
    unregister_chrdev_region(mydev,count);
    /* Remueve el dispositivo del kernel                                       */
    cdev_del(&my_XPS_HWICAP_dev->cdev);
    /* Libera el espacio de memoria de los registros del XPS_HWICAP            */
    if(&my_XPS_HWICAP_dev->intercambio_de_datos != NULL && bandera==1)
        kfree(my_XPS_HWICAP_dev->intercambio_de_datos);
    /* Libera la memoria reservada para el struct XPS_HWICAP_dev               */
    kfree(my_XPS_HWICAP_dev);
#ifdef depura
    /* Mensaje de despedida                                                    */
    printk(PRINTK_LEVEL "XPS_HWICAP_hello: XPS_HWICAP_exit: Bye bye, Mundo\n");
    /* Prueba que permite saber cual proceso lo invoca a uno                   */
    printk(PRINTK_LEVEL "XPS_HWICAP_depura: XPS_HWICAP_exit: He sido invocado por:[%i] %s\n",current->pid,current
#endif
}

/* Indica cuales funciones son las que se deben ejecutar al ejecutar insmod y *  * rmmod.
*/
module_init(XPS_HWICAP_init);
module_exit(XPS_HWICAP_exit);

/* Información adicional sobre el módulo.                                     */
MODULE_AUTHOR("William Salamanca y Sergio Abreo.");
MODULE_ALIAS("XPS_HWICAP_template");
MODULE_DESCRIPTION("Driver que maneja un XPS_HWICAP y que sirve como plantilla\
 para controladores de otro tipo de periféricos.");
```

Cuadro de Código I.2: Nueva librería `xil_io`: Rutinas para escribir y leer datos de 32 bits en la memoria principal desde el espacio del usuario.