Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Proyecto Terminal en Ingeniería en Computación

Implementación de un *firewall* utilizando un FPGA

Proyecto que presenta:

Aldama Mejía Genaro

Ochoa Jiménez José Eduardo

para obtener el titulo de:

Ingeniero en Computación

Director de Proyecto:

M. en C. Oscar Alvarado Nava

Mxico, D.F. **Diciembre de 2010**

Resumen

La seguridad es una cuestión de suma importancia para proteger sistemas que mantienen servicios para una gran cantidad de usuarios, así como para proteger información confidencial en redes de computadoras, o para cualquier persona que cuenta con un equipo de computo conectado a una red.

Por lo que para proteger la integridad de estos equipos se recurre a un *firewall*, el cual es un sistema que detecta e impide ataques a una computadora o red de computadoras, filtrando el tráfico de red de acuerdo a una lista de reglas. Dicho sistema es frecuentemente encontrado en *software*, el cual consume recursos como: memoria y capacidad de procesamiento, que bien podrían utilizarse para otro tipo de procesos, y en algunas ocasiones en *hardware* que en muchos casos no se cuenta con los recursos para adquirir lo o éste excede las necesidades reales de una persona común.

El desarrollo tecnológico ha permitido crear dispositivos programables que ofrecen una gran densidad de compuertas para la programación de módulos de *hardware* y que cuentan con procesadores empotrados que ayudan a desarrollar sistemas completos dentro de un mismo chip, aprovechando las características que ofrecen tanto el software como el *hardware* (*Codiseño Hardware-Software*).

Mediante ésta metodología de diseño se desarrollo un sistema *firewall* que permite modificar los criterios de filtrado desde una aplicación en el *host* a proteger, además genera una bitácora con los datos rechazados por el *firewall* y que permite visualizar estadísticas de la red. Los componentes del sistema que hacen el filtrado y la generación de bitácora se realizaron en *hardware* mediante el lenguaje de descripción de *hardware* VHDL y los *drivers* que permiten la comunicación entre el módulo en *hardware* y el resto del sistema se realizaron en software con ayuda del lenguaje C.

En este trabajo se presenta el codiseño *hardware-software* de un sistema *firewall* basado en FPGA, que muestra una solución para la seguridad de un *host* conectado a una red.

Agradecimientos

- A la División de Ciencias Básicas e Ingeniería de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco.

- Al Departamento de Electrónica y de Sistemas

- Al M. en C. Oscar Alvarado Nava, nuestro asesor de proyecto

Dedicatoria

Aldama Mejía Genaro

A mis padres que me ofrecieron su apoyo incondicional, y a Magdalena que se mantuvo siempre a mi lado, sin los cuales no hubiese tenido las oportunidades o la fuerza para seguir siempre adelante.

Ochoa Jiménez José Eduardo

A mis padres y a mis hermanos que son la inspiración que me hace ser mejor persona cada día, y que han sabido brindarme su a apoyo incondicional.

Índice general

Índice de figuras

Indice de tablas

Capítulo 1

Introducción

1.1. Motivación

La gran cantidad de información que posee una persona en la actualidad es muy basta y la mayoría de esa información es privada o de vital importancia. Y debido al crecimiento de Internet y a que la mayoría de las personas cuenta con una computadora conectada a la red, surge la necesidad de proteger la información que es considerada importante de terceras personas que podrían hacer mal uso de dicha información o afectar su integridad. En los últimos años se han hecho presentes gran cantidad de sistemas que pretenden proteger la información, dichos sistemas pueden ser encontrados en su mayoría en *software* y en algunas ocasiones en *hardware* o una combinación de ambos.Tales sistemas que protegen datos en redes de computadoras se les conoce como *firewalls*. Un *firewall* es un sistema diseñado para bloquear o permitir el trafico de una computadora a otra, a través de una serie de normas y criterios.
La desventaja que presentan los sistemas basados en *software* se centra en el uso de recursos, que podrían ser aprovechados para otro tipo de procesamiento y en el caso de los sistemas de *hardware* la desventaja se encuentra en que estos dispositivos no son accesibles para cualquier persona o en su defecto exceden las características que necesitaría un usuario promedio.

Dado que la tecnología ha permitido crear dispositivos programables como los FPGAs, por sus siglas en inglés (*Field-Programmable Gate Arrays*), que nos brindan un conjunto denso de compuertas para la programación de *hardware* y que cuentan con uno o varios procesadores de propósito general. Podemos generar sistemas que conjunten las características del *hardware* y la flexibilidad que nos brinda el *software* (*Codiseño Hardware-Software*), a un bajo costo.
Partiendo de este enfoque nos propusimos hacer un sistema *firewall*, enfocado a personas promedio que necesitan protección no muy elaborada, que no fuera dependiente de los recursos de una computadora para el procesamiento del tráfico de red, y que pudiese ser programado de manera sencilla con reglas hechas de acuerdo al criterio del usuario.

1.2. Objetivos

En este trabajo se presenta la metodología seguida para la implementación de un *firewall* bajo el enfoque del *Codiseño Hardware-Software*, sobre la plataforma FPGA XC2VP30 de la familia Virtex II Pro.

El objetivo principal es el desarrollo de un sistema sobre FPGA que permita la aplicación normas para hacer seguro el tráfico en una red de computadoras, así como una interfaz gráfica que permita al usuario la modificación de los criterios de filtrado y que permita visualizar algunas características de los paquetes que entran y salen de la red.

Para alcanzar el objetivo principal de este trabajo, fue necesario resolver previamente lo siguiente:

- **Modelar la estructura del sistema** El sistema cuenta con diversos módulos, algunos en *hardware* y otros en *software*, por lo que hay que ver el sistema desde un punto de vista de arquitectura para tener definido como es que va a estar organizado.
- **Implementar el Sistema** Una ves estructurado el sistema se tienen las bases para comenzar a implementar los módulos que formaran el sistema.
- **Configurar la plataforma** En este objetivo se configura la tarjeta de desarrollo para tener los componentes necesarios para obtener la funcionalidad completa del sistema.

1.3. Organización del proyecto

El proyecto esta organizado de la siguiente manera: En el capítulo 2 se presenta una introducción al concepto de *firewall* y se explica la manera en que opera un *firewall* implementado con IPTables, En el capítulo 3, se explica el funcionamiento de un *firewall* de filtrado de paquetes que es la base de este proyecto, y se da una breve explicación acerca de el protocolo IP. En el capitulo 4 se encuentra la introducción a la metodología de diseño ocupada para la realización del proyecto. A lo largo del capítulo 5 se presenta el desarrollo del diseño del proyecto siguiendo el proceso de codiseño *hardware*-software, en el capítulo 6 se presentan las pruebas realizadas al proyecto y sus respectivos resultados. Y en el capítulo 7 se presentan las conclusiones a las que se llegaron con la elaboración del proyecto.

Capítulo 2

Firewall

2.1. Introducción

Un *Firewall* es un sistema de contención que funciona dividiendo la red en zonas y conteniendo la actividad dentro de ellas, Figura2.1. Evitando que el trafico no autorizado entre o salga de una zona a través de directivas de seguridad. Una directiva de seguridad especifica las acciones que un usuario o host esta autorizado para llevar a cabo. un *firewall* impide los ataques filtrando el trafico según la directiva de seguridad. Es decir, el cortafuegos acepta el trafico que es consistente con la directiva de seguridad y bloquea el resto del trafico [1].

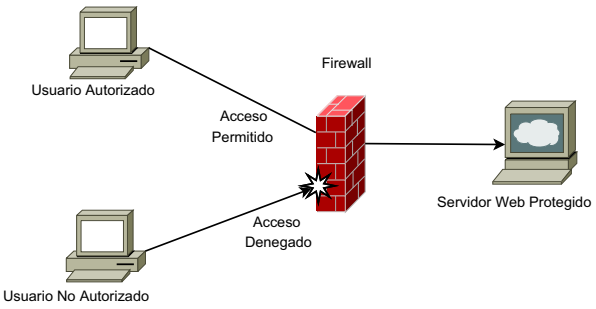


Figura 2.1: Firewall.

Un *firewall* no solo impide los accesos no autorizados, si no también detecta los ataques, registrando los intentos de acceso al *host* y alerta a los administradores cuando uno de tales intentos es sospechoso de ser un ataque.
Un *firewall* puede filtrar el trafico de varias formas. Las más comunes son a través de Dirección IP y servicio. Debido a que el tráfico de red esta etiquetado con un par de direcciones IP que indican el *host* origen y el *host* destino, y que el trafico de red que implica acceder a un servicio esta etiquetado con un numero de puerto que, junto con la dirección IP de destino identifica el servicio.

2.1.1. Tipos de *Firewall*

Los *firewalls* pueden ser clasificados dependiendo del mecanismo que se utiliza para implementarlo, el nivel de la pila TCP/IP sobre el que trabaja el *firewall* y las arquitecturas de red y enrutamiento que se usen [2]. De acuerdo a lo anterior podemos identificar tres tipos principales de *firewalls*: de filtrado de paquetes, de pasarela de aplicación, también llamado *firewall* de host explorado, y de pasarela de circuito del nivel de aplicación, también llamado *firewall* proxy.

Firewall de filtrado de paquetes

Un *firewall* de filtrado de paquetes se suele implementar dentro del sistema operativo, funcionando en las capas de transporte y de red del protocolo de Internet (IP). Protege el sistema realizando las decisiones de enrutamiento después de haber filtrado los paquetes de red basándose en la información contenida en la cabecera IP del paquete.

Este tipo de *firewall* en especifico es el que se implemento en este trabajo por lo que sera detallado en el siguiente capitulo.

Firewall de host explorado

Un *firewall* de host explorado se implementa en los niveles de arquitectura de red y configuración del sistema. Aplica mecanismos de seguridad cuando una conexión TCP o UDP es establecida. Una vez que la conexión se ha hecho, los paquetes pueden fluir entre los anfitriones sin más control. Permite el establecimiento de una sesión que se origine desde una zona de mayor seguridad hacia una zona de menor seguridad. Ademas, el sistema que controla las sesiones puede implementar filtrado de paquetes, tanto en su interfaz interna como en la externa.

Firewall proxy

Un *firewall* proxy suele implementarse como aplicación independiente para cada servicio con el que se desea usar un proxy. Cada aplicación proxy aparece ante el servidor como el programa cliente y ante el cliente como el servidor real. Los programa cliente especiales, o programas cliente configurados especialmente, se conectan al servidor proxy en lugar de a un servidor remoto. El proxy establece la comunicación con el servidor remoto en el lado de la aplicación cliente, después de sustituir la dirección origen del cliente por la suya propia.

2.2. IPTables

Una herramienta que es usualmente ocupada por los usuarios de *Linux* para tener un *firewall* en software, nos da un acercamiento al funcionamiento de un *firewall* y se detalla a continuación.

IPTables es el nombre que se le da a la herramienta mediante la cual un administrador puede definir políticas de filtrado de tráfico de red. IPTables es un componente construido sobre el *Framework* Netfilter disponible en el núcleo de *Linux*, que es una herramienta *firewall* que permite no solo filtrar paquetes, si no también realizar traducción de direcciones (NAT) o mantener registros de log. El proyecto Netfilter no sólo ofrece componentes disponibles como módulos del núcleo sino que también ofrece herramientas de espacio de usuario y bibliotecas.

En sistemas operativos *Linux*, los *firewalls* generalmente se implementan con IPTables, basado en tablas, cadenas y reglas. IPTables tiene seis cadenas principales que se agrupan en tres tablas que son: filter, nat y mangle [1]. Las tablas están compuestas de un conjunto de cadenas y las cadenas están compuestas de una serie de reglas que deben cumplir los paquetes para determinar si son aceptados o no.

Su funcionamiento se describe a continuación: cuando un paquete llega por una interfaz de red, el paquete pasa por las tablas de IPTables y es verificado por cada una de las reglas dentro de las cadenas. Cuando un paquete cumple con todas las reglas, es aceptado y reenviado. En caso de que el paquete no cumpla con alguna regla se rechaza. Este proceso se presenta en la Figura2.2.

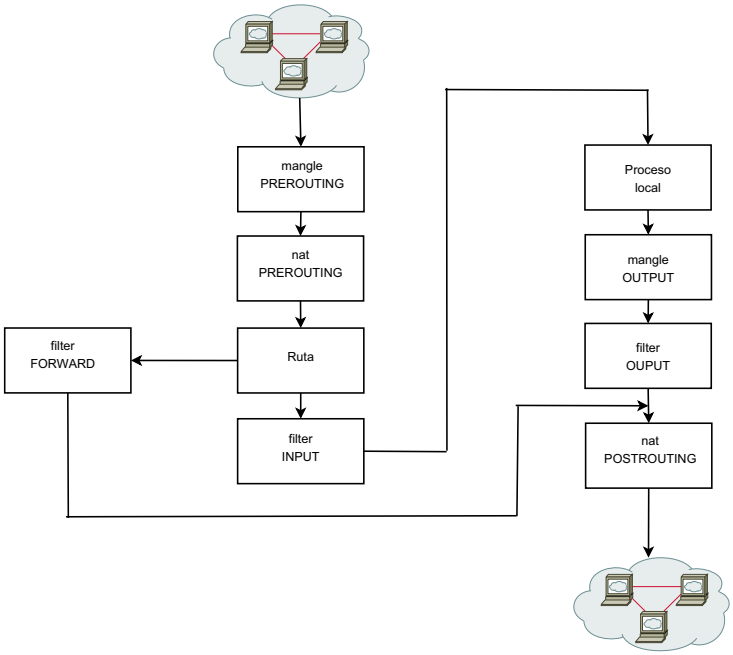


Figura 2.2: La ruta de paquetes IPTables.

2.2.1. Reglas IPTables

Las reglas IPTables siguen la estructura que se muestra en la Figura2.3.

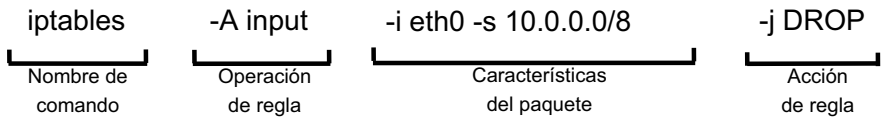


Figura 2.3: La estructura de una regla IPTables.

IPTables permite especificar las siguientes características de paquete relacionadas con la cabecera de paquete IP [1]:

- Protocolo (indicador: -p, i.e. -p udp o -p ! tcp)
- Dirección IP de origen (indicador: -s, i.e. -s 192.168.0/24 o -s ! 10.0.0.0/8)
- Dirección IP de destino (indicador: -d, i.e. -d 192.168.0/24 o -d ! 10.0.0.0/8)
- Interfaz de entrada (indicador: -i, i.e. -i eth0 o -i ! eth0)
- Interfaz de salida (indicador: -o, i.e. -o eth0 o -o ! eth0)
- Indicador de fragmento (indicador: -f)

Ademas, podemos especificar las siguientes características de cabecera de los tipos de paquete indicados:

- Datagramas TCP
  - Puerto de origen (indicador: –sport, i.e. –sport 53)
  - Puerto destino (indicador: –dport, i.e. –dport 53)
  - SYN y otros indicadores TCP
- Datagramas UDP
  - Puerto de origen (indicador: –sport, i.e. –sport 53)
  - Puerto destino (indicador: –dport, i.e. –dport 53)
- Mensajes ICMP
  - Tipo y código ICMP (indicador: –icmp-type, i.e. –icmp-type echo-request)

Capítulo 3

Firewall de filtrado de paquetes

3.1. Introducción

Un *firewall* de filtrado de paquetes consta de una lista de reglas de aceptación y denegación. Estas reglas definen explícitamente los paquetes que se permiten pasar y los que no, a través de la interfaz de red. Las reglas del *firewall* usan los campos del encabezado del paquete, para decidir enrutar un paquete hacia su destino, eliminar el paquete o bloquear un paquete y devolver una condición de error a la maquina emisora. Éstas reglas se basan en la tarjeta de interfaz de red específica y en la dirección IP del host, las direcciones IP origen y destino del nivel de red, los puertos de servicio UDP y TCP de la capa de transporte, los indicadores de conexión TCP, los tipos de mensaje ICMP del nivel de red y en si el paquete es entrante o saliente [2].
Un *firewall* de filtrado de paquetes funciona en las capas de red y de transporte como se muestra en la figura3.1.

El objetivo principal del *firewall* es que el usuario pueda controlar lo que sucede entre Internet y la maquina conectada directamente a Internet. En la interfaz externa que comunica con el mundo exterior el usuario puede filtrar individualmente lo que procede de Internet y lo que sale de la maquina de forma tan precisa y explícita como sea posible.
El *firewall* filtra de manera independiente lo que entra y lo que sale a través de la interfaz que comunica con el mundo exterior. El filtrado de entrada y el filtrado de salida pueden tener reglas completamente diferentes. Las listas de reglas que definen lo que puede entrar y lo que puede salir se llaman cadenas, se llaman cadenas por que se compara un paquete con cada regla de la lista, una a una, hasta que se encuentra una coincidencia o la lista se termina.

Este tipo de *firewall* por si solo no es un mecanismo de seguridad infalible, ya que no todos los protocolos de comunicación se prestan para el filtrado de paquetes. Además éste tipo de filtrado pertenece a un nivel demasiado bajo como para permitir

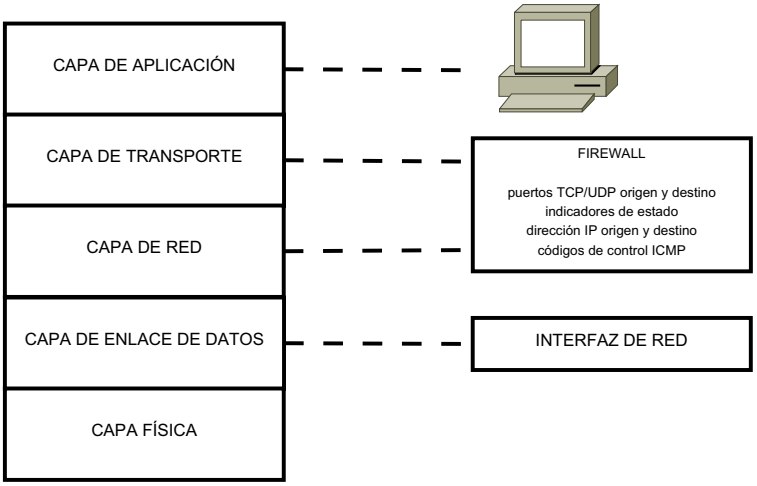


Figura 3.1: Firewall en el modelo de referencia TCP/IP.

autentificación y control de acceso preciso. Éstos servicios de seguridad deben proporcionarse a niveles mas altos. IP no tiene la capacidad de verificar que el emisor es quien dice ser. La única información de identificación disponible en este nivel es la dirección origen del encabezado de paquete IP y este campo puede ser modificado muy fácilmente. Ni la capa de red ni la capa de transporte pueden verificar que los datos de la aplicación son correctos. Sin embargo, el nivel de paquete permite un control más preciso y sencillo sobre el acceso directo a un puerto, el contenido del paquete y los protocolos de comunicación.

Sin filtrado a nivel de paquete, el filtrado de alto nivel y las medidas de seguridad proxy son inútiles o probablemente ineficaces ya que cada nivel de la pila de protocolo de seguridad agrega características que los demás niveles no pueden ofrecer.

3.2. Protocolo IP

El protocolo internet proporciona los medios necesarios para la transmisión de bloques de datos llamados datagramas desde el origen al destino, donde origen y destino son hosts identificados por direcciones de longitud fija. El protocolo internet también se encarga, si es necesario, de la fragmentación y el reensamblaje de grandes datagramas para su transmisión a través de redes de trama pequeña [3].

La cabecera IP se encuentra justo después de la cabecera ethernet. Y Dentro de la cabecera IP pueden encontrarse los protocolos de comunicación de alto nivel como TCP o UDP y dentro de éstos pueden venir telnet, FTP y TFTP respectivamente.

A continuación se presenta un resumen del contenido de la cabecera IP. figura3.2.

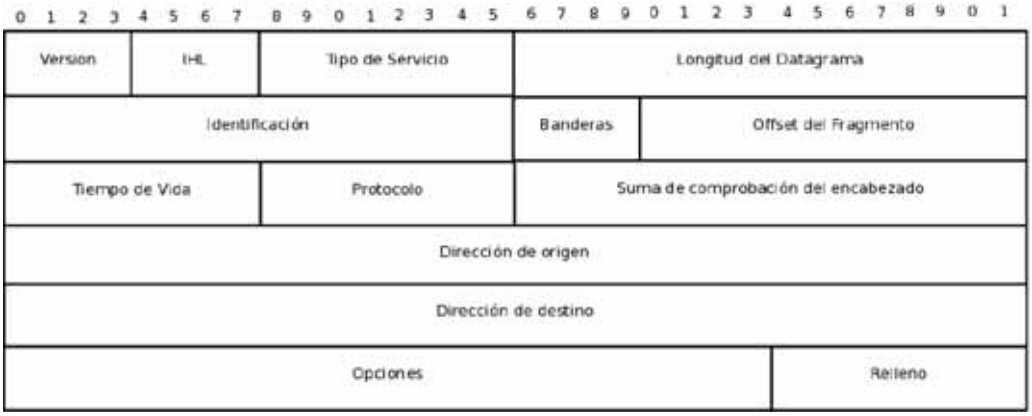


Figura 3.2: Contenido de la cabecera IP.

- Versión (4 bits): Describe el formato de la cabecera IP.
- IHL (4 bits): Internet header length, es la longitud de la cabecera en palabras de 32 bits y por tanto apunta al comienzo de los datos. El valor minimo que puede tomar para que una cabecera sea correcta es de 5.
- Topo de servicio (8 bits): El tipo de servicio indica los parámetros abstractos de la calidad de servicio deseada, como: prioridad, rendimiento, fiabilidad o demora.
- Longitud total (16 bits): es la longitud del datagrama, medida en octetos, incluyendo la cabecera y los datos. Este campo permite que la longitud máxima de un datagrama sea de 65,535 octetos.
- Identificación (16 bits): Es un valor de identificación asignado por el remitente como ayuda en el ensamblaje de fragmentos de un datagrama.
- Flags (3 bits): Son diversos indicadores de control, como: No fragmentar o Mas fragmentos.
- Posición del fragmento (13 bits): Este campo indica a que parte del datagrama pertenece este fragmento. La posición del fragmento se mide en unidades de 8 octetos (64 bits). El primer fragmento tiene posición 0.
- Tiempo de vida (8 bits): Este campo indica el tiempo máximo que el datagrama tiene permitido permanecer en el sistema internet.
- Protocolo (8 bits): Este campo indica el protocolo del siguiente nivel usado en la parte de datos del datagrama internet.
- Suma de Control de Cabecera (16 bits)
- Dirección de Origen (32 bits)

- Dirección de Destino (32 bits)
- Opciones (variable): Las opciones pueden o no aparecer en los datagramas. Deben ser implementadas por todos los módulos IP (host y pasarelas).

3.3. Elección de una directiva de seguridad predeterminada

Cada cadena del *firewall* tiene una directiva predeterminada y una colección de acciones a realizar en respuesta a tipos de mensajes específicos. Cada paquete se compara, uno a uno, con cada regla de la lista hasta que se encuentra una coincidencia. Si el paquete no coincide con ninguna regla, fracasa y se aplica la directiva predeterminada al paquete. Existen dos perspectivas básicas para un *firewall*:

- Denegar todo de forma predeterminada y permitir que pasen paquetes seleccionados de forma explícita.
- Aceptar todo de forma predeterminada y denegar el paso a paquetes seleccionados de forma explícita.

La directiva predeterminada que se recomienda utilizar es la directiva de denegar todo. Esta propuesta facilita la configuración de un *firewall* seguro, pero es necesario habilitar explícitamente cada servicio y la transacción de protocolo relacionada que el usuario desee.
La directiva aceptar todo facilita mucho la configuración y la puesta en marcha de un *firewall*, pero obliga a prever todo tipo de acceso imaginable que quiera deshabilitar. El peligro es que preverá un tipo de acceso peligroso hasta que sea demasiado tarde, o posteriormente habilitará un servicio no seguro sin bloquear primero el acceso externo al mismo.

3.4. Rechazar frente a denegar un paquete

El mecanismo del *firewall* de filtrado de paquetes ofrece la opción de rechazar o denegar los paquetes. Como se muestra en la figura3.3, cuando se rechaza un paquete, el paquete se descarta y se devuelve un mensaje de error ICMP al remitente. Cuando se deniega un paquete, simplemente de descarta el paquete sin ningún tipo de notificación al remitente.

La denegación es casi siempre la mejor elección. Hay tres razones para esto. Primero, enviar una respuesta de error duplica el tráfico de red. Segundo, cualquier paquete al que responda se puede usar en un ataque por denegación de servicio. Tercero, cualquier respuesta, incluso un mensaje de error, ofrece información potencialmente útil que podría ser usada para algún tipo de ataque.

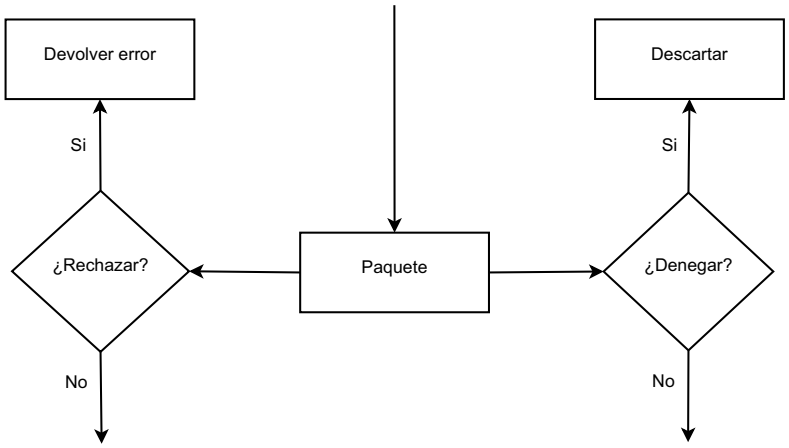


Figura 3.3: Rechazar o denegar un paquete.

Capítulo 4

Codiseño Hardware-Software

4.1. Introducción

La utilización de la metodología de codiseño *hardware-software* tiene como objetivo dividir una aplicación en uno o más módulos *hardware* y en uno o más procesos *software* [6]. La inclusión de una parte de funcionalidad hecha en *hardware* se hace con la finalidad de acelerar la ejecución de la aplicación, ya que las funcionalidades de la aplicación que son candidatas a ser llevadas a *hardware* son las que representan mayor tiempo de procesamiento. La conservación de funcionalidad en *software* se debe a que algunas partes de la aplicación no son criticas desde el punto de vista de la velocidad de ejecución y por lo tanto no implican grandes cambios en ella.

Una de las principales ventajas que se tienen al seguir esta metodología es que, resulta más barato hacer una implementación híbrida de un sistema, es decir, parte en *hardware* y parte en *software*, que hacer una implementación totalmente en *hardware*. Aprovechando la flexibilidad que aporta la parte *software* ejecutándose sobre un procesador de propósito general y aprovechando la eficiencia y el paralelismo que ofrece el *hardware*.

Con el gran avance en la creación de dispositivos programables FPGAs, que cuentan con uno o más procesadores de propósito específico, memorias de datos e instrucciones, y *hardware* dedicado para proporcionar otras funcionalidades. Es posible implementar en un solo dispositivo el particionamiento de una aplicación, dando lugar a los sistemas en un solo chip SoC (*System on a Chip*).

4.2. SoC en un FPGA

Un FPGA es un dispositivo semiconductor que contiene componentes lógicos programables e interconexiones entre ellos. Estos componentes pueden ser programados para duplicar la funcionalidad de puertas lógicas básicas tales como: AND, OR, XOR, NOT, o funciones combinacionales mas complejas.
Los FPGAs son generalmente mas lentos que los ASICs(*Application Specific Integrated Circuit*), no pueden soportar diseños muy complejos y consumen más energía. Sin embargo tienen muchas ventajas como la reducción del tiempo para el desarrollo de aplicaciones, la posibilidad de ser reprogramados cientos de veces con el fin de mejorar el diseño y con ello reducir los costos de ingeniería, diseño y prueba de una aplicación.

Para lograr los objetivos de esta metodología de diseño, es necesario crear interfaces de comunicación entre los módulos de *hardware* programados y el CPU. La interfaz de comunicación entre el CPU y el *hardware* que es conocida como controlador o *driver*, se encarga de convertir datos manejados por el *software* en señales que pueda interpretar el *hardware*. Mientras que la interfaz de comunicación entre el *hardware* y el CPU, es una serie de circuitos que conectan el módulo en *hardware* a uno de los buses del sistema, con el fin de que el módulo pueda responder a las diferentes señales que conforman el protocolo de arbitraje del bus y así poder interactuar con el sistema y de manera especifica con el CPU.

4.2.1. Características del FPGA

La tarjeta de desarrollo que se utilizada para la realización del proyecto es una XUP Virtex-II Pro que provee una plataforma de *hardware* que consiste de una plataforma FPGA de alto rendimiento Virtex-II Pro rodeada por una amplia colección de componentes periféricos que pueden ser usados para crear un sistema complejo.

A continuación se presentan las características de los componentes con que cuenta la tarjeta de desarrollo.

FPGA Virtex II pro

La tabla4.2.1 enumera las características del FPGA Virtex II Pro contenido en la tarjeta de desarrollo[4].

Características	XC2VP30
Slices	13969
Array Size	80 x 46
Distributed RAM	428 kb
Multiplier Blocks	136
Block RAMs	2448 kb
DCMs	8
PowerPC RISC cores	2
Multi-Gigabit Transceivers	8

Tabla 4.1: Características del XC2VP30

System RAM

La tarjeta de desarrollo provee la instalación de un módulo de memoria RAM (*Double Data Rate synchronous Dynamic RAM memory module*) bajo el estándar JEDEC de 184-pin. La tarjeta es compatible con módulos de memoria de 2 GB o menos, organizados en 64 o 72 bits.

Controlador *System ACE Compact Flash*

El controlador provee una interfaz inteligente entre una cadena de destino FPGA y distintas fuentes de configuración. El controlador dispone de varios puertos: el puerto *compact flash*, el puerto de configuración JTAG, el puerto de microprocesador y el puerto de pruebas JTAG. La tarjeta de desarrollo soporta un solo controlador *System ACE*.

Interfaz *Fast Ethernet*

La tarjeta de desarrollo provee un transceptor *Fast Ethernet* que soporta tanto 100BASE-TX como 10BASE-T. Soporta operación *full-Duplex* a 10 Mb/s y 100 Mb/s, con auto negociación y detección paralela. La PHY provee un Media *Independent Interface* (MII) para conectarlo a la 10/100 *Media Access Controler* (MAC) implementada en el FPGA. Cada tarjeta es equipada con un número de serie del silicio que identifica cada tarjeta con un numero serial de 48-bits. Este numero serial se recupera usando el protocolo "1-Wire", y puede se usado como la dirección MAC del sistema.

Puerto serial

La tarjeta de desarrollo provee tres puertos seriales: un puerto RS-232 y dos puertos PS/2. El puerto RS-232 es configurado como un DCE con *hardware handshake* usando un conector serial estándar DB-9. Este conector es usado típicamente para comunicación con un host usando un cable serial estándar 9-pin conectado a un

puerto COM. Los dos puertos PS/2 deberían ser usados para conectar un teclado o un *mouse* a la tarjeta de desarrollo. Todos los puertos están equipados con circuitos *level-shifting*, por que el FPGA Virtex II pro no puede conectarse directamente a los niveles de voltaje requeridos por RS-232 o PS/2.

LEDs, *switches* y *push buttons*

Provee un total de cuatro LEDs para propósitos definidos por el usuario. Cuando el FPGA manda un cero lógico el correspondiente LED se enciende. Un DIP *switch* de cuatro posiciones y cinco *push buttons* son puestos a disposición del usuario. Si el DIP switch es encendido o un push button es encendido, un cero lógico es enviado al FPGA, de otra forma se enviá un 1 lógico.

Salida XSGA

La tarjeta provee un DAC de video y un conector D-sub de 15 pines de alta densidad para soportar la salida XSGA. El DAC de video puede operar con un reloj de pixel de hasta 180 MHz. Esto permite una salida compatible con VESA de 1280 x1024 a 75 Hz y una resolución máxima de 1600 x 1200 a 70 Hz.

Audio CODEC AC97

Un audio CODEC y un amplificador de potencia estéreo son incluidos en la tarjeta de desarrollo para proveer una ruta de audio de alta calidad y provee toda la funcionalidad analógica en in sistema de audio para PC. Cuenta con un ADC estéreo *full-duplex* y un DAC, con un mezclador analógico, combinando las lineas de nivel de entrada, entrada de micrófono y datos PCM.

Interfaz de programación USB 2.0

La tarjeta de desarrollo incluye un microcontrolador embebido USB 2.0 capaz de comunicarse con su *high-speed* (480 Mb/s) o su *full-speed* (12 Mb/s). Esta interfaz es usada para programación o configuración del FPGA Virtex II Pro en modo *Boundary-Scan* (IEEE 1149.1/IEEE 1532).Se pueden seleccionar velocidades de reloj de destino desde 750 kHz hasta 24 MHz. El microcontrolador USB 2.0 es conectado a una PC con un cable USB A-B de alta velocidad.

4.3. Metodología de codiseño *hardware-software*

La metodología de codiseño *hardware-software* puede ser definida como el diseño cooperativo de *hardware* y *software*, esto es, el desarrollo de sistemas heterogéneos, con el objetivo concreto de implementar un sistema con los mínimos costos de diseño, desarrollo y pruebas posibles.

El flujo de diseño de la metodología de codiseño *hardware-software* es presentado en la figura4.1 y es descrito por los siguientes puntos:

- El proceso inicia con las especificaciones del comportamiento del sistema. En este punto se detallan características del sistema como: los datos de entrada, datos de salida, etc.
- El análisis de rendimiento, que se refiere a identificar las partes mas costosas en cuanto a procesamiento que componen la aplicación.
- Una vez localizadas las partes mas costosas de la aplicación se procede a hacer el particionamiento *hardware-software*, en esta parte se determina que partes son factibles para ser llevadas a *hardware* y cuales se mantendrán en *software*. Las partes que tengan mayor consumo de tiempo serán las primeras candidatas en ser llevadas a *hardware*. Para una correcta partición se debe considerar tanto los recursos que ocuparan la parte *hardware* como la parte *software* así como la interacción entre ambas partes partes.
- El diseño de los componentes pueden hacerse de forma separada o quizás de forma simultanea.
- Se compila la parte *software* y sintetiza la parte *hardware*.
- En este punto es de gran utilidad poder simular los componentes de la aplicación por separado, para prever algún comportamiento inesperado.
- En la integración de los componentes, es necesario desarrollar los *drivers* para la sincronización y comunicación de los módulos de *hardware* con la parte *software*. Ademas es necesario desarrollar los componentes de *hardware* que permiten la integración del módulo de *hardware* al sistema de buses.
- La cosimulación de la aplicación ya integrada, con la finalidad de verificar la correcta comunicación entre los componentes.
- Una vez alcanzados los requerimientos de desempeño el proceso ha terminado, de lo contrario será necesario regresar a alguno de los puntos anteriores.

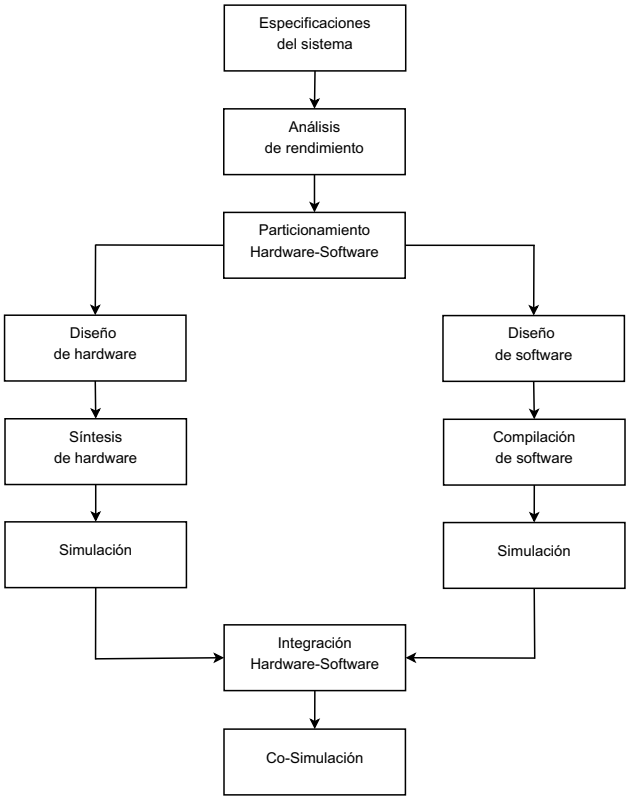


Figura 4.1: Diagrama de flujo del diseño Hardware-Software.

Capítulo 5

Codiseño *Hardware-Software* del *Firewall*

5.1. Introducción

En este capítulo se presenta el codiseño *Hardware-Software* de un *firewall* para el desarrollo de un sistema en un chip (SoC), en una plataforma basada en FPGA. Este sistema permitirá el filtrado del tráfico de red, de acuerdo a directivas de seguridad propuestas por el usuario a partir de una interfaz gráfica, en la que podrá consultar la bitácora de paquetes que fueron rechazados por el filtro, ademas de poder visualizar algunas estadísticas de los paquetes que pasan por el.

El *firewall* a implementar esta basado en el *firewall* de filtrado de paquetes explicado en el capítulo 3 y la sintaxis de las directivas de seguridad esta basada en la sintaxis de las reglas de IPTables que se plantearon en el capítulo 2.

5.2. Especificaciones del sistema

5.2.1. Especificaciones iniciales

El sistema *Firewall* estará contenido en la tarjeta de desarrollo XUP Virtex-II Pro, por lo que la comunicación se hará a través del puerto ethernet de la tarjeta de desarrollo a Internet y del puerto USB de la tarjeta de desarrollo al puerto USB de la computadora a ser protegida por el sistema *firewall*, figura 5.2. El sistema debe obtener los paquetes de red de una de las interfaces de la tarjeta, extraer las partes del encabezado IP a ser filtradas para posteriormente filtrar el paquete y dependiendo de las directivas de seguridad del componente de filtrado ser aceptado o denegado. Una vez que el paquete ha sido aceptado es pasado a la otra interfaz para completar la comunicación, en caso de que el paquete haya sido denegado este sera almacenado en una memoria externa al FPGA, figura 5.1.

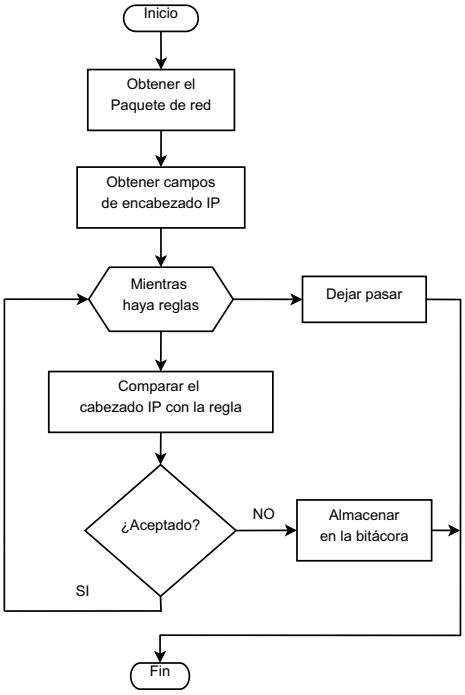


Figura 5.1: Diagrama de flujo del sistema de filtrado.

La interfaz gráfica estará contenida en la computadora a ser protegida, en ella el usuario podrá modificar las directivas de seguridad, esta funcionalidad primero obtendrá los parámetros para crear la regla a través de la interfaz de usuario, posteriormente la regla formada se pasara al sistema de *firewall* por medio del puerto RS-232. Desde la interfaz de gráfica el usuario podrá consultar la bitácora de paquetes denegados por el *firewall*.

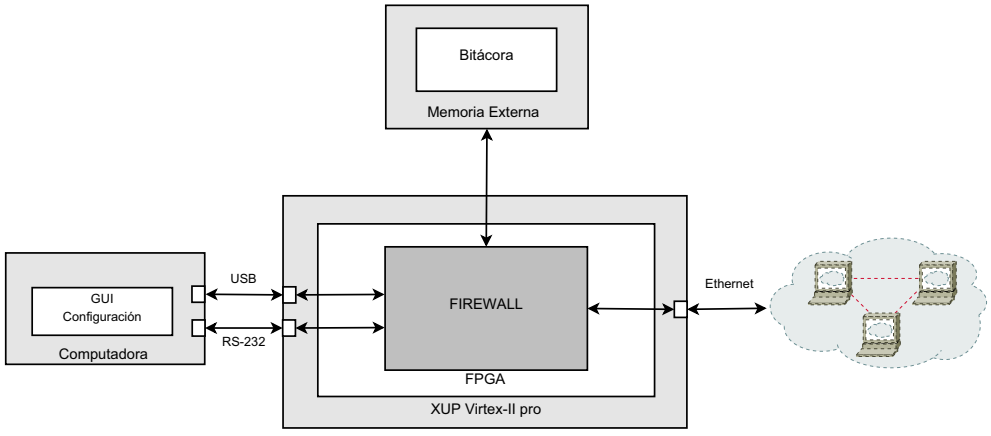


Figura 5.2: Diagrama inicial del dispositivo.

5.2.2. Especificaciones finales

Dado que parte de las especificaciones iniciales no pudieron ser alcanzadas debido a que las características de la tarjeta y su funcionalidad no fueron favorables al proyecto, se decidió modificar ligeramente las características de funcionamiento de la aplicación. La descripción de los contratiempos con la tarjeta de desarrollo que provocaron el cambio en las especificaciones se detallaran durante este capitulo.

El sistema *firewall* tendrá la misma funcionalidad establecida en las especificaciones iniciales, solo que se tomara en cuenta la parte de comunicación dada por el puerto ethernet de la tarjeta de desarrollo y el puerto ethernet de la computadora. Y la bitácora del sistema de filtrado estará implementada como parte del sistema de *firewall*, figura 5.3. La interfaz gráfica de usuario tendrá las mismas características que se establecieron en las especificaciones iniciales. Dado que se modificaron las especificaciones iniciales se decidió que el sistema contara con una funcionalidad mas, la cual consiste en mostrar algunas características de los paquetes que pasan por el sistema de *firewall*.

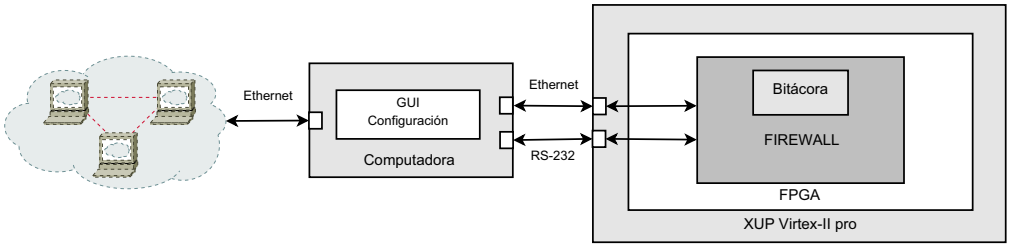


Figura 5.3: Diagrama final del dispositivo.

5.3. Particionamiento Hardware-Software

El sistema completo tiene una parte indiscutible a realizarse en software que es la interfaz gráfica. El funcionamiento de la interfaz no es muy compleja ya que solo se encarga de hacer la traducción de las reglas hachas por el usuario a valore que pueda interpretar el sistema *firewall* y mostrar la bitácora generada por el *firewall*.

En cuanto al sistema que compone el *firewall* así como la comunicación de este con la interfaz de red o con el puerto RS-232. Solo el sistema *firewall* tiene que procesar los paquetes provenientes de la interfaz de red, para poder tomar la decisión de aceptarlo o denegarlo, como se observa en la figura 5.1.

Dado lo anterior se decidió mantener en *software* la interfaz de usuario así como la comunicación del puerto ethernet y el puerto RS-232. Y la única parte a llevar a *hardware* es el sistema *firewall*, lo que implica diseñar el *hardware* necesario para que

almacene las directivas de seguridad, haga el filtrado y maneje la bitácora de paquetes denegados.

5.4. Diseño de *Hardware*

El diseño del sistema *firewall* será explicado bajo la metodología top-down, la cual fue utilizada para hacer el diseño del *hardware*. Esta metodología propone la formulación inicial de un modelo sin especificar detalles, posteriormente el modelo es dividido en partes, las cuales son redefinidas cada vez con mayor detalle hasta que la especificación completa es lo suficientemente detallada para validar el modelo.

Pese a que la comunicación de red no se hizo completamente, el diseño del *firewall* contempla el funcionamiento para soportar las dos interfaces de red, por lo que si en algún momento se decide implementar en una tarjeta de desarrollo que cumpla con los requisitos necesarios, se podrán lograr las especificaciones iniciales.

A continuación se realizará una descripción de los componentes que forman parte del *firewall* escritos en VHDL. Este está diseñado para procesar simultáneamente tanto los paquetes que ingresan a la computadora desde la red (señales "in"), como los paquetes que salen desde la computadora hacia la red (señales "out").

5.4.1. Componente Completo

Este componente escrito en VHDL, contiene todos los componentes utilizados por el *firewall* para ser instanciados en el FPGA. Se encarga de recibir los datos de los encabezados IP de los paquetes de red, así como las directivas de seguridad que serán utilizadas para determinar si dichos paquetes son aceptados o no. El tamaño de entradas y salidas se ha ajustado para que pueda integrarse correctamente al bus de datos del FPGA añadiendo nuevos elementos para su manejo. Así mismo se ha añadido un registro de interrupciones para controlar el *firewall* vía *software*. Este componente también será el encargado de contener una bitácora con un resumen de los paquetes que han sido denegados.

El componente completo cuenta con las siguientes entradas y es presentado en la figura 5.5:

- clk, recibe los pulsos del reloj.
- reset se encarga de reiniciar la bitácora.
- interrup contiene un vector que será utilizado para el control de los componentes desde el sotfware, tanto para la habilitación del filtrado, la programación de las tablas de reglas y la utilización de la bitácora, figura 5.4.

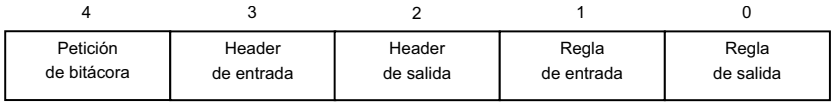


Figura 5.4: Registro de Interrupciones.

- Las entradas H contienen la información extraída de los encabezados de los encabezados IP, las entradas H_in, del 1 al 3 son las que contienen la información de los paquetes que ingresan desde la red hacia la computadora, las entradas H_out, también del 1 al 3, son para los encabezados de los paquetes que salen desde la computadora hacia la red.
- Las entradas Reg del 1 al 3 contienen las reglas que se van a programar dentro del *firewall*.
- La entrada dir contiene la dirección del registro en dónde se va a cargar la regla entrante.
- La salida bitac, contiene la salida de los datos procesados en la bitácora.
- Las salidas rdy_in y rdy_out contienen la información acerca de los resultados obtenidos del firewall.

Las entradas H y Reg tienen un tamaño de 32 bits con el fin de que se puedan ajustar al tamaño del bus de datos del FPGA, posteriormente, estos datos se integraran en datos de encabezado y reglas de tamaño completo de 88 y 93 bits respectivamente.

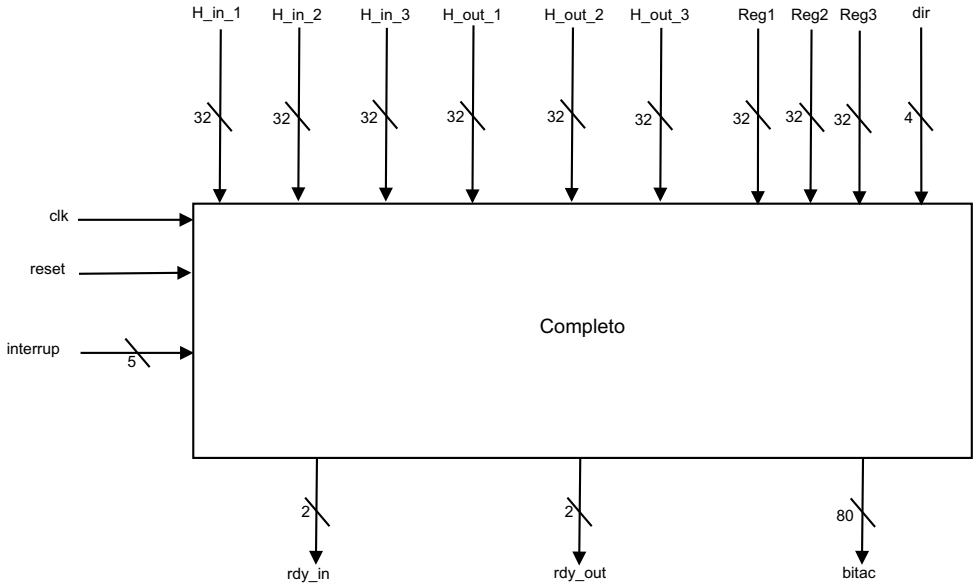


Figura 5.5: Componente completo.

Internamente Completo consta de 3 componentes y su interconexión se muestra en la figura 5.6:

- Una unidad de carga que se encarga de tomar los datos y las reglas, divididas en fragmentos de 32 bits, y unirlos a los tamaños correspondientes, de 88 y 93 bits correspondientes.
- La unidad de *firewall* que es la encargada de realizar las comparaciones entre los datos extraídos de los encabezados de los paquetes IP, y las reglas definidas por el usuario, así mismo se encargará de informar cuáles son los resultados obtenidos.
- La unidad de bitácora se encarga de tomar la información de los paquetes IP, y según el resultado obtenido hará un conteo de los paquetes rechazados y mostrará información sobre los mismos.

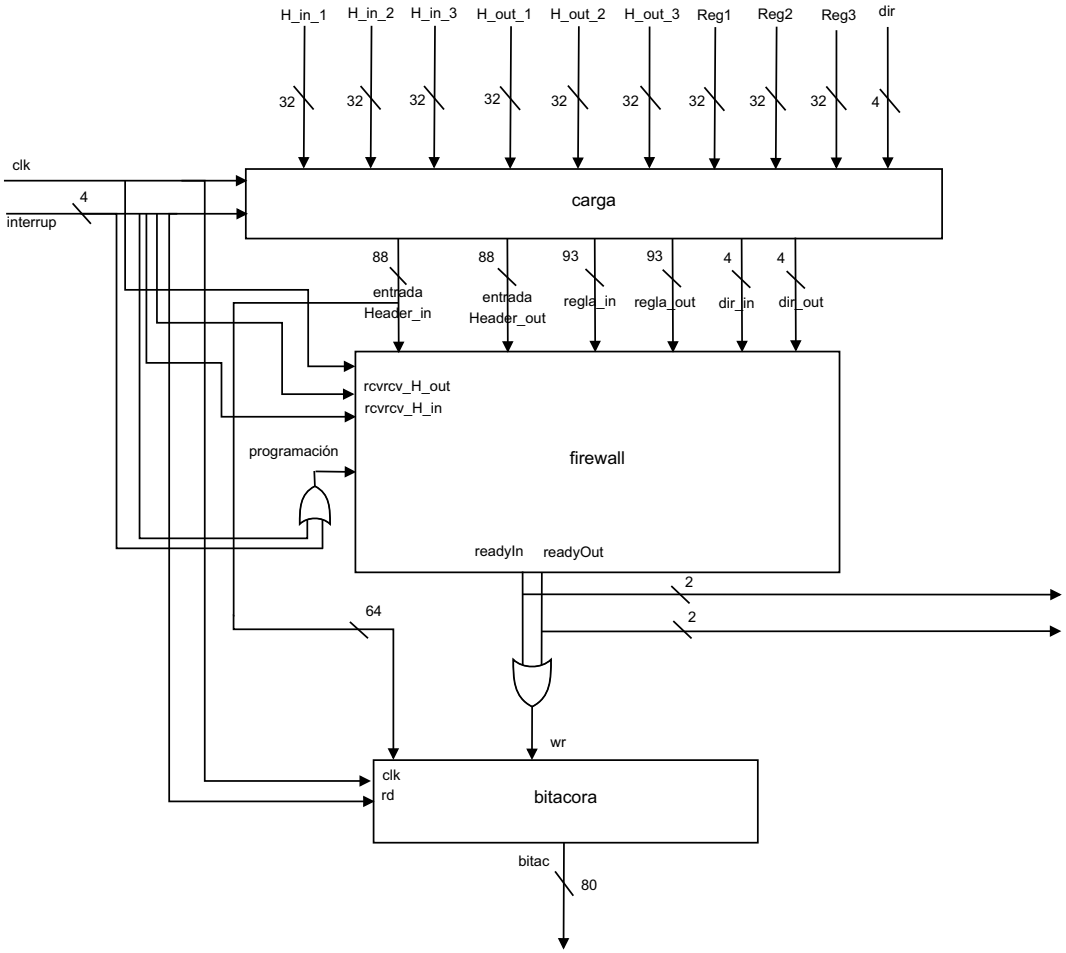


Figura 5.6: Conexiones internas del componente completo.

5.4.2. Descripción de las reglas

Cada regla consiste en una cadena de 93 bits que contiene la información de los campos que van a utilizarse como base para la comparación, es decir se buscará si el el encabezado coincide o no con lo especificado por las reglas.
Los campos a comparar serán el puerto de destino del paquete, el protocolo del paquete, la dirección ip de destino y la dirección ip de origen.
Entre cada uno de los campos existe un bit que habilita o deshabilita la comparación en ese campo, es decir si la regla definida por el usuario no especifica el campo, o no lo utiliza, el bit correspondiente indicará que dicho campo no sea tomado en cuanta al momento de realizar las comprobaciones. Además existe un bit (Aceptar) que indica si la regla es para aceptar o rechazar un paquete, la estructura de la regla se muestra en la figura 5.7.

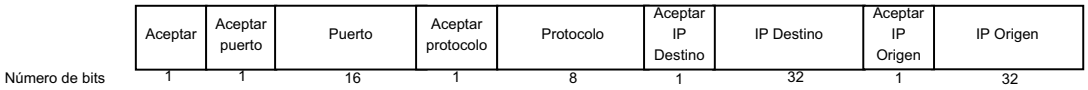


Figura 5.7: Estructura de la reglas.

5.4.3. Componente carga

El componente se encarga de tomar los fragmentos de los datos y de las reglas, y unirlos en datos y reglas completos, depende principalmente del contenido del registro de interrupciones, el cual le indicará a la unidad de carga, cuales son las entradas que debe de tomar, y a que salida las debe de dirigir.
El funcionamiento se describe con la siguiente figura 5.8:

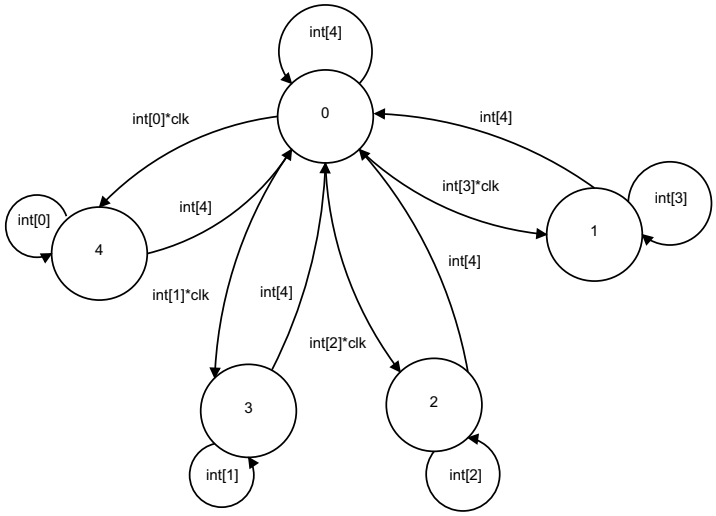


Figura 5.8: Máquina de estados del componente carga.

En este caso las señales int corresponden a los elementos del vector de interrupciones.

1. Estado 0:Las señales H_in, H_out, R_in, R_out, D_in, D_out se cargan con '0' Mientras el elemento [4] del vector de interrupción esté con valor de '1', la máquina permanecerá en este estado, Si ocurre un cambio en el reloj con valor de 1, y:
   - El elemento [3] del int adquiere un valor de 1, pasa al estado 1.
   - El elemento [2] del int adquiere un valor de 1, pasa al estado 2.
   - El elemento [1] del int adquiere un valor de 1, pasa al estado 3.
   - El elemento [0] del int adquiere un valor de 1, pasa al estado 4.
2. Estado 1: Concatena los bits que están el las entradas H_in 1, 2 y 3, y los asigna a la entrada datos_in del *firewall*, en este proceso se pierden los 8 bits más significativos. De H_in_1. Regresa al estado 0 cuando el elemento[4] del vector de interrupciones cambia a '1'.
3. Estado 2: Concatena los bits que están el las entradas H_out 1, 2 y 3, y los asigna a la entrada datos_out del *firewall*, en este proceso se pierden los 8 bits más significativos. De H_in_1. Regresa al estado 0 cuando el elemento[4] del vector de interrupciones cambia a '1'.
4. Estado 3: Concatena los bits que están el las entradas Reg 1, 2 y 3, y los asigna a la entrada reglas_in del *firewall*, en este proceso se pierden los 3 bits más significativos. De Reg1. Regresa al estado 0 cuando el elemento[4] del vector de interrupciones cambia a '1'.
5. Estado 4: Concatena los bits que están el las entradas Reg 1, 2 y 3, y los asigna a la entrada reglas_out del *firewall*, en este proceso se pierden los 3 bits más significativos. De Reg1. Regresa al estado 0 cuando el elemento[4] del vector de interrupciones cambia a '1'.

5.4.4. Componente bitácora

La bitácora en un principio estaría en una memoria externa al FPGA, la memoria donde se almacenaría la bitácora es una memoria Compact flash, accedida por medio del controlador System ACE, que proporciona la tarjeta de desarrollo. Debido a que la memoria Compact flash no pudo ser accedida para leer o escribir archivos, aunque se cambio de memoria, de sistema de archivos entre FAT12 y FAT16, y se siguieron las soluciones encontradas en la pagina del fabricante de la tarjeta de desarrollo. se decidió poner la bitácora como un componente mas de *hardware*.

Este componente se encarga de llevar un conteo de los paquetes que son denegados por el *firewall*. La bitácora agrupa los paquetes denegados según sus direcciones de

origen y de destino, y cuenta cuantas veces un paquete con esas características se ha denegado. Utiliza las salidas de control del *firewall* para determinar que paquetes se denegaron y entonces contarlos.

Consta de una entrada para el reloj, clk; una entrada que indica en que momento se debe de escribir a la salida del componente,rd; la entrada wr indica el momento en el cual de deben de cargar los datos del paquete en la bitácora; la entrada reset sirve para borrar los datos de la bitácora. La entrada datos_in contiene los campos de dirección origen y destino extraídos de los encabezados IP de los paquetes, se utiliza esta información para clasificar los paquetes denegados. La salida datos_out contiene la información de los paquetes denegados, así como el número de veces que ha sido denegado, figura 5.9.

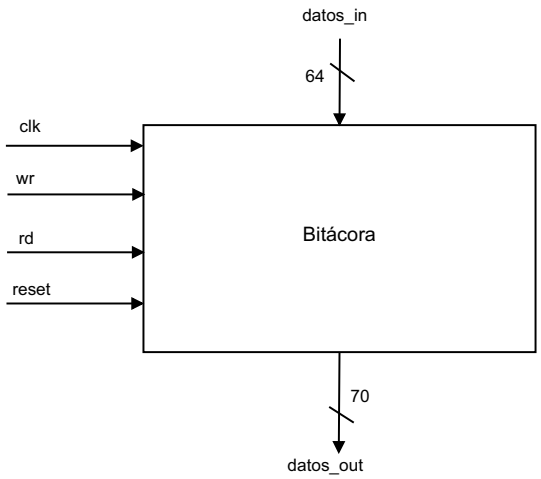


Figura 5.9: Componente bitácora.

5.4.5. Componente firewall

Este componente en VHDL, incorpora el filtro y las tablas de reglas. Su objetivo es recibir los encabezados IP de los paquetes, así como recibir y almacenar las reglas de filtrado.
Los datos de los encabezados se compararán con las reglas almacenadas en las tablas. Dependiendo del resultado obtenido tras la comparación, el componente indicará por medio de señales, cual será la acción a tomar, si el paquete correspondiente debe de ser denegado, o se debe de permitir su paso.

El componente *firewall* consta de las siguientes señales de entrada y se muestra en la figura 5.10:

- Una señal de reloj, clk.

- Dos señales de habilitación, rcv_H_in y rcv_H_out, para indicar en qué momento se deben de iniciar las comparaciones entre los datos y las reglas almacenadas.
- La señal programación indicará en qué momento serán reemplazadas las reglas.
- Las entradas datos_in y datos_out contienen los datos tomados de los encabezados de los paquetes IP, los que ingresan hacia la computadora y los que salen desde la misma, respectivamente.
- Las entradas reglas_in y reglas_out, contienen los datos de las reglas que serán aplicadas a los paquetes de entrada y de salida respectivamente, y serán almacenadas en las tablas de reglas.
- Las entradas dir_in, y dir_out contienen las direcciones de los registros de la tabla de reglas, en dónde se almacenarán las reglas correspondientes cuando estas deban de ser reemplazadas.
- Las salidas readyIn, y readyOut, indican cual será la acción a realizar con los paquetes, si serán descartados o si se permitirá el paso.

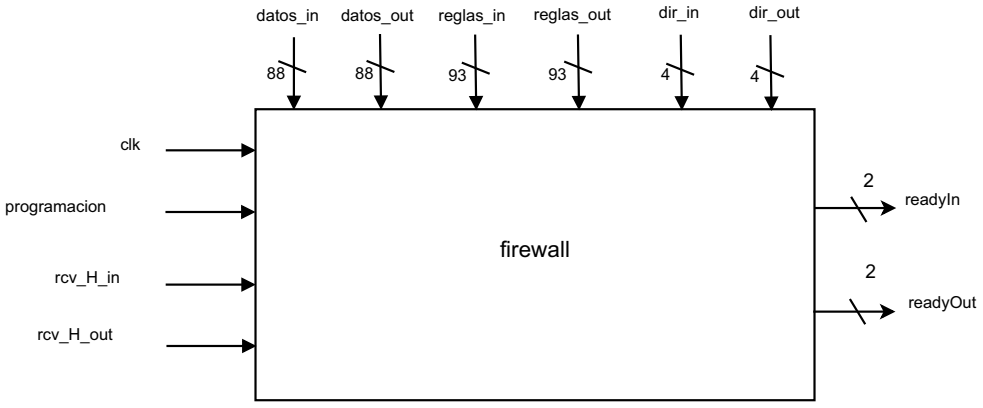


Figura 5.10: Componente *firewall*.

Internamente el *firewall* consiste de dos tablas de reglas conectadas a un filtro. Las tablas de reglas reciben los datos de las nuevas reglas, y la dirección del los registros en donde estas serán almacenadas. Hay una tabla de reglas para los paquetes entrantes (in) y una tabla de reglas para los paquetes salientes (out). Las tablas se programan al habilitar la señal de programación. Así mismo las tablas de reglas envían una a una las diferentes reglas de filtrado hacia el modulo de filtro, así como la dirección máxima utilizada, para conocer cuantas reglas existen en cada tabla. El filtro recibe los datos del encabezado los compara con las reglas una a una hasta determinar si el paquete debe o no se ser denegado para lo cual envía la señal correspondiente a través de las señales de control. Esto se puede observar en la figura 5.11.

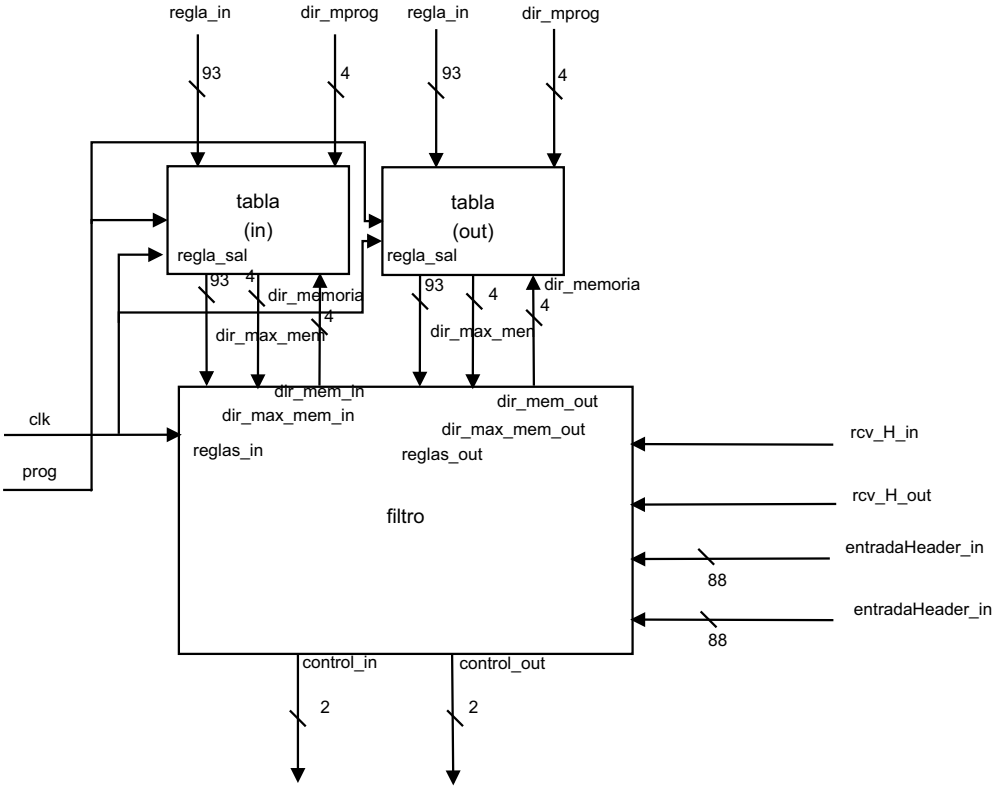


Figura 5.11: Conexiones internas del componente *firewall*.

5.4.6. Componente Tabla

Este componente escrito en VHDL se encargará de almacenar en sus registros internos las reglas que serán utilizadas para la comparación de los datos de los encabezados y determinar si los encabezados IP de los paquetes correspondientes serán o no aceptados.
La tabla de reglas contiene un total de 10 registros internos para su utilización. Se utilizaron 10 registro ya que se especificó que el numero máximo de reglas sería de 10, sin embargo este número esta repartido entre las reglas de entrada y de salida. Como existe la posibilidad de que el usuario no defina reglas entrantes o salientes cada tabla contiene espacio para el número máximo posible de reglas.

El *firewall* dispone de dos tablas de reglas, una que contiene las reglas para los paquetes entrantes (in),y otra que contiene las reglas para los paquetes salientes (out), el componente se puede ver en la figura 5.12.

Las entradas están compuestas por:

- Una señal de reloj, clk.
- Una señal que indica que el componente va a recibir reglas para ser programado, prog.

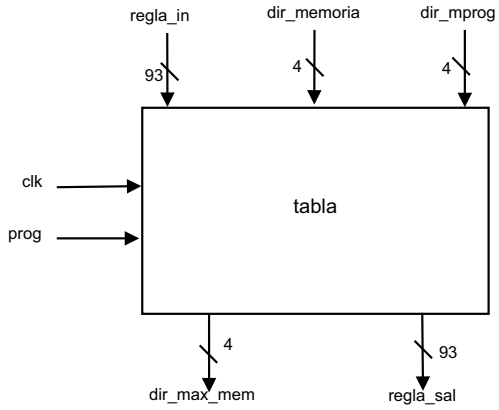


Figura 5.12: Componente tabla.

- La entrada regla_in indica el registro que contendrá la regla que será cargada en la tabla.
- La entrada dir_mprog, indica la dirección de memoria en donde será cargada la regla actual, mientras el componente se encuentre en el modo de programación.
- La entrada dir_memoria, indica al componente la dirección de memoria de dónde se tomará la regla que será enviada, esto ocurre cuando ya no se este en el modo de programación.
- La salida regla_sal, contiene la regla almacenada en la dirección especificada por dir_memoria, para su uso por el filtro.
- La salida dir_max_mem contiene el valor de la dirección máxima en dónde se encuentra una regla, esto con el fin de que la maquina de estados tenga información sobre cuántas reglas hay.

La tabla de reglas está compuesta por 10 registros de 93 bits cada uno. Estos registro contendrán las reglas que serán utilizadas para la comparación de los datos del encabezado para su utilización por el filtro. La tabla incluye un decodificador para la dirección de memorias, y un multiplexor para elegir entre la dirección para la programación y la dirección para la lectura. Además incluye un registro con la última dirección de programación, la cual se enviará como la dirección máxima de memoria, la estructura se muestra en la figura 5.13.

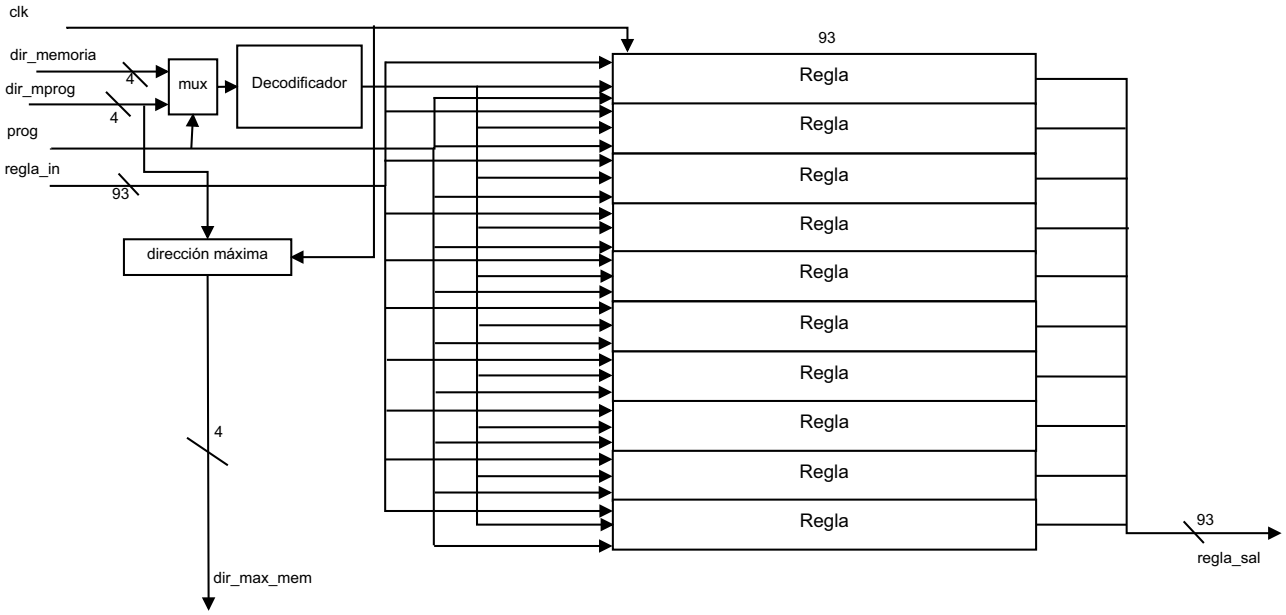


Figura 5.13: Estructura interna del componente tabla.

5.4.7. Componente Filtro

Este componente es el que se encarga de tomar los datos de los encabezados IP de los paquetes, y las reglas contenidas en las tablas para que puedan llevarse a cabo las comparaciones correspondientes, con el fin de determinar si los datos coinciden con las reglas, y por lo tanto, si el paquete correspondiente debe o no de ser denegado. Ya que el *firewall* debe de operar simultáneamente con paquetes entrantes y paquetes salientes, cuenta con un conjunto de entradas y de salidas para dicho procesamiento. El componente se muestra en la figura 5.14.

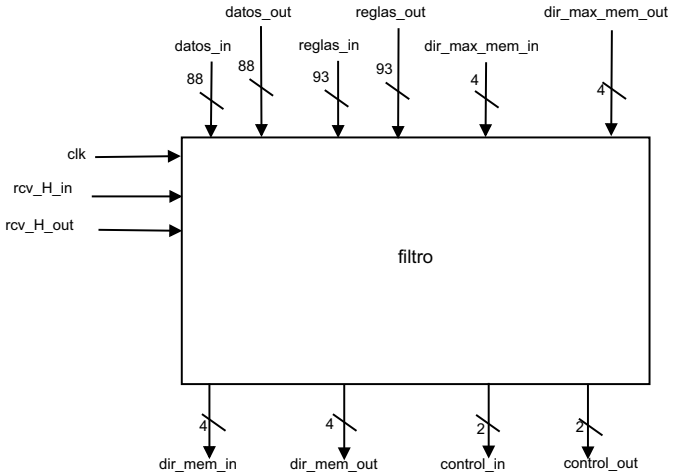


Figura 5.14: Componente filtro.

Este componete cuanta con una entrada de reloj, clk, así como las entradas rcv_H_in y rcv_H_out, las cuales servirán para habilitar las máquinas de estados encargadas de realizar las comparaciones.
Las entradas datos_in y datos_out contienen los datos extraídos de los encabezados de los paquetes IP.
Las entradas reglas_in y reglas_out contienen las reglas contenidas por la tabla de reglas y con las cuales se deberán de comparar los datos.
Las entradas dir_max_mem_in y dir_max_mem_out contienen la dirección de memoria con la última regla para que las máquinas de estados las puedan utilizar.
Las salidas dir_mem_in y dir_mem_out le indican a la tabla cual debe de ser la siguiente regla a enviar.
Las salidas control_in y control_out indican que es lo que se debe hacer con el paquete detenido.

El filtro esta compuesto esencialmente por dos máquinas de estados, una para procesar los datos que van desde la red hacia la computadora (in) y la otra se encarga del procesamiento de los paquetes que van desde la computadora hacia la red (out). Esto con el propósito de llevar a cabo de forma simultánea el procesamiento de los paquetes en ambas direcciones.
Con la excepción del reloj compartido, cada máquina de estados tiene su propio conjunto de entradas y de salidas, por lo que es posible que trabajen de forma completamente independiente, la estructura interna del componente se muestra en la figura **??**.

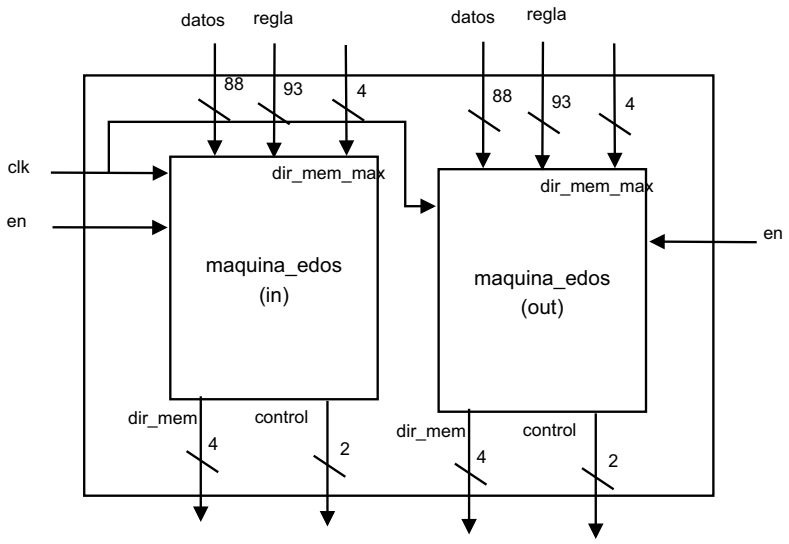


Figura 5.15: Estructura interna del componente filtro.

5.4.8. Componente Maquina de Estados

Es el componente principal del *firewall*, ya que es el encargado de recibir tanto los datos como las reglas, efectuar las comparaciones basadas en las banderas de las reglas, y determinar si los datos corresponden a un paquete que debe o no de ser denegado, figura 5.16.

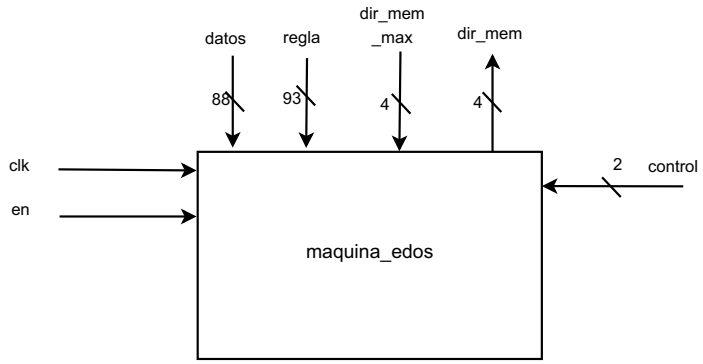


Figura 5.16: Componente máquina de estados.

Este proceso se realizará mediante una máquina de estados y cuenta con las siguientes salidas:

- La máquina de estados tiene un reloj, clk.
- Una señal para habilitar su funcionamiento, en.
- La entrada datos, contiene los campos extraídos del encabezado que serán utilizados para la comparación con la regla, la cual proviene de la entrada regla.
- La entrada dir_mem_max contiene la dirección de la última regla en la tabla, ya que la máquina de estados necesita saber en qué momento ha comparado los datos del encabezado con todas las reglas existentes.
- La salida dir_mem, indica cual es la la dirección de la siguiente regla que será solicitada por la máquina de estados.

La salida control son señales de control que indican si los paquetes detenidos deben o no de denegarse. El componete maquina_edos consta principalmente de un conjunto de registros que almacenan las entradas datos y reglas, y datos adicionales requeridos para el funcionamiento de la misma, así como las compuertas lógicas necesarias para hacer comparaciones entre dichos registros, o entre subconjuntos de dichos registros, así como una unidad que incremente la dirección de la memora de la cual se ha tomado la regla actual, con el fin de solicitar la siguiente, figura 5.17.

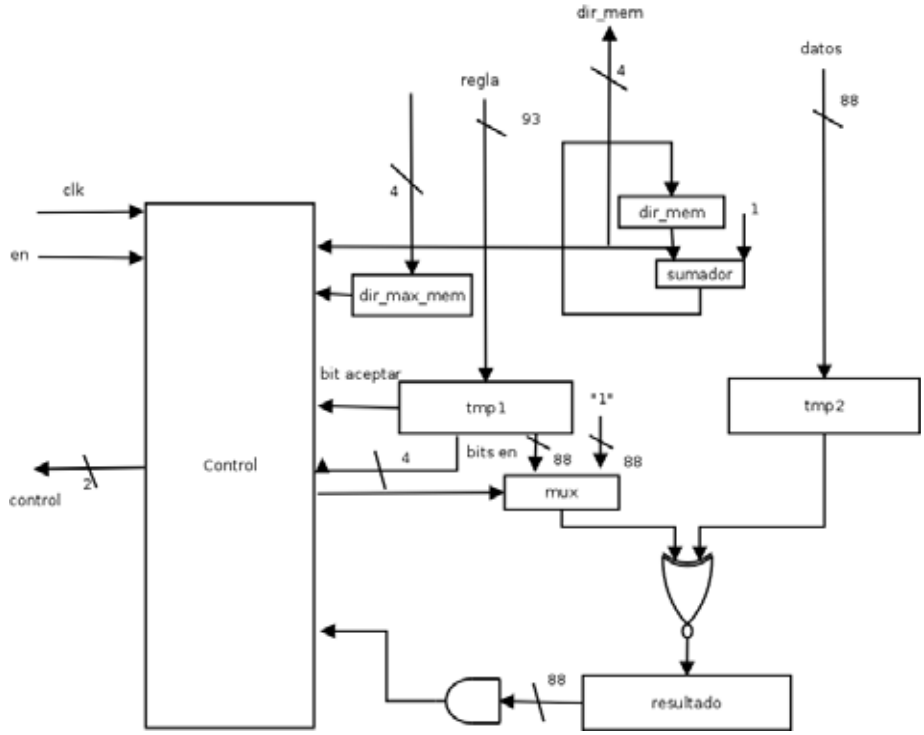


Figura 5.17: Estructura interna del componente maquina de estados.

Todo esto está controlado por una Unidad de Control cuyo funcionamiento está descrito por el diagrama de flujo de la figura 5.18.

La comparación se lleva a cabo de la siguiente forma:

1. Mientas se mantenga el flujo de paquetes hacia el *firewall* se ingresarán uno de ellos hacia el *firewall* y este será detenido mientras se realiza el filtrado.

2. Se extrae el encabezado de los paquetes para que este sea comparado con las reglas programadas en el *firewall*.

3. El proyecto tiene una capacidad para 10 reglas. Comienza a comparar los datos del encabezado con cada una de la reglas.

4. Si la información del encabezado coincide con la presente en la regla, es decir, si la regla aplica a ese paquete en particular, se procede a comprobar si la regla en cuestión permite o no el paso del paquete.

5. Si la regla no permite al paquete, este se desecha y se carga un paquete nuevo.

6. Si la regla permite al paquete, continúa. Si el el encabezado del paquete no coincide con la regla cargada, se mantiene el paquete y se procede a probar con otra regla.Si ninguna de las reglas de la tabla aplica al paquete, éste es aceptado.

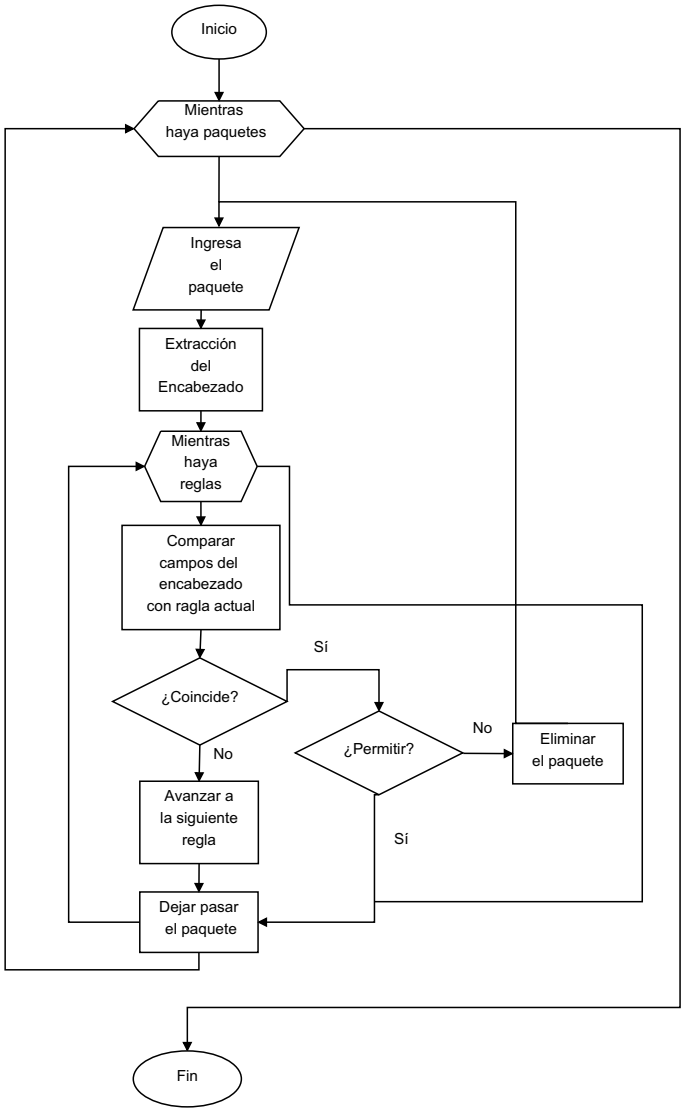


Figura 5.18: Diagrama de flujo de la maquina de estados.

Para su implementación en VHDL, fue modelado como una maquina de estados, la que se presenta en la figura 5.19.

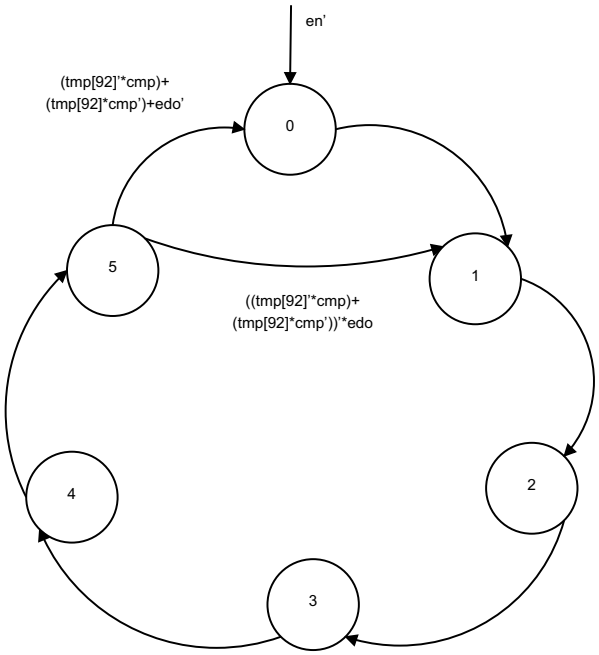


Figura 5.19: Maquina de estados del control del componente maquina de estados.

A continuación se detalla cada uno de los estados:

1. Estado 0: La dirección de memoria a solicitar cambia a 0. Pasa al estado 1.

2. Estado 1: La variable que contiene el resultado de la variable que indica si se ha alcanzado la última regla (edo) adquiere valor de 0. Se solicita a la tabla de reglas la dirección de memoria actual. Pasa al estado 2.

3. Estado 2: La señal de control adquiere el valor de "00". Se carga la regla en el registro temporal 1. Se cargan los datos del encabezado en el registro temporal 2. Pasa al estado 3.

4. Estado 3: Si la bandera de "Aceptar" para la IP de destino es '1' se hace una operación "XOR" entre los campos de la IP Destino de la regla y los datos, y se almacena en el registro resultado; en caso contrario se cargan '1'. Si la bandera de "Aceptar" para la IP de origen es '1' se hace una operación "XOR" entre los campos de la IP Origen de la regla y los datos, y se almacena en el registro resultado; en caso contrario se cargan '1'. Si la bandera de .ªceptar"para el protocolo es '1' se hace una operación "XOR" entre los campos Protocolo de la regla y los datos, y se almacena en el registro resultado; en caso contrario se cargan '1'. Si la bandera de "Aceptar" para el puerto es '1' se hace una operación "XOR" entre los campos Puerto de la regla y los datos, y se almacena en el registro resultado; en caso contrario se cargan '1'. Pasa al estado 4.

5. Estado 4: Se realiza una operación "AND" entre todos los bits del registro resultado y se almacena el resultado en cmp. Se hace una comparación entre

la dirección de memoria actual y la dirección máxima recibida, se almacena el resultado en edo. Pasa al estado 5.

6. Estado 5: Se incrementa la dirección de memoria en 1. Si, (!tmp1[92] AND cmp) OR (tmp1[92] AND !cmp) el registro ctrl toma el valor de "11" y pasa al estado 0. De lo contrario, si edo=0, el registro ctrl toma el valor de "10" y pasa a estado 0, si edo=1, pasa a estado 1.

5.4.9. Funciones adicionales para el funcionamiento del *firewall*

A continuación se describen funciones adicionales que se desarrollaron para el funcionamiento el *firewall*:

1. bit_vector2integer:
   - Convierte un vector de bits a su correspondiente entero.
   - Recibe: un vector del tipo std_logic_vector de 4 elementos.
   - Produce: un entero de entre 0 y 15.
2. integer2bit_vector:
   - Convierte un número entero a un vector de bits.
   - Recibe: un número entero de entre 0 y 15.
   - Produce: un vector del tipo std_logic_vector de 4 elementos.
3. operacionand:
   - Hace la operación lógica AND entre todos los elementos de un vector.
   - Recibe: un vector del tipo std_logic_vector de 88 elementos.
   - Produce: el resultado de la operación AND entre todos los elementos de ese vector.
4. comprueba:
   - Dados los vectores D y M comprueba si son iguales.
   - Recibe: dos vectores, D y M, del tipo std_logic_vector de 4 elementos.
   - Produce: un 0 si son iguales, u 1 si no lo son.

5.5. Interfaces de comunicación

Para la integración del sistema de *firewall* con la tarjeta de desarrollo, que contiene microprocesadores y bloques de memoria interconectados por un sistema de buses, en donde para escribir a un dispositivo es necesario mandar los datos a un espacio de direcciones de memoria específico. Sera necesario agregar dispositivos extra y agregar funciones especiales que realicen operaciones de lectura y escritura al dispositivo. Tanto los dispositivos extra como las funciones de lectura y escritura al dispositivo componen la interfaz de comunicación entre el *hardware* y el *software*.

La interfaz *hardware-software*, se hizo agregando registros de propósito especifico para la entrada y salida de datos conectados al sistema de buses. El conjunto de los registros de entrada-salida se dividen en: registros de envío y recepción de señales de control y registros de envío y recepción de datos. Los registros que reciben señales del sistema de buses al sistema *firewall* tienen el prefijo Bus2IP, lo que quiere decir, que la señal va del bus del sistema al IP (Intellectual Property). Mientras que los registros que envían señales desde el *firewall* al sistema de buses tienen el prefijo IP2Bus, que significa que los datos van de nuestro IP al sistema de buses.
Los registros de control que se utilizaron para la comunicación del bus con el *firewall* fueron: Bus2IP_Clk para proporcionarle la señal de reloj, Bus2IP_Reset, Bus2IP_Data se utilizo para recibir datos del bus, Bus2IP_WrCE y Bus2IP_RdCE utilizados para direccionar los registros slv_reg al los que se quiere escribir o leer, respectivamente. Y para la comunicación del *firewall* con el bus se utilizo el registros IP2Bus_Data para enviar datos por el bus.

La interfaz de comunicación *software-hardware*, para realizar esta interfaz fueron utilizadas las funciones generadas por el entorno de desarrollo (Xilinx EDK), que son un conjunto de macros para escribir en el espacio de direcciones definido para el modulo de *firewall*. Las funciones utilizadas tienen el siguiente prototipo:
FIREWALL_mWriteSlaveReg0(XPAR_FIREWALL_0_BASEADDR, value);
en Donde el primer argumento de la función es la dirección base de memoria del modulo de *firewall* y el segundo argumento es el valor que se desea escribir al registro slv_reg0.

5.6. Diseño de *Software*

Como se menciono anteriormente, en un principio la comunicación seria del puerto etherntet de la tarjeta de desarrollo a Internet y del puerto USB de la computadora al puerto USB de la tarjeta de desarrollo. Esta ultima no pudo llevarse a cabo debido a que no hay documentación suficiente a cerca del manejo del puerto USB en la tarjeta de desarrollo. Ademas de que en la documentación se menciona que el puerto USB solo es usado para configuración y para depuración de *hardware*. Debido a lo anterior se

decidio cambiar la comunicación USB por comunicación a través del puerto RS-232 de la tarjeta al USB de la computadora utilizando un cable NULL-Modem, este rumbo de trabajo tuvo muchas desventajas ya que la velocidad disminuiría notablemente y ademas funcionalidad del puerto de la tarjeta es mínima y al tratar de hacer la comunicación por medio del puerto serial nos encontramos con que necesitábamos agregar una dirección ip al puerto serial, este detalle nos hizo replantear el proyecto y limitarlo en la comunicación, a solo usar la interfaz ethernet.

Una ves replantada la comunicación se desarrollo una aplicación que trabaja conjuntamente con la parte *hardware*, esta aplicación seria la encargada de enviarle los datos necesarios al modulo en *hardware* y que se encargara de la comunicación entre la interfaz gráfica de usuario y el *hardware* desarrollado.

A continuación se explica la estructura del programa:

El programa utiliza 3 hilos los cuales se encargan de obtener los paquetes de red, procesar los paquetes y apoyar la programación de las reglas y la petición de la bitácora. Su diagrama de flujo se presenta en la figura 5.20.

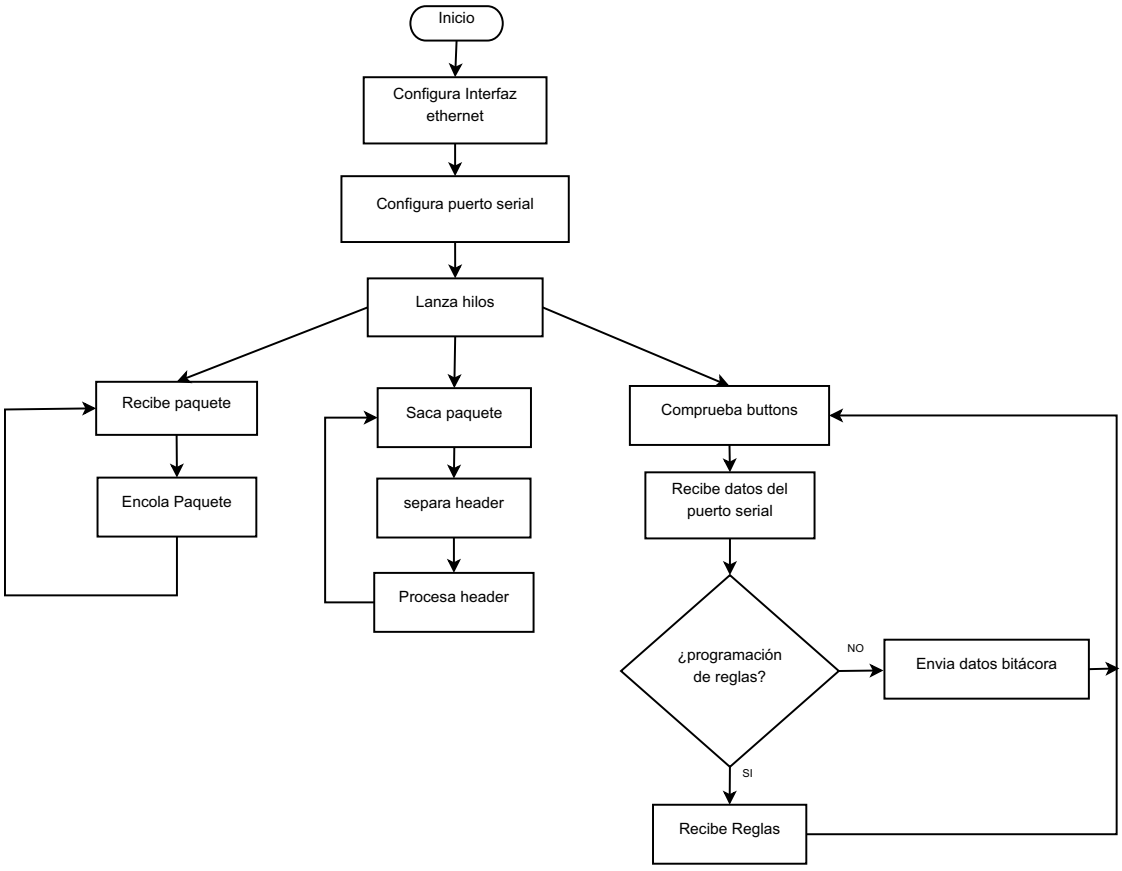


Figura 5.20: Diagrama de flujo de la aplicación para la utilización del *firewall*.

1. El programa lo primero que hace es configurar la interfaz ethernet, para que pueda capturar cualquier paquete que entre por ella, es decir , poner la interfaz en modo promiscuo.

2. Después se inicializa el UART (*Universal Asynchronous Receiver-Transmitter*), para poder enviar y recibir datos por el puerto serial.

3. se lanzan los hilos.

4. Hilo de recepción de paquetes, se encarga de leer la entrada de datos por la interfaz ethernet, en cuanto obtiene un dato, lo encola para que después sea procesado.

5. Hilo saca paquete, este hilo se encarga de obtener las partes del paquete que procesara el *firewall*, ademas de imprimir en la salida estándar (en nuestro caso el puerto serial) los paquetes, así como algunos campos del encabezado IP, con el fin de poder visualizarlos en la terminal de la computadora. este hilo también se encarga de mandar las partes del encabezado ip al *firewall*, después de lo cual espera el resultado y dependiendo del resultado podría pasarlo a otra interfaz de red si se tuviera.

6. El hilo de comprobación de botones, se utiliza para detener el hilo que se encarga de sacar los paquetes, esto lo hace comprobando el estado de los botones de la tarjeta de desarrollo, una vez que cambia el estado de los botones para el hilo que saca los paquetes de la cola y se pone en espera de datos provenientes del puerto serial. Una ves que llega un paquete ve si es un paquete de regla o uno de petición de bitácora, si es un paquete de regla, toma la regla y la almacena dentro del *firewall* y si es un paquete de bitácora manda la interrupción al *firewall* para que este le envíe el contenido de la bitácora.

5.7. Sistema Funcional

El sistema fue implantado en la tarjeta de desarrollo XUP2VP, la cual interconecta el FPGA con otros dispositivos que pueden ser utilizados como periféricos en el desarrollo de sistemas. En la figura 5.21 se muestra la estructura del sistema final.

EL cuadro marcado como RJ-45 representa el módulo de la interfaz de red contenido en la tarjeta de desarrollo, El cuadro marcado como DIMM DDR representa el módulo de memoria DDR SDRAM incrustado en el slot de la tarjeta, el cuadromarcado como RS232 representa el dispositivo para la transmisión serial, el cuadro marcado como LED representa los LEDS accesibles por el usuario que contiene la tarjeta y por ultimo el cuadro marcado como PUSH BUTTON representa los botones contenidos en la tarjeta de desarrollo XUP2VP.

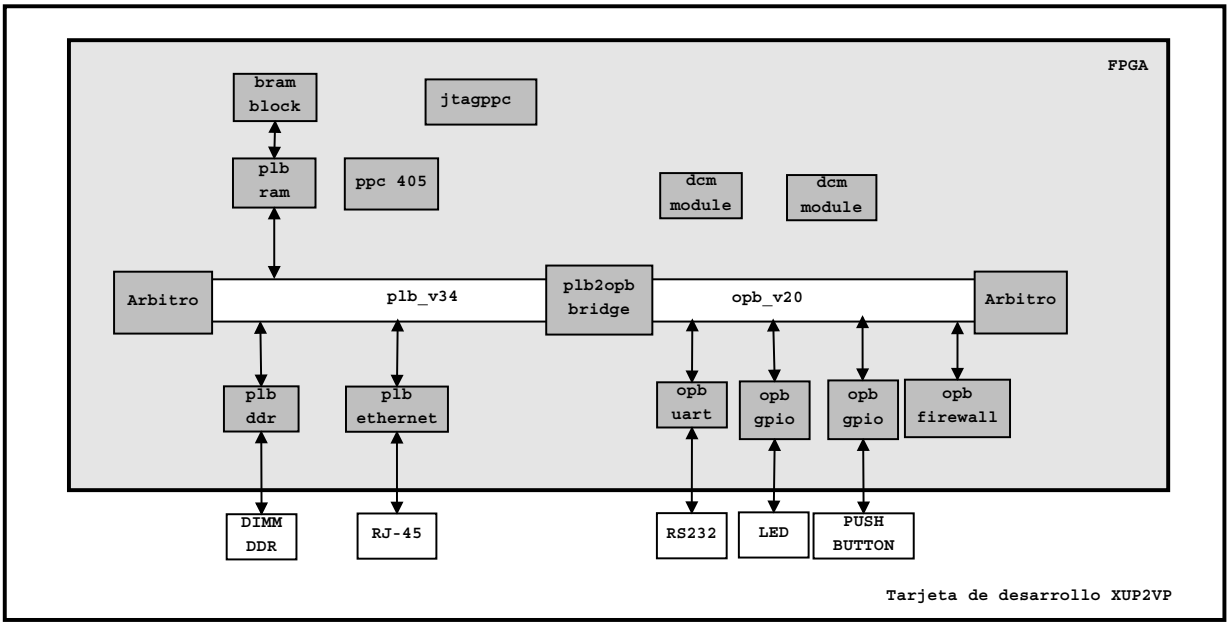


Figura 5.21: Estructura del sistema final.

5.7.1. Mapa de direcciones

La tabla 5.7.1 muestra el mapa de dispositivos en memoria ocupados para el sistema final.

Dispositivo	Bus	Rango de direcciones
plb_bram	plb	0xFFFE0000-0xFFFFFFFF
Ethernet_MAC	plb	0x84000000-0x8040FFFF
Firewall_0	opb	0x71000000-0x7100FFFF
RS232_Uart_1	opb	0x40600000-0x4060FFFF
Leds	opb	0x40020000-0x4002FFFF
Push_Buttons	opb	0x40000000-0x4000FFFF
DDR_128MB	plb	0x00000000-0x07FFFFFF

Tabla 5.1: Mapa de dispositivos en memoria

5.7.2. Descripción de los componentes del sistema

Descripción del *hardware* del sistema en el chip

- **ppc405**, procesadores PowerPC 405 a 100 Mhz.
- **jtagppc cntlr**, controlador del puerto JTAG (Joint Test Action Group) para la programación del FPGA, el microprocesador incrustado y de memorias flash.
- **plb ram**, controlador para bloques de memoria RAM en chip.

- **bram block**, bloque de memoria RAM en chip de 128KB. En este bloque de memoria se debe alojar el programa principal: texto, pila y heap.

- **plb ddr**, controlador para bloques de memoria DDRSDRAM fuera del chip.

- **plb v34**, es el bus local del procesador. Es un de alta velocidad, utilizado para conectar bloques de memoria interna y externa al FPGA, así como periféricos de alta velocidad.

- **opb v20**, bus de periféricos de baja velocidad en el chip.

- **plb2opb bridge**, puente para la interconexión del bus local del procesador y el bus de periféricos.

- **opb uartlite**, controlador para un transmisor serial agregado al bus de periféricos en chip. La configuración utilizada es: 9600 baudios, 8 bits de datos, sin paridad y sin interrupciones.

- **dcm module** (Digital Clock Manager), módulos de reloj digital, los cuales optimizan el manejo de relojes digitales en el FPGA.

- **proc sys reset**, sistema de reset general para el microprocesador y el sistema.

- **plb ethernet**, controlador para comunicacion via ethernet conectado al bus local del procesador, Soporta operación *full-Duplex* a 10 Mb/s y 100 Mb/s.

- **opb gpio**, contralador para puertos de entrada y salida de propósito general-conectado al bus de periféricos.

Descripción del *software*

- **Xilkernel**[5], es un Kernel pequeño, robusto y modular, que permite un alto grado de personalización. Ya que da la posibilidad de utilizar varios hilos de ejecución, permitiendo configurar la secuencia de aplicación.

- **cpu ppc405**, indica la arquitectura objetivo al compilador. Debido a que el FPGA es capaz de contener dos tipos de procesadores, MicroBlaze o PowerPC405.

- **plbarb, uartlite, emac, gpio, bram, ddr**. Representan el código fuente de los drivers de los diferentes dispositivos del sistema, los cuales deberán ser compilados y enlazados en la aplicació por medio de bibliotecas estáticas.

- **firewall_bitacora**, contiene las funciones necesarias para la comunicación con el modulo de hardware firewall_bitacora.

5.7.3. Recursos Ocupados por el Sistema

La tabla 5.7.3 muestra los recursos ocupados por el sistema diseñado.

El numero de Slices, en su mayoría fueron ocupados por el componente de bitácora ya que ocupo una memoria de 32 localidades cada una con tamaño de 80 bits. El numero de IOBs, fueron ocupados para la entrada de datos al sistema de firewall, esta cantidad podría ser menor si se implementara una maquina de estados para obtener los datos.

Recursos	Usados	Disponibles	Porcentaje
Number of slices	3419	13696	24 %
Number of Filp-Flops	3591	27392	13 %
Number of 4 input LUTs	5877	27392	21 %
Number of bonded IOBs	364	556	65 %
Number of BRAMs	6	136	4 %
Number of GCLKs	8	8	50 %

Tabla 5.2: Recursos Ocupados por el sistema

5.8. Interfaz gráfica de usuario (GUI)

La interfaz Gráfica de Usuario (GUI por sus siglas en inglés) es una aplicación que permitirá al usuario construir las reglas que serán utilizadas para su uso por el *firewall*.

El propósito de esta aplicación es permitirle al usuario una fácil creación de las reglas que serán utilizadas por el *firewall*, simplemente haciendo 'clic' en los elementos que componen la GUI, y llenando algunos campos de texto, la interfaz será capaz de convertir esas entradas recibidas por parte del usuario, al formato utilizado por el *firewall* para realizar las comparaciones correspondientes. Para este proyecto, la GUI está escrita utilizando la biblioteca Qt3, fue diseñada con el IDE Qt3 Designer, y compilada para *Linux* utilizando el compilador de GNU.

La aplicación muestra dos pestañas que pueden ser seleccionadas por el usuario, la vista normal, que es en dónde se programarán las reglas del *firewall* por parte del usuario. La segunda pestaña es en dónde aparecerán los resultados del proceso de filtrado del *firewall*, indicando cuáles son las direcciones IP de origen y destino de los paquetes bloqueados, así como su número

Vista Normal

Es la ventana que se le muestra al usuario al ejecutar la aplicación, contiene opciones básicas para que el usuario pueda construir de forma sencilla las reglas para el *firewall*, figura 5.22.

Figura 5.22: Vista normal de la interfaz de usuario.

La ventana principal contiene tres contenedores para agrupar los elementos principales de selección, grupoProtocolos, el cual contiene los botones para la selección de protocolo, TCP, UDP, ICMP.

Los botones son mutuamente excluyentes.
Al seleccionarse un protocolo, se realiza una conversión utilizando su número de puerto a un valor hexadeximal de un byte.
Si se encuentra que uno de los botones está seleccionado, se indicará que será utilizada la bandera de habilitación correspondiente en las reglas.
El grupoServicios que contiene una selección de los puertos principales en los que se puede trabajar.
Al iagual que en el grupo de protocolos, en este grupo la selección de los botones son mutuamente excluyentes.
Dependiendo de la opción seleccionada, se convertirá el número de puerto correspondiente a un valor hexadecimal de 2 bytes.
Si se encuentra alguna opción seleccionada, se indicará que se utilizará la bandera de habilitación correspondiente al puerto.
EL grupoIPs contiene los campos en dónde se podrán introducir las direcciones IP que serán utilizados por las reglas.
Las entradas que reciben los campos son cuatro números enteros con valores de entre 0 y 255 cada uno; cada uno de los números esta separado por un punto (.).
Si los campos no están vacíos, se indicará que se utilizará la banderá de habilitación correspondiente.
El botón para Aceptar paquete (aceptarPaquete) indica si la regla será o no aceptada, si está habilitado, la regla será para aceptar un paquete; si no esta habilitado, la regla será para rechazar un paquete.

El botón que marca la Regla de entrada(entradasalida) indicará, de estar seleccionado, si la regla se cargará como una regla de entrada, es decir para el flujo de datos que va desde la red hacia la computadora
Si no está selecciona se tomará como regla de salida, es decir, que aplica cunado el flujo de datos va desde la computadora hacia la red.
El botón Nueva, sirve para procesar la selección hecha y crear una nueva regla. Sólo se permite la realización de 10 regas como máximo.
El botón finalizar prepara las reglas para que sean cargadas al firewall.
El botón cancelar, reinicializa los valores que hay sido almacenados a sus valores originales.

Funcionamiento

El usuario puede o no seleccionar alguna opción de cada uno de los grupos que hay en la interfaz gráfica, ya sea haciendo 'clic' en los diferentes botones, o bien, introduciendo las direcciones IP en las cajas de texto correspondientes.
Una vez que se ha pulsado el botón de nueva regla, se verificará cada una de las opciones presentadas para poder determinar cuáles están seleccionadas, o si las cajas de texto para las direcciones contienen texto en ellas, en cuyo caso se consideran seleccionadas.
Dependiendo de en cual grupo se encuentren las opciones seleccionadas, se llamarán las funciones correspondientes para convertir sus valores al valor hexadecimal correspondiente. Además, si hay alguna selección, se considerará que el campo correspondiente esta en uso para que se utilicen las banderas de habilitación correspondiente.
Al presionar el botón Finalizar, las reglas creadas son enviadas para que el *firewall* pueda ser programado.

Bitácora

En esta pestaña se muestra el contenido de la bitácora generada por el *firewall*, mostrando información sobre los paquetes rechazados, sus ips de origen y de destino, y el número. La información de la bitácora se presenta en una caja de texto que aparece en la pestaña. Para el control se emplean los botones Recibir, y limpiar, que reciben y limpian el contenido de la caja de texto respectivamente, figura 5.23.

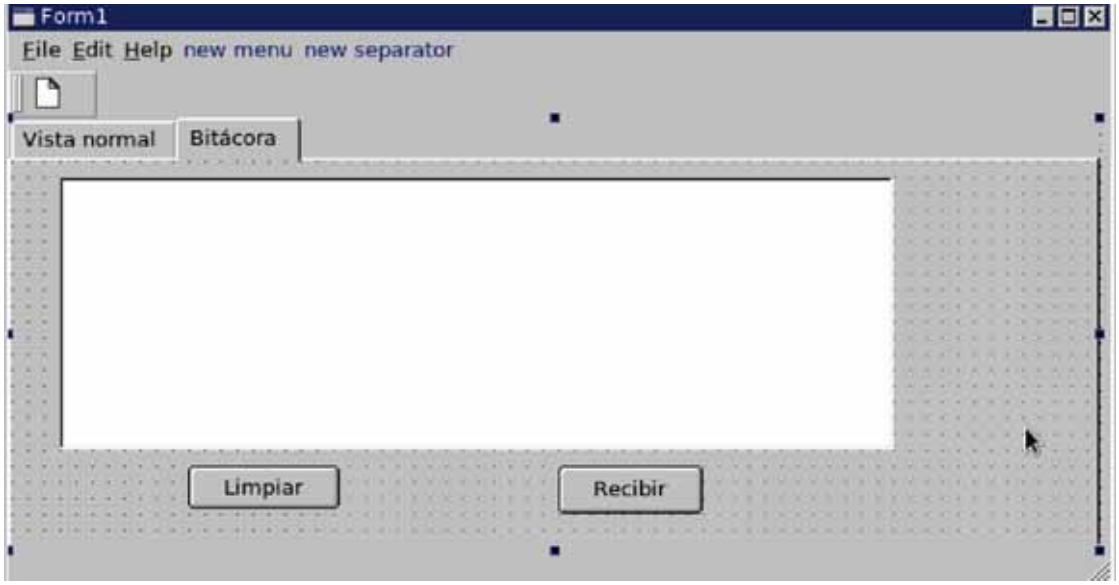


Figura 5.23: Vista Bitácora de la interfaz de usuario.

Funcionamiento

Al hacer 'clic' en el botón recibir, las Interfaz, recibirá información que produce el *firewall* a través del puerto serial, y dicha información será escrita en la caja de texto.
El botón limpiar se encargará de borrar la información contenida en la caja de texto

Funciones de conversión de datos

Para poder realizar la conversión de los datos en formato decimal presentados al usuario, a una forma hexadecimal equivalente que pueda ser utilizado por el dispositivo como reglas para la comparación de correspondiente con los datos extraídos de los encabezados, es necesaria una conversión de los mismos. Las computadoras y dispositivos como el *firewall* diseñado trabajan con bits, diferentes representaciones de esos bits nos dan diferentes tipos de datos con diferentes valores. Por ejemplo, las direcciones IP, de origen o de destino se presentan como un conjunto de 32 bits. Estos pueden verse como un conjunto de 4 campos de 8 bits cada uno, los cuales pueden tomar valores que van desde el 0 hasta el 255 cada uno. Por esta razón es necesario convertir los datos desde su representación utilizada por la interfaz gráfica de usurio, a la forma en la que el FPGA.

Descripción de las funciones de conversión

Se describirán las funciones utilizadas para la conversión de datos, así como sus entradas y sus salidas.

- **Struct unPuerto**: esta estructura contiene los 2 bytes necesarios para representar uno de los puertos en hexadecimal.

  La estructura tiene como miembros las siguientes variables:
  h: de tipo char, almacena los 8 bits más significativos del puerto
  l: de tipo char, almacena los 8 bits menos significativos del puerto:

- **ConvierteIP**: hace la conversión entre una dirección IP a un valor hexadecimal.

  Prototipo: char *ConvierteIp(char*)
  Regresa: un apuntador a un char que contendrá el valor hexadecimal de la ip.
  Recibe: La dirección IP, en forma de cuatro números enteros enteros con valor de entre 0 y 255 cada uno, separados por puntos.

- **Habilitados**: Convierte una cadena de caracteres a un sólo valor hexadecimal que indicará al *firewall* cuáles son las banderas que serán utilizadas por las reglas.

  Prototipo: char HabilitadostoHex(char *)
  Regresa: Una variable tipo char que contiene la información sobre las banderas de habilitación que han sido seleccionadas.
  Recibe: Un arreglo de char que indican cuales son los campos de las reglas seleccionados desde la interfaz de usuario.

- **CampotoHex**: Convierte un entero a su representación en caracter.

  Prototipo: char CampotoHex(int);
  Regresa: Un caracter.
  Recibe: Un entero con valor entre 0 y 255.

- **Puerto2Hex**: Convierte un número entero al un valor con el puerto correspondiente en hexadecimal.

  Prototipo: struct unPuerto puerto2Hex(int)
  Regresa: una estructura del tipo unPuerto conteniendo la información en hexadecimal del puerto a utilizar.
  Recibe: un entero de con valor entre 0 y 65535.

Funciones para el manejo del puerto serial

Estas funciones se encargarán de la trasmisión y de la recepción de los datos, hacia y desde el puerto serial, pues es a través de este puerto que se realiza la comunicación entre la computadora y el dispositivo. Estas funciones son utilizadas para el envío de las reglas para la programación del dispositivo, y la recepción de la información generada por el *firewall* para las bitácoras.

Descripción de las funciones para el manejo del puerto serial

Se describen las funciones utilizadas para el manejo del puerto serial:

- **serial_open**: abre un descriptor de un archivo para manejo del puerto serial.

  Prototipo: int serial_open(char *serial_name, speed_t baud)
  Regresa: Un entero con el valor del descriptor.
  Recibe: Una cadena con el nombre que se le va a dar al descripotor, y la velocidad de transmisión.

- **serial_send**: Escribe un paquete de datos al descriptor que maneja el puerto serial.

  Prototipo: int serial_send(int serial_fd, char *data, int size)
  Regresa: Número de bytes enviados.
  Recibe: Descriptor para trabajar con puerto serial,apuntador a la cadena con los datos a enviar, número de bytes a enviar.

- **serial_recive**: Lee datos desde el puerto serial.

  Prototipo: int serial_recive(int serial_fd, char *data, int size)
  Regresa: Número de bytes recibidos.
  Recibe: Descriptor para trabajar con el puerto serial, apuntador a la cadena que recibirá los datos, número de bytes a recibir.

Capítulo 6

Pruebas y Resultados

6.1. Pruebas

Las pruebas del sistema se hicieron teniendo la tarjeta conectada directamente a Internet y conectada a una computadora a través del puerto serial.

Para comenzar a hacer las pruebas fue necesario comprobar que la tarjeta fue reconocida por la computadora, para poder bajar el sistema, una vez reconocida la tarjeta y asegurando nos de que el *firmware* fue cargado correctamente, configuramos el puerto serial para que tenga la misma velocidad y las mismas características de comunicación en ambos lados, tanto en la parte correspondiente a la tarjeta como la parte correspondiente al puerto USB de la computadora, esto se debe a que se utiliza un cable que tiene conexión USB-RS232. La velocidad a la que se programo el puerto serial fue a 9600 bits/s con las características de comunicación: sin paridad, con un bit de paro y con 8 bits de recepción.

Una ves configurado lo anterior bajamos el programa a la tarjeta de desarrollo, para poder realizar las pruebas correspondientes. El proceso de configuración se presenta de manera detallada en la parte de Apéndice.

Una ves bajado el sistema al FPGA, este comenzó a filtrar, permitiendo el paso de todos los paquetes, esto se debe a que el *firewall* aun no cuenta con reglas para hacer el filtrado y tiene por defecto permitir todo el trafico de red. Esto se muestra en la figura 6.1.

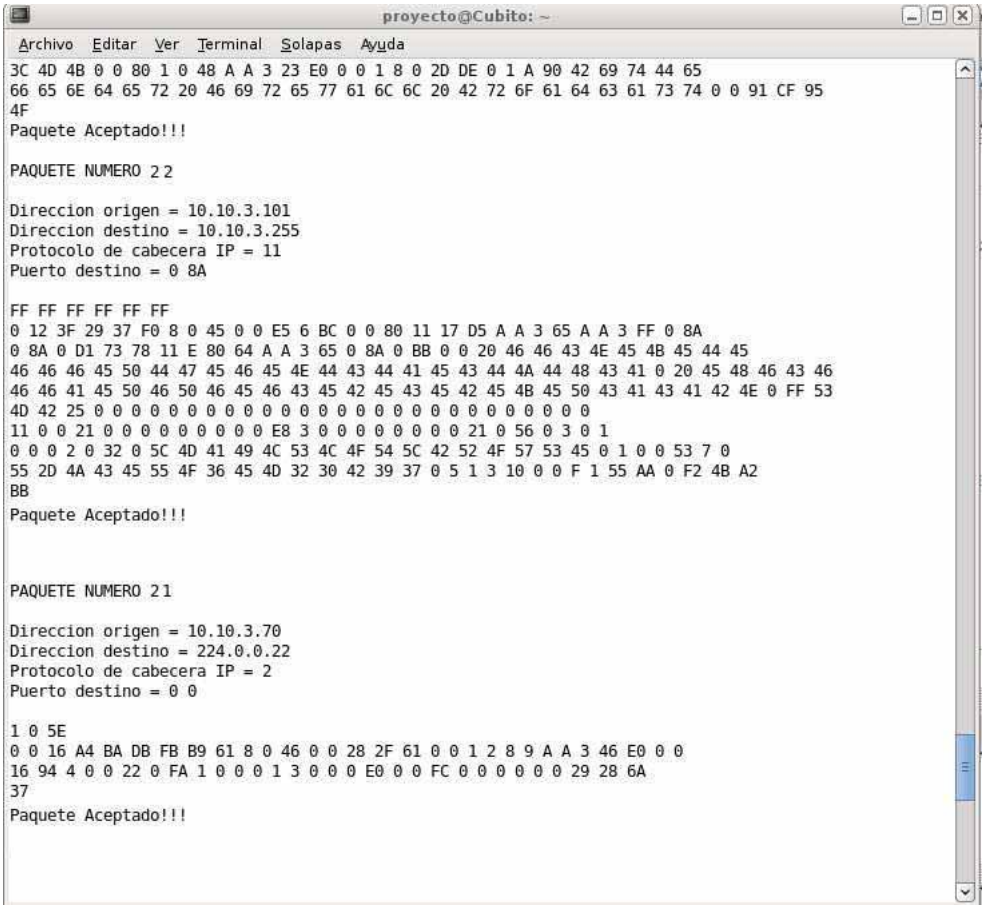


Figura 6.1: Prueba realizada sin cargar ninguna regla.

Ahora se presenta la prueba cargando le las reglas desde la interfaz de usuario. Para realizar esta prueba se inicio la interfaz de usuario, para cargar las reglas al *firewall*. Para iniciar la interfaz de usuario entramos a la carpeta que contiene el proyecto y ejecutamos $qmake y despues $make para compilar la aplicación y por ultimo $./GUI para arrancar la interfaz de usuario. Una vez puesta en marcha la interfaz de usuario, figura 6.2, se cargo nuevamente el sistema en el FPGA, pulsamos algún *PUSH BUTTON* del FPGA, la aplicación en el FPGA queda esperando las reglas.

Figura 6.2: Vista para la programación de reglas.

En la interfaz de usuario se crearon dos reglas una regla que denegara el acceso a un paquete con dirección destino 10.10.3.255 y una regla que permitiera el paso a paquetes que tuviera como dirección origen 10.10.3.70. Ya creadas las reglas pasamos las reglas al *firewall* haciendo clic en el botón Finalizar. Los resultados obtenidos son los de la figura 6.3.

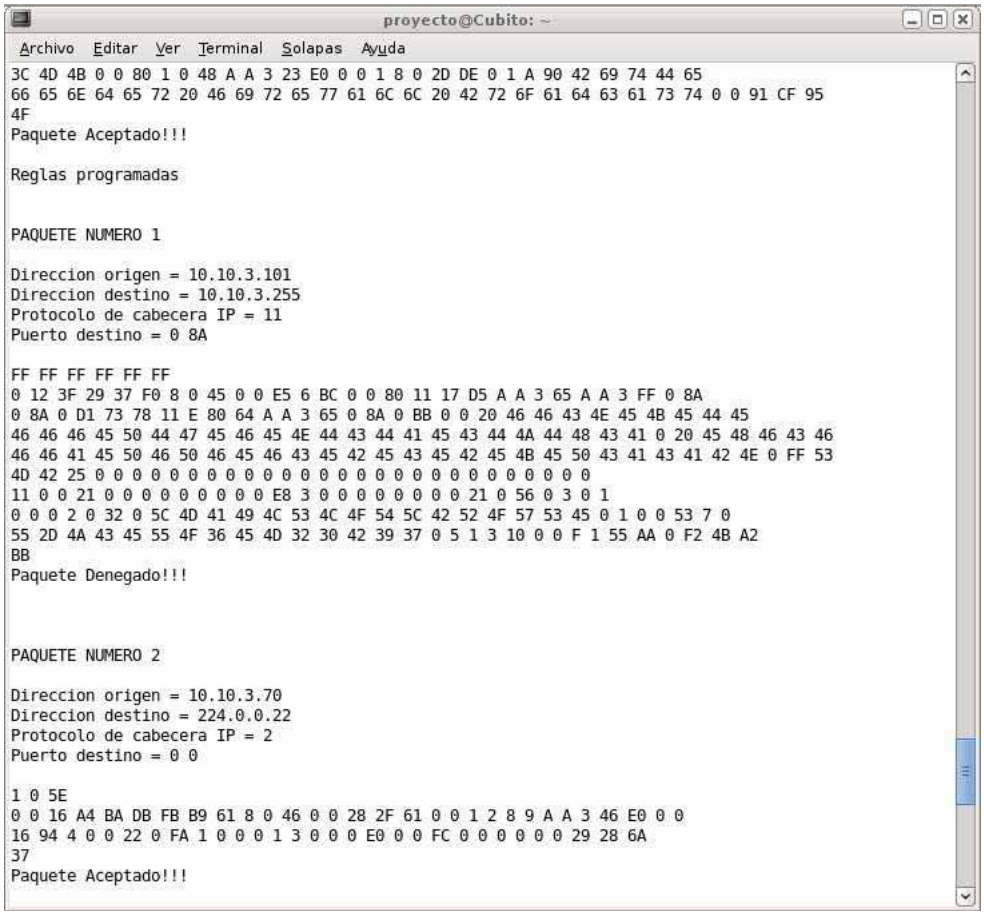


Figura 6.3: Prueba realizada cargando dos reglas.

Ya que los resultados del filtrado fueron satisfactorios, se asume que la interfaz de usuario hizo una traducción correcta y que la comunicación no tiene fallas.

Ahora se presenta la prueba de petición de bitácora, para esta prueba se ocuparon los resultados de la prueba anterior ya que se filtraron algunos paquetes. Para hacer la petición de la bitácora se presiona nuevamente uno de los *PUSH BUTTONS*, esto se hace para que la aplicación que corre en el FPGA no escriba a la terminal, con la finalidad de que no afecte la comunicación con la interfaz gráfica. Una ves pulsado el botón el sistema en el FPGA espera un dato del puerto serial, como enviamos una petición de bitácora, entonces comenzara a enviar el contenido de la bitácora para presentarlo en la interfaz de usuario. Cada que se envía un paquete de petición de bitácora, el *firewall* regresa un dato de la bitácora, el envío de peticiones de bitácora termina cuando la interfaz recibe como resultado de la petición un arreglo de ceros. El resultado de la petición se presenta en la figura 6.4

En este capitulo se presentaron las pruebas realizadas al sistema *firewall* conectado con la interfaz de usuario. Y dado resultados que se obtuvieron fueron satisfactorios

Figura 6.4: Resultado de la petición de bitácora.

en todos los casos. Se concluye que se cumplieron los objetivos establecidos para el *firewall* y para la interfaz gráfica.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

7.1.1. Genaro Aldama Mejía

Primero, considero que depués de la realización de este proyecto, fui capaz de comprender la complejidad que implica el funcionamiento y el desarrollo de una aplicación como esta. A pesar de que se utilizó una forma muy simplificada de las reglas, y que el firewall sólo emplea la funcionalidad mínima, aún así el diseño y la implementación resultaron ser tareas complejas.
Además de que se tuvieron que rediseñar varios componentes del firewall, puesto que sólo se habían llevado a cabo simulaciones, tras intentar implementar los componentes fue necesario realizar cambios en el diseño. Definitivamente la falta de experiencia tanto con las herramientas de desarrollo como con algunos conceptos utilizados para el proyecto limitaron la forma con la que se trabajó al principio, sin embargo, a medida que se fueron investigando las soluciones a los problemas encontrados, se fue adquiriendo una experiencia importante que hizo posible la continuación del proyecto. Después de haber visto el resultado puedo realizar las siguientes afirmaciones.

1. La implementación de un dispositivo dedicado exclusivamente a realizar las funciones de un firewall perimite que se omitan funciones innecesarias, lo que trae como resultado el ahorro de recursos, o la mejor utilización de los recursos ya disponibles.

2. El desplazar estas tareas a un dispositivo externo, reduce la carga de las tareas que deberá de realizar la computadora.

3. Además, a medida que se incrementan las velocidades de transmisión de datos disponibles para el usuario final, hace que sea necesario que los firewall sean capaces de operar a velocidades cada vez mayor. Con la complejidad del software ejecutado por las computadoras personales incrementándose contínuamente,

junto con la cantidad de recursos necesarios para su ejecución, hacen parecer que una solución de seguridad en hardware sea razonable.

4. Los componentes críticos que requieran del mejor desempeño posible, podrán ser escritos para el hardware, mientras los recursos así lo permitan.

5. Se reduce la cantidad de componentes de software a aquellos que serán utilizados como intermediario y control, entre la computadora y el dispositivo.

6. Un firewall elaborado con una combinación entre hardware y software perimite la flexibilidad necesaria, mediante el diseño y programación del reglas al dispositivo que cumplan con las necesidades del usuario, así como una considerable rapidez en su ejecución. Dicho en otras palabras, combina las mejores características de cada uno de los enfoques.

7. El uso de un FPGA como medio en el cual se ha desarrollado el proyecto se justifica debido a la flexibilidad que este ofrece, permitiendo el empleo de multiples recursos y la posibilidad de ser programado varias veces, a medida que se encuentran los mejores resultados en el diseño o cambios en el mismo.

8. El uso de un lenguaje de descipción de hardware, VHDL en nuestro caso, se justifica por el alto nivel de abstracción que permite, así como el hecho de que es más fácil de utilizar cuando se cuenta con experiencia previa en otros lenguajes de programación.

9. Si este proyecto se continuará, se podrían desarrollar dispositivos con una mayor funcionalida y que estuviesen disponibles de forma comercial para el usuario final.

10. Las limitaciones presentadas en el proyecto, son producto de las limitaciones en el equipo utilizado para el desarrollo o debidas a la falta de experiencia y tiempo para su elaboración, y no debidas a la naturaleza misma del proyecto; con mayor tiempo disponible hubiese sido posible presentar un resultado más refinado. Por último, considero que los objetivos planteados para la realización de este Proyecto Terminal han sido cumplidos, al haber podido implementar en un FPGA un firewall básico que utilice tanto componenetes de software para su programación y control, como componentes de hardware para su ejecución.

7.1.2. José Eduardo Ochoa Jiménez

Con la realización del proyecto pude comprender lo complejo que es desarrollar una aplicación como lo fue el Firewall que aunque la funcionalidad fue muy básica, la implementación fue un trabajo complicado ya que no se contaba con la experiencia necesaria y por lo tanto se avanzava de a poco, esto me dejo grandes lecciones que me servirán en mi vida profesional, como lo es la importancia de una buena planeacion y estructura de una aplicación. Aunque sin duda el tener que replantear el proyecto

varias veces ayudo a experimentar los errores que se pueden cometer al realizar el diseño, para poder evitarlos en trabajos posteriores.
Una de las cosas que ayudaron a la realización de este proyecto fue el que los FPGAs proporcionan una gran flexibilidad para desarrollar sistemas ya que pueden ser reprogramados, miles de veces con el objetivo de ir mejorando el diseño inicial, hasta llegar al resultado esperado. Esta característica de los FPGAs es de gran ayuda para propositos academicos ya que gracias a eso se pueden experimentar diversos comportamientos de una aplicación con el fin de encontrar una solución optima.

La elaboración del proyecto fue un reto debido a que se debieron resolver los problemas que se presentaron al momento de tener que utilizar la tarjeta de desarrollo, problemas que fueron causados en su mayoría por la falta de experiencia en el manejo de las herramientas necesarias para la utilización de la tarjeta y en parte a la mala documentación encontrada. Esta experiencia con los dispositivos reprogramables me hizo cambiar la manera de ver las cosas ya que el diseño de hardware tiene muchos beneficios que valen la pena explorar y aplicar no solo en propósitos de investigación si no en la vida diaria, como ejemplo esta el sistema de firewall, el tener un dispositivo independiente de los recursos de una computadora para el filtrado de paquetes es algo que debiera existir para poder llevar tu firewall a donde quiera que vayas.

Con la flexibilidad de la metodología de codiseño hardware-software en conjunto con la flexibilidad antes mencionada de los FPGAs, es posible agregar al proyecto mas funcionalidades para con esto, poder tener una herramienta en hardware que proporcione un alto nivel de protección, con la finalidad de quitar aplicaciones que ocupan recursos de computo que podrian ser ocupados en otro tipo de procesamiento. Ademas de que se aprovecharia el nivel de paralelismo que ofrece el hardware.

Apéndice A

Archivos de configuración

Archivo: `system.mhs` (*Especificaciones de hardware del microprocesador*)

```
1 # ##############################################################################
2 # Created by Base System Builder Wizard for Xilinx EDK 8.2.02 Build EDK_Im_Sp2.4
3 # Mon Dec  6 11:16:20 2010
4 # Target Board:  Xilinx XUP Virtex-II Pro Development System Rev C
5 # Family:       virtex2p
6 # Device:       xc2vp30
7 # Package:      ff896
8 # Speed Grade:  -7
9 # Processor: PPC 405
10 # Processor clock frequency: 300.000000 MHz
11 # Bus clock frequency: 100.000000 MHz
12 # Debug interface: FPGA JTAG
13 # On Chip Memory : 128 KB
14 # Total Off Chip Memory : 128 MB
15 # - DDR_SDRAM_16Mx64 Single Rank = 128 MB
16 # ##############################################################################
17
18
19 PARAMETER VERSION = 2.1.0
20
21
22 PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = I
23 PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = O
24 PORT fpga_0_Ethernet_MAC_slew1_pin = net_vcc, DIR = O
25 PORT fpga_0_Ethernet_MAC_slew2_pin = net_vcc, DIR = O
26 PORT fpga_0_Ethernet_MAC_PHY_rst_n_pin = fpga_0_Ethernet_MAC_PHY_rst_n, DIR = O
27 PORT fpga_0_Ethernet_MAC_PHY_crs_pin = fpga_0_Ethernet_MAC_PHY_crs, DIR = I
28 PORT fpga_0_Ethernet_MAC_PHY_col_pin = fpga_0_Ethernet_MAC_PHY_col, DIR = I
29 PORT fpga_0_Ethernet_MAC_PHY_tx_data_pin = fpga_0_Ethernet_MAC_PHY_tx_data,
30 DIR = O, VEC = [3:0]
31 PORT fpga_0_Ethernet_MAC_PHY_tx_en_pin = fpga_0_Ethernet_MAC_PHY_tx_en, DIR = O
32 PORT fpga_0_Ethernet_MAC_PHY_tx_clk_pin = fpga_0_Ethernet_MAC_PHY_tx_clk, DIR = I
33 PORT fpga_0_Ethernet_MAC_PHY_tx_er_pin = fpga_0_Ethernet_MAC_PHY_tx_er, DIR = O
34 PORT fpga_0_Ethernet_MAC_PHY_rx_er_pin = fpga_0_Ethernet_MAC_PHY_rx_er, DIR = I
35 PORT fpga_0_Ethernet_MAC_PHY_rx_clk_pin = fpga_0_Ethernet_MAC_PHY_rx_clk, DIR = I
36 PORT fpga_0_Ethernet_MAC_PHY_dv_pin = fpga_0_Ethernet_MAC_PHY_dv, DIR = I
37 PORT fpga_0_Ethernet_MAC_PHY_rx_data_pin = fpga_0_Ethernet_MAC_PHY_rx_data, DIR = I,
38 VEC = [3:0]
39 PORT fpga_0_Ethernet_MAC_PHY_Mii_clk_pin = fpga_0_Ethernet_MAC_PHY_Mii_clk, DIR = IO
```

```
40 PORT fpga_0_Ethernet_MAC_PHY_Mii_data_pin = fpga_0_Ethernet_MAC_PHY_Mii_data, DIR = IO
41 PORT fpga_0_LEDs_4Bit_GPIO_IO_pin = fpga_0_LEDs_4Bit_GPIO_IO, DIR = IO, VEC = [0:3]
42 PORT fpga_0_PushButtons_5Bit_GPIO_IO_pin = fpga_0_PushButtons_5Bit_GPIO_IO, DIR = IO,
43 VEC = [0:4]
44 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Clk_pin =
45 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Clk, DIR = O, VEC = [0:2]
46 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Clkn_pin =
47 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Clkn, DIR = O, VEC = [0:2]
48 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Addr_pin =
49 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Addr, DIR = O, VEC = [0:12]
50 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_BankAddr_pin =
51 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_BankAddr, DIR = O, VEC = [0:1]
52 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CASn_pin =
53 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CASn, DIR = O
54 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_RASn_pin =
55 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_RASn, DIR = O
56 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_WEn_pin =
57 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_WEn, DIR = O
58 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DM_pin =
59 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DM, DIR = O, VEC = [0:7]
60 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DQS_pin =
61 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DQS, DIR = IO, VEC = [0:7]
62 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DQ_pin =
63 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DQ, DIR = IO, VEC = [0:63]
64 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CKE_pin =
65 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CKE, DIR = O
66 PORT fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CSn_pin =
67 fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CSn, DIR = O
68 PORT fpga_0_DDR_CLK_FB = ddr_feedback_s, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
69 PORT fpga_0_DDR_CLK_FB_OUT = ddr_clk_feedback_out_s, DIR = O
70 PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
71 PORT sys_rst_pin = sys_rst_s, DIR = I, RST_POLARITY = 0, SIGIS = RST
72
73
74 BEGIN ppc405
75 PARAMETER INSTANCE = ppc405_0
76 PARAMETER HW_VER = 2.00.c
77 BUS_INTERFACE JTAGPPC = jtagppc_0_0
78 BUS_INTERFACE IPLB = plb
79 BUS_INTERFACE DPLB = plb
80 PORT PLBCLK = sys_clk_s
81 PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ
82 PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ
83 PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ
84 PORT RSTC405RESETCHIP = RSTC405RESETCHIP
85 PORT RSTC405RESETCORE = RSTC405RESETCORE
86 PORT RSTC405RESETSYS = RSTC405RESETSYS
87 PORT CPMC405CLOCK = proc_clk_s
88 END
89
90 BEGIN ppc405
91 PARAMETER INSTANCE = ppc405_1
92 PARAMETER HW_VER = 2.00.c
93 BUS_INTERFACE JTAGPPC = jtagppc_0_1
94 END
95
96 BEGIN jtagppc_cntlr
97 PARAMETER INSTANCE = jtagppc_0
98 PARAMETER HW_VER = 2.00.a
99 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
100 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1
101 END
102
103 BEGIN proc_sys_reset
104 PARAMETER INSTANCE = reset_block
105 PARAMETER HW_VER = 1.00.a
```

```
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Ext_Reset_In = sys_rst_s
 PORT Slowest_sync_clk = sys_clk_s
 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
 PORT Core_Reset_Req = C405RSTCORERESETREQ
 PORT System_Reset_Req = C405RSTSYSRESETREQ
 PORT Rstc405resetchip = RSTC405RESETCHIP
 PORT Rstc405resetcore = RSTC405RESETCORE
 PORT Rstc405resetsys = RSTC405RESETSYS
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Dcm_locked = dcm_1_lock
END

BEGIN plb_v34
 PARAMETER INSTANCE = plb
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT PLB_Clk = sys_clk_s
END

BEGIN opb_v20
 PARAMETER INSTANCE = opb
 PARAMETER HW_VER = 1.10.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb2opb_bridge
 PARAMETER INSTANCE = plb2opb
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_RNG0_BASEADDR = 0x40000000
 PARAMETER C_RNG0_HIGHADDR = 0x7fffffff
 PARAMETER C_NUM_ADDR_RNG = 1
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE MOPB = opb
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE SOPB = opb
 PORT RX = fpga_0_RS232_Uart_1_RX
 PORT TX = fpga_0_RS232_Uart_1_TX
END

BEGIN plb_ethernet
 PARAMETER INSTANCE = Ethernet_MAC
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DMA_PRESENT = 1
 PARAMETER C_IPIF_FIFO_DEPTH = 32768
 PARAMETER C_PLB_CLK_PERIOD_PS = 10000
 PARAMETER C_BASEADDR = 0x80400000
 PARAMETER C_HIGHADDR = 0x8040ffff
 BUS_INTERFACE SPLB = plb
 PORT PHY_rst_n = fpga_0_Ethernet_MAC_PHY_rst_n
```

```
172 PORT PHY_crs = fpga_0_Ethernet_MAC_PHY_crs
173 PORT PHY_col = fpga_0_Ethernet_MAC_PHY_col
174 PORT PHY_tx_data = fpga_0_Ethernet_MAC_PHY_tx_data
175 PORT PHY_tx_en = fpga_0_Ethernet_MAC_PHY_tx_en
176 PORT PHY_tx_clk = fpga_0_Ethernet_MAC_PHY_tx_clk
177 PORT PHY_tx_er = fpga_0_Ethernet_MAC_PHY_tx_er
178 PORT PHY_rx_er = fpga_0_Ethernet_MAC_PHY_rx_er
179 PORT PHY_rx_clk = fpga_0_Ethernet_MAC_PHY_rx_clk
180 PORT PHY_dv = fpga_0_Ethernet_MAC_PHY_dv
181 PORT PHY_rx_data = fpga_0_Ethernet_MAC_PHY_rx_data
182 PORT PHY_Mii_clk = fpga_0_Ethernet_MAC_PHY_Mii_clk
183 PORT PHY_Mii_data = fpga_0_Ethernet_MAC_PHY_Mii_data
184 END
185
186 BEGIN opb_gpio
187 PARAMETER INSTANCE = LEDs_4Bit
188 PARAMETER HW_VER = 3.01.b
189 PARAMETER C_GPIO_WIDTH = 4
190 PARAMETER C_IS_DUAL = 0
191 PARAMETER C_IS_BIDIR = 0
192 PARAMETER C_ALL_INPUTS = 0
193 PARAMETER C_BASEADDR = 0x40020000
194 PARAMETER C_HIGHADDR = 0x4002ffff
195 BUS_INTERFACE SOPB = opb
196 PORT GPIO_IO = fpga_0_LEDs_4Bit_GPIO_IO
197 END
198
199 BEGIN opb_gpio
200 PARAMETER INSTANCE = PushButtons_5Bit
201 PARAMETER HW_VER = 3.01.b
202 PARAMETER C_GPIO_WIDTH = 5
203 PARAMETER C_IS_DUAL = 0
204 PARAMETER C_IS_BIDIR = 1
205 PARAMETER C_ALL_INPUTS = 1
206 PARAMETER C_BASEADDR = 0x40000000
207 PARAMETER C_HIGHADDR = 0x4000ffff
208 BUS_INTERFACE SOPB = opb
209 PORT GPIO_IO = fpga_0_PushButtons_5Bit_GPIO_IO
210 END
211
212 BEGIN plb_ddr
213 PARAMETER INSTANCE = DDR_128MB_16MX64_rank1_row13_col9_cl2_5
214 PARAMETER HW_VER = 2.00.a
215 PARAMETER C_PLB_CLK_PERIOD_PS = 10000
216 PARAMETER C_NUM_BANKS_MEM = 1
217 PARAMETER C_NUM_CLK_PAIRS = 4
218 PARAMETER C_REG_DIMM = 0
219 PARAMETER C_DDR_TMRD = 20000
220 PARAMETER C_DDR_TWR = 20000
221 PARAMETER C_DDR_TRAS = 60000
222 PARAMETER C_DDR_TRC = 90000
223 PARAMETER C_DDR_TRFC = 100000
224 PARAMETER C_DDR_TRCD = 30000
225 PARAMETER C_DDR_TRRD = 20000
226 PARAMETER C_DDR_TRP = 30000
227 PARAMETER C_DDR_AWIDTH = 13
228 PARAMETER C_DDR_COL_AWIDTH = 9
229 PARAMETER C_DDR_BANK_AWIDTH = 2
230 PARAMETER C_DDR_DWIDTH = 64
231 PARAMETER C_MEM0_BASEADDR = 0x00000000
232 PARAMETER C_MEM0_HIGHADDR = 0x07ffffff
233 BUS_INTERFACE SPLB = plb
234 PORT DDR_Addr = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Addr
235 PORT DDR_BankAddr = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_BankAddr
236 PORT DDR_CASn = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CASn
237 PORT DDR_CKE = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CKE
```

```
238  PORT DDR_CSn = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_CSn
239  PORT DDR_RASn = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_RASn
240  PORT DDR_WEn = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_WEn
241  PORT DDR_DM = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DM
242  PORT DDR_DQS = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DQS
243  PORT DDR_DQ = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_DQ
244  PORT DDR_Clk = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Clk
245  & ddr_clk_feedback_out_s
246  PORT DDR_Clkn = fpga_0_DDR_128MB_16MX64_rank1_row13_col9_cl2_5_DDR_Clkn & 0b0
247  PORT Clk90_in = clk_90_s
248  PORT Clk90_in_n = clk_90_n_s
249  PORT PLB_Clk_n = sys_clk_n_s
250  PORT DDR_Clk90_in = ddr_clk_90_s
251  PORT DDR_Clk90_in_n = ddr_clk_90_n_s
252 END
253
254 BEGIN plb_bram_if_cntlr
255  PARAMETER INSTANCE = plb_bram_if_cntlr_1
256  PARAMETER HW_VER = 1.00.b
257  PARAMETER c_plb_clk_period_ps = 10000
258  PARAMETER c_baseaddr = 0xfffe0000
259  PARAMETER c_highaddr = 0xffffffff
260  BUS_INTERFACE SPLB = plb
261  BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
262 END
263
264 BEGIN bram_block
265  PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
266  PARAMETER HW_VER = 1.00.a
267  BUS_INTERFACE PORTA = plb_bram_if_cntlr_1_port
268 END
269
270 BEGIN util_vector_logic
271  PARAMETER INSTANCE = sysclk_inv
272  PARAMETER HW_VER = 1.00.a
273  PARAMETER C_SIZE = 1
274  PARAMETER C_OPERATION = not
275  PORT Op1 = sys_clk_s
276  PORT Res = sys_clk_n_s
277 END
278
279 BEGIN util_vector_logic
280  PARAMETER INSTANCE = clk90_inv
281  PARAMETER HW_VER = 1.00.a
282  PARAMETER C_SIZE = 1
283  PARAMETER C_OPERATION = not
284  PORT Op1 = clk_90_s
285  PORT Res = clk_90_n_s
286 END
287
288 BEGIN util_vector_logic
289  PARAMETER INSTANCE = ddr_clk90_inv
290  PARAMETER HW_VER = 1.00.a
291  PARAMETER C_SIZE = 1
292  PARAMETER C_OPERATION = not
293  PORT Op1 = ddr_clk_90_s
294  PORT Res = ddr_clk_90_n_s
295 END
296
297 BEGIN dcm_module
298  PARAMETER INSTANCE = dcm_0
299  PARAMETER HW_VER = 1.00.a
300  PARAMETER C_CLK0_BUF = TRUE
301  PARAMETER C_CLK90_BUF = TRUE
302  PARAMETER C_CLKFX_BUF = TRUE
303  PARAMETER C_CLKFX_DIVIDE = 1
```

```
304  PARAMETER C_CLKFX_MULTIPLY = 3
305  PARAMETER C_CLKIN_PERIOD = 10.000000
306  PARAMETER C_CLK_FEEDBACK = 1X
307  PARAMETER C_DFS_FREQUENCY_MODE = HIGH
308  PARAMETER C_DLL_FREQUENCY_MODE = LOW
309  PARAMETER C_EXT_RESET_HIGH = 1
310  PORT CLKIN = dcm_clk_s
311  PORT CLK0 = sys_clk_s
312  PORT CLK90 = clk_90_s
313  PORT CLKFX = proc_clk_s
314  PORT CLKFB = sys_clk_s
315  PORT RST = net_gnd
316  PORT LOCKED = dcm_0_lock
317 END
318
319 BEGIN dcm_module
320  PARAMETER INSTANCE = dcm_1
321  PARAMETER HW_VER = 1.00.a
322  PARAMETER C_CLK0_BUF = TRUE
323  PARAMETER C_CLK90_BUF = TRUE
324  PARAMETER C_CLKIN_PERIOD = 10.000000
325  PARAMETER C_CLK_FEEDBACK = 1X
326  PARAMETER C_DLL_FREQUENCY_MODE = LOW
327  PARAMETER C_PHASE_SHIFT = 60
328  PARAMETER C_CLKOUT_PHASE_SHIFT = FIXED
329  PARAMETER C_EXT_RESET_HIGH = 0
330  PORT CLKIN = ddr_feedback_s
331  PORT CLK90 = ddr_clk_90_s
332  PORT CLK0 = dcm_1_FB
333  PORT CLKFB = dcm_1_FB
334  PORT RST = dcm_0_lock
335  PORT LOCKED = dcm_1_lock
336 END
337
338 BEGIN firewall_bitacora
339  PARAMETER INSTANCE = firewall_bitacora_0
340  PARAMETER HW_VER = 2.00.a
341  PARAMETER C_BASEADDR = 0x71000000
342  PARAMETER C_HIGHADDR = 0x7100ffff
343  BUS_INTERFACE SOPB = opb
344 END
```

Archivo: `system.mss` (*Especificaciones de software del microprocesador*)

```
1
2
3  PARAMETER VERSION = 2.2.0
4
5
6 BEGIN OS
7  PARAMETER OS_NAME = xilkernel
8  PARAMETER OS_VER = 3.00.a
9  PARAMETER PROC_INSTANCE = ppc405_0
10 PARAMETER stdout = RS232_Uart_1
11 PARAMETER stdin = RS232_Uart_1
12 PARAMETER config_elf_process = true
13 PARAMETER pthread_stack_size = 5000
14 PARAMETER max_pthreads = 5
15 PARAMETER static_elf_process_table = ((redireccion_thread,1))
16 PARAMETER static_pthread_table = ((redireccion_thread,1))
17 END
18
19 BEGIN OS
20 PARAMETER OS_NAME = standalone
21 PARAMETER OS_VER = 1.00.a
22 PARAMETER PROC_INSTANCE = ppc405_1
23 END
24
25
26 BEGIN PROCESSOR
27 PARAMETER DRIVER_NAME = cpu_ppc405
28 PARAMETER DRIVER_VER = 1.00.a
29 PARAMETER HW_INSTANCE = ppc405_0
30 PARAMETER COMPILER = powerpc-eabi-gcc
31 PARAMETER ARCHIVER = powerpc-eabi-ar
32 PARAMETER CORE_CLOCK_FREQ_HZ = 300000000
33 END
34
35 BEGIN PROCESSOR
36 PARAMETER DRIVER_NAME = cpu_ppc405
37 PARAMETER DRIVER_VER = 1.00.a
38 PARAMETER HW_INSTANCE = ppc405_1
39 PARAMETER COMPILER = powerpc-eabi-gcc
40 PARAMETER ARCHIVER = powerpc-eabi-ar
41 END
42
43
44 BEGIN DRIVER
45 PARAMETER DRIVER_NAME = generic
46 PARAMETER DRIVER_VER = 1.00.a
47 PARAMETER HW_INSTANCE = jtagppc_0
48 END
49
50 BEGIN DRIVER
51 PARAMETER DRIVER_NAME = plbarb
52 PARAMETER DRIVER_VER = 1.01.a
53 PARAMETER HW_INSTANCE = plb
54 END
55
56 BEGIN DRIVER
57 PARAMETER DRIVER_NAME = opbarb
58 PARAMETER DRIVER_VER = 1.02.a
59 PARAMETER HW_INSTANCE = opb
```

```
60 END
61
62 BEGIN DRIVER
63  PARAMETER DRIVER_NAME = plb2opb
64  PARAMETER DRIVER_VER = 1.00.a
65  PARAMETER HW_INSTANCE = plb2opb
66 END
67
68 BEGIN DRIVER
69  PARAMETER DRIVER_NAME = uartlite
70  PARAMETER DRIVER_VER = 1.01.a
71  PARAMETER HW_INSTANCE = RS232_Uart_1
72 END
73
74 BEGIN DRIVER
75  PARAMETER DRIVER_NAME = emac
76  PARAMETER DRIVER_VER = 1.00.e
77  PARAMETER HW_INSTANCE = Ethernet_MAC
78 END
79
80 BEGIN DRIVER
81  PARAMETER DRIVER_NAME = gpio
82  PARAMETER DRIVER_VER = 2.01.a
83  PARAMETER HW_INSTANCE = LEDs_4Bit
84 END
85
86 BEGIN DRIVER
87  PARAMETER DRIVER_NAME = gpio
88  PARAMETER DRIVER_VER = 2.01.a
89  PARAMETER HW_INSTANCE = PushButtons_5Bit
90 END
91
92 BEGIN DRIVER
93  PARAMETER DRIVER_NAME = ddr
94  PARAMETER DRIVER_VER = 1.00.b
95  PARAMETER HW_INSTANCE = DDR_128MB_16MX64_rank1_row13_col9_cl2_5
96 END
97
98 BEGIN DRIVER
99  PARAMETER DRIVER_NAME = bram
100  PARAMETER DRIVER_VER = 1.00.a
101  PARAMETER HW_INSTANCE = plb_bram_if_cntlr_1
102 END
103
104 BEGIN DRIVER
105  PARAMETER DRIVER_NAME = generic
106  PARAMETER DRIVER_VER = 1.00.a
107  PARAMETER HW_INSTANCE = firewall_bitacora_0
108 END
```

Apéndice B

Codigo fuente del Firewall

Archivo: `firewall_bitácora.vhd`

```
1 Este archivo fue generado de forma automatica por las
2 herramientas de desarrollo
3
4 library ieee;
5 use ieee.std_logic_1164.all;
6 use ieee.std_logic_arith.all;
7 use ieee.std_logic_unsigned.all;
8
9 library proc_common_v2_00_a;
10 use proc_common_v2_00_a.proc_common_pkg.all;
11 use proc_common_v2_00_a.ipif_pkg.all;
12 library opb_ipif_v3_01_c;
13 use opb_ipif_v3_01_c.all;
14
15 library firewall_bitacora_v2_00_a;
16 use firewall_bitacora_v2_00_a.all;
17
18 ------------------------------------------------------------------------------
19 -- Entity section
20 ------------------------------------------------------------------------------
21 -- Definition of Generics:
22 --   C_BASEADDR              -- User logic base address
23 --   C_HIGHADDR              -- User logic high address
24 --   C_OPB_AWIDTH            -- OPB address bus width
25 --   C_OPB_DWIDTH            -- OPB data bus width
26 --   C_USER_ID_CODE          -- User ID to place in MIR/Reset register
27 --   C_FAMILY                -- Target FPGA architecture
28 --
29 -- Definition of Ports:
30 --   OPB_Clk                 -- OPB Clock
31 --   OPB_Rst                 -- OPB Reset
32 --   Sl_DBus                 -- Slave data bus
33 --   Sl_errAck               -- Slave error acknowledge
34 --   Sl_retry                -- Slave retry
35 --   Sl_toutSup              -- Slave timeout suppress
36 --   Sl_xferAck              -- Slave transfer acknowledge
37 --   OPB_ABus                -- OPB address bus
38 --   OPB_BE                  -- OPB byte enable
39 --   OPB_DBus                -- OPB data bus
40 --   OPB_RNW                 -- OPB read/not write
```

```
41 --    OPB_select                -- OPB select
42 --    OPB_seqAddr               -- OPB sequential address
43 ------------------------------------------------------------------------------
44
45 entity firewall_bitacora is
46   generic
47   (
48     -- ADD USER GENERICS BELOW THIS LINE ---------------
49     --USER generics added here
50     -- ADD USER GENERICS ABOVE THIS LINE ---------------
51
52     -- DO NOT EDIT BELOW THIS LINE ---------------------
53     -- Bus protocol parameters, do not add to or delete
54     C_BASEADDR                : std_logic_vector     := X"00000000";
55     C_HIGHADDR                : std_logic_vector     := X"0000FFFF";
56     C_OPB_AWIDTH              : integer              := 32;
57     C_OPB_DWIDTH              : integer              := 32;
58     C_USER_ID_CODE            : integer              := 3;
59     C_FAMILY                  : string               := "virtex2p"
60     -- DO NOT EDIT ABOVE THIS LINE ---------------------
61   );
62   port
63   (
64     -- ADD USER PORTS BELOW THIS LINE ------------------
65     --USER ports added here
66     -- ADD USER PORTS ABOVE THIS LINE ------------------
67
68     -- DO NOT EDIT BELOW THIS LINE ---------------------
69     -- Bus protocol ports, do not add to or delete
70     OPB_Clk                 : in  std_logic;
71     OPB_Rst                 : in  std_logic;
72     Sl_DBus                 : out std_logic_vector(0 to C_OPB_DWIDTH-1);
73     Sl_errAck               : out std_logic;
74     Sl_retry                : out std_logic;
75     Sl_toutSup              : out std_logic;
76     Sl_xferAck              : out std_logic;
77     OPB_ABus                : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
78     OPB_BE                  : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
79     OPB_DBus                : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
80     OPB_RNW                 : in  std_logic;
81     OPB_select              : in  std_logic;
82     OPB_seqAddr             : in  std_logic
83     -- DO NOT EDIT ABOVE THIS LINE ---------------------
84   );
85
86   attribute SIGIS : string;
87   attribute SIGIS of OPB_Clk       : signal is "Clk";
88   attribute SIGIS of OPB_Rst       : signal is "Rst";
89
90 end entity firewall_bitacora;
91
92 ------------------------------------------------------------------------------
93 -- Architecture section
94 ------------------------------------------------------------------------------
95
96 architecture IMP of firewall_bitacora is
97
98   ---------------------------------------------
99   -- Constant: array of address range identifiers
100  ---------------------------------------------
101  constant ARD_ID_ARRAY                    : INTEGER_ARRAY_TYPE    :=
102    (
103      0  => USER_00,     -- user logic S/W register address space
104      1  => IPIF_RST     -- include IPIF S/W Reset/MIR service
105    );
106
```

```
107  ------------------------------------------
108  -- Constant: array of address pairs for each address range
109  ------------------------------------------
110  constant ZERO_ADDR_PAD       : std_logic_vector(0 to 64-C_OPB_AWIDTH-1) := (others => '0');
111
112  constant USER_BASEADDR       : std_logic_vector     := C_BASEADDR or X"00000000";
113  constant USER_HIGHADDR       : std_logic_vector     := C_BASEADDR or X"000000FF";
114
115  constant RST_BASEADDR        : std_logic_vector     := C_BASEADDR or X"00000100";
116  constant RST_HIGHADDR        : std_logic_vector     := C_BASEADDR or X"000001FF";
117
118  constant ARD_ADDR_RANGE_ARRAY  : SLV64_ARRAY_TYPE     :=
119    (
120      ZERO_ADDR_PAD & USER_BASEADDR,     -- user logic base address
121      ZERO_ADDR_PAD & USER_HIGHADDR,     -- user logic high address
122      ZERO_ADDR_PAD & RST_BASEADDR,      -- MIR/Reset register base address
123      ZERO_ADDR_PAD & RST_HIGHADDR       -- MIR/Reset register high address
124    );
125
126  ------------------------------------------
127  -- Constant: array of data widths for each target address range
128  ------------------------------------------
129  constant USER_DWIDTH             : integer              := 32;
130
131  constant ARD_DWIDTH_ARRAY        : INTEGER_ARRAY_TYPE   :=
132    (
133      0  => USER_DWIDTH,           -- user logic data width
134      1  => C_OPB_DWIDTH           -- MIR/Reset register data width
135    );
136  ------------------------------------------
137  -- Constant: array of desired number of chip enables for each address range
138  ------------------------------------------
139  constant USER_NUM_CE             : integer              := 12;
140  constant ARD_NUM_CE_ARRAY        : INTEGER_ARRAY_TYPE   :=
141    (
142      0  => pad_power2(USER_NUM_CE),     -- user logic number of CEs
143      1  => 1                            -- MIR/Reset register - 1 CE
144    );
145
146  ------------------------------------------
147  -- Constant: array of unique properties for each address range
148  ------------------------------------------
149  constant ARD_DEPENDENT_PROPS_ARRAY       : DEPENDENT_PROPS_ARRAY_TYPE :=
150    (
151      0  => (others => 0),    -- user logic slave space dependent properties (none defined)
152      1  => (others => 0)     -- IPIF reset/mir dependent properties (none defined)
153    );
154
155  ------------------------------------------
156  -- Constant: pipeline mode
157  -- 1 = include OPB-In pipeline registers
158  -- 2 = include IP pipeline registers
159  -- 3 = include OPB-In and IP pipeline registers
160  -- 4 = include OPB-Out pipeline registers
161  -- 5 = include OPB-In and OPB-Out pipeline registers
162  -- 6 = include IP and OPB-Out pipeline registers
163  -- 7 = include OPB-In, IP, and OPB-Out pipeline registers
164  -- Note:
165  -- only mode 4, 5, 7 are supported for this release
166  ------------------------------------------
167  constant PIPELINE_MODEL          : integer  := 5;
168
169  ------------------------------------------
170  -- Constant: user core ID code
171  ------------------------------------------
172  constant DEV_BLK_ID              : integer := C_USER_ID_CODE;
```

```
173
174  ------------------------------------------
175  -- Constant: enable MIR/Reset register
176  ------------------------------------------
177  constant DEV_MIR_ENABLE         : integer := 1;
178
179  ------------------------------------------
180  -- Constant: array of IP interrupt mode
181  -- 1 = Active-high interrupt condition
182  -- 2 = Active-low interrupt condition
183  -- 3 = Active-high pulse interrupt event
184  -- 4 = Active-low pulse interrupt event
185  -- 5 = Positive-edge interrupt event
186  -- 6 = Negative-edge interrupt event
187  ------------------------------------------
188  constant IP_INTR_MODE_ARRAY       : INTEGER_ARRAY_TYPE   :=
189    (
190      0  => 0  -- not used
191    );
192
193  ------------------------------------------
194  -- Constant: enable device burst
195  ------------------------------------------
196  constant DEV_BURST_ENABLE         : integer    := 0;
197
198  ------------------------------------------
199  -- Constant: include address counter for burst transfers
200  ------------------------------------------
201  constant INCLUDE_ADDR_CNTR        : integer    := 0;
202
203  ------------------------------------------
204  -- Constant: include write buffer that decouples OPB and IPIC write transactions
205  ------------------------------------------
206  constant INCLUDE_WR_BUF           : integer    := 0;
207
208  ------------------------------------------
209  -- Constant: index for CS/CE
210  ------------------------------------------
211  constant USER00_CS_INDEX      : integer  := get_id_index(ARD_ID_ARRAY, USER_00);
212
213  constant USER00_CE_INDEX      : integer
214  := calc_start_ce_index(ARD_NUM_CE_ARRAY, USER00_CS_INDEX);
215
216  ------------------------------------------
217  -- IP Interconnect (IPIC) signal declarations -- do not delete
218  -- prefix 'i' stands for IPIF while prefix 'u' stands for user logic
219  -- typically user logic will be hooked up to IPIF directly via i<sig>
220  -- unless signal slicing and muxing are needed via u<sig>
221  ------------------------------------------
222  signal iBus2IP_RdCE         : std_logic_vector(0 to calc_num_ce(ARD_NUM_CE_ARRAY)-1);
223  signal iBus2IP_WrCE         : std_logic_vector(0 to calc_num_ce(ARD_NUM_CE_ARRAY)-1);
224  signal iBus2IP_Data         : std_logic_vector(0 to C_OPB_DWIDTH-1);
225  signal iBus2IP_BE           : std_logic_vector(0 to C_OPB_DWIDTH/8-1);
226  signal iIP2Bus_Data         : std_logic_vector(0 to C_OPB_DWIDTH-1)   := (others => '0');
227  signal iIP2Bus_Ack          : std_logic   := '0';
228  signal iIP2Bus_Error        : std_logic   := '0';
229  signal iIP2Bus_Retry        : std_logic   := '0';
230  signal iIP2Bus_ToutSup      : std_logic   := '0';
231  signal ENABLE_POSTED_WRITE : std_logic_vector(0 to ARD_ID_ARRAY'length-1)
232  := (others => '0');
233
234 posted write behavior
235
236 signal ZERO_IP2RFIFO_Data  :
237 std_logic_vector(0 to ARD_DWIDTH_ARRAY
238 (get_id_index_iboe(ARD_ID_ARRAY,IPIF_RDFIFO_DATA))-1)   := (others => '0');
```

```
239
240  -- work around for XST not taking (others => '0') in port mapping
241
242  signal ZERO_WFIFO2IP_Data    : std_logic_vector
243  (0 to ARD_DWIDTH_ARRAY(get_id_index_iboe
244  (ARD_ID_ARRAY, IPIF_WRFIFO_DATA))-1)   := (others => '0');
245
246  for XST not taking (others => '0') in port mapping
247   signal ZERO_IP2Bus_IntrEvent    : std_logic_vector(0 to IP_INTR_MODE_ARRAY'length-1)
248     := (others => '0');
249   -- work around for XST not taking (others => '0') in port mapping
250
251   signal iBus2IP_Clk          : std_logic;
252   signal iBus2IP_Reset        : std_logic;
253   signal uBus2IP_Data         : std_logic_vector(0 to USER_DWIDTH-1);
254   signal uBus2IP_BE           : std_logic_vector(0 to USER_DWIDTH/8-1);
255   signal uBus2IP_RdCE         : std_logic_vector(0 to USER_NUM_CE-1);
256   signal uBus2IP_WrCE         : std_logic_vector(0 to USER_NUM_CE-1);
257   signal uIP2Bus_Data         : std_logic_vector(0 to USER_DWIDTH-1);
258
259 begin
260
261   ------------------------------------------
262   -- instantiate the OPB IPIF
263   ------------------------------------------
264   OPB_IPIF_I : entity opb_ipif_v3_01_c.opb_ipif
265     generic map
266     (
267       C_ARD_ID_ARRAY                => ARD_ID_ARRAY,
268       C_ARD_ADDR_RANGE_ARRAY        => ARD_ADDR_RANGE_ARRAY,
269       C_ARD_DWIDTH_ARRAY            => ARD_DWIDTH_ARRAY,
270       C_ARD_NUM_CE_ARRAY            => ARD_NUM_CE_ARRAY,
271       C_ARD_DEPENDENT_PROPS_ARRAY   => ARD_DEPENDENT_PROPS_ARRAY,
272       C_PIPELINE_MODEL              => PIPELINE_MODEL,
273       C_DEV_BLK_ID                  => DEV_BLK_ID,
274       C_DEV_MIR_ENABLE              => DEV_MIR_ENABLE,
275       C_OPB_AWIDTH                  => C_OPB_AWIDTH,
276       C_OPB_DWIDTH                  => C_OPB_DWIDTH,
277       C_FAMILY                      => C_FAMILY,
278       C_IP_INTR_MODE_ARRAY          => IP_INTR_MODE_ARRAY,
279       C_DEV_BURST_ENABLE            => DEV_BURST_ENABLE,
280       C_INCLUDE_ADDR_CNTR           => INCLUDE_ADDR_CNTR,
281       C_INCLUDE_WR_BUF              => INCLUDE_WR_BUF
282     )
283     port map
284     (
285       OPB_select                    => OPB_select,
286       OPB_DBus                      => OPB_DBus,
287       OPB_ABus                      => OPB_ABus,
288       OPB_BE                        => OPB_BE,
289       OPB_RNW                       => OPB_RNW,
290       OPB_seqAddr                   => OPB_seqAddr,
291       Sln_DBus                      => Sl_DBus,
292       Sln_xferAck                   => Sl_xferAck,
293       Sln_errAck                    => Sl_errAck,
294       Sln_retry                     => Sl_retry,
295       Sln_toutSup                   => Sl_toutSup,
296       Bus2IP_CS                     => open,
297       Bus2IP_CE                     => open,
298       Bus2IP_RdCE                   => iBus2IP_RdCE,
299       Bus2IP_WrCE                   => iBus2IP_WrCE,
300       Bus2IP_Data                   => iBus2IP_Data,
301       Bus2IP_Addr                   => open,
302       Bus2IP_AddrValid              => open,
303       Bus2IP_BE                     => iBus2IP_BE,
304       Bus2IP_RNW                    => open,
```

```
305       Bus2IP_Burst                    => open,
306       IP2Bus_Data                     => iIP2Bus_Data,
307       IP2Bus_Ack                      => iIP2Bus_Ack,
308       IP2Bus_AddrAck                  => '0',
309       IP2Bus_Error                    => iIP2Bus_Error,
310       IP2Bus_Retry                    => iIP2Bus_Retry,
311       IP2Bus_ToutSup                  => iIP2Bus_ToutSup,
312       IP2Bus_PostedWrInh              => ENABLE_POSTED_WRITE,
313       IP2RFIFO_Data                   => ZERO_IP2RFIFO_Data,
314       IP2RFIFO_WrMark                 => '0',
315       IP2RFIFO_WrRelease              => '0',
316       IP2RFIFO_WrReq                  => '0',
317       IP2RFIFO_WrRestore              => '0',
318       RFIFO2IP_AlmostFull             => open,
319       RFIFO2IP_Full                   => open,
320       RFIFO2IP_Vacancy                => open,
321       RFIFO2IP_WrAck                  => open,
322       IP2WFIFO_RdMark                 => '0',
323       IP2WFIFO_RdRelease              => '0',
324       IP2WFIFO_RdReq                  => '0',
325       IP2WFIFO_RdRestore              => '0',
326       WFIFO2IP_AlmostEmpty            => open,
327       WFIFO2IP_Data                   => ZERO_WFIFO2IP_Data,
328       WFIFO2IP_Empty                  => open,
329       WFIFO2IP_Occupancy              => open,
330       WFIFO2IP_RdAck                  => open,
331       IP2Bus_IntrEvent                => ZERO_IP2Bus_IntrEvent,
332       IP2INTC_Irpt                    => open,
333       Freeze                          => '0',
334       Bus2IP_Freeze                   => open,
335       OPB_Clk                         => OPB_Clk,
336       Bus2IP_Clk                      => iBus2IP_Clk,
337       IP2Bus_Clk                      => '0',
338       Reset                           => OPB_Rst,
339       Bus2IP_Reset                    => iBus2IP_Reset
340     );
341
342   ------------------------------------------
343   -- instantiate the User Logic
344   ------------------------------------------
345   USER_LOGIC_I : entity firewall_bitacora_v2_00_a.user_logic
346     generic map
347     (
348       -- MAP USER GENERICS BELOW THIS LINE ---------------
349       --USER generics mapped here
350       -- MAP USER GENERICS ABOVE THIS LINE ---------------
351
352       C_DWIDTH                        => USER_DWIDTH,
353       C_NUM_CE                        => USER_NUM_CE
354     )
355     port map
356     (
357       -- MAP USER PORTS BELOW THIS LINE ------------------
358       --USER ports mapped here
359       -- MAP USER PORTS ABOVE THIS LINE ------------------
360
361       Bus2IP_Clk                      => iBus2IP_Clk,
362       Bus2IP_Reset                    => iBus2IP_Reset,
363       Bus2IP_Data                     => uBus2IP_Data,
364       Bus2IP_BE                       => uBus2IP_BE,
365       Bus2IP_RdCE                     => uBus2IP_RdCE,
366       Bus2IP_WrCE                     => uBus2IP_WrCE,
367       IP2Bus_Data                     => uIP2Bus_Data,
368       IP2Bus_Ack                      => iIP2Bus_Ack,
369       IP2Bus_Retry                    => iIP2Bus_Retry,
370       IP2Bus_Error                    => iIP2Bus_Error,
```

```
371       IP2Bus_ToutSup                => iIP2Bus_ToutSup
372     );
373
374   ----------------------------------------------
375   -- hooking up signal slicing
376   ----------------------------------------------
377   uBus2IP_BE <= iBus2IP_BE(0 to USER_DWIDTH/8-1);
378   uBus2IP_Data <= iBus2IP_Data(0 to USER_DWIDTH-1);
379   uBus2IP_RdCE <= iBus2IP_RdCE(USER00_CE_INDEX to USER00_CE_INDEX+USER_NUM_CE-1);
380   uBus2IP_WrCE <= iBus2IP_WrCE(USER00_CE_INDEX to USER00_CE_INDEX+USER_NUM_CE-1);
381   iIP2Bus_Data(0 to USER_DWIDTH-1) <= uIP2Bus_Data;
382
383 end IMP;
```

Archivo: `user_logic.vhd`

```
1
2 Este archivo fue generado automaticamente por las
3 herramientas de desarollo, y modificado para su
4 utilizacion en el proyecto
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.std_logic_arith.all;
9 use ieee.std_logic_unsigned.all;
10
11 library proc_common_v2_00_a;
12 use proc_common_v2_00_a.proc_common_pkg.all;
13 -- DO NOT EDIT ABOVE THIS LINE ------------------
14
15 --USER libraries added here
16
17 ------------------------------------------------------------------------------
18 -- Entity section
19 ------------------------------------------------------------------------------
20 -- Definition of Generics:
21 --   C_DWIDTH                  -- User logic data bus width
22 --   C_NUM_CE                  -- User logic chip enable bus width
23 --
24 -- Definition of Ports:
25 --   Bus2IP_Clk                -- Bus to IP clock
26 --   Bus2IP_Reset              -- Bus to IP reset
27 --   Bus2IP_Data               -- Bus to IP data bus for user logic
28 --   Bus2IP_BE                 -- Bus to IP byte enables for user logic
29 --   Bus2IP_RdCE               -- Bus to IP read chip enable for user logic
30 --   Bus2IP_WrCE               -- Bus to IP write chip enable for user logic
31 --   IP2Bus_Data               -- IP to Bus data bus for user logic
32 --   IP2Bus_Ack                -- IP to Bus acknowledgement
33 --   IP2Bus_Retry              -- IP to Bus retry response
34 --   IP2Bus_Error              -- IP to Bus error response
35 --   IP2Bus_ToutSup            -- IP to Bus timeout suppress
36 ------------------------------------------------------------------------------
37
38 entity user_logic is
39   generic
40   (
41     -- ADD USER GENERICS BELOW THIS LINE ---------------
42     --USER generics added here
43     -- ADD USER GENERICS ABOVE THIS LINE ---------------
44
```

```
45     -- DO NOT EDIT BELOW THIS LINE ---------------------
46     -- Bus protocol parameters, do not add to or delete
47     C_DWIDTH                       : integer              := 32;
48     C_NUM_CE                       : integer              := 12
49     -- DO NOT EDIT ABOVE THIS LINE ---------------------
50   );
51   port
52   (
53     -- ADD USER PORTS BELOW THIS LINE ---------------------
54     --USER ports added here
55     -- ADD USER PORTS ABOVE THIS LINE ---------------------
56
57     -- DO NOT EDIT BELOW THIS LINE ---------------------
58     -- Bus protocol ports, do not add to or delete
59     Bus2IP_Clk                     : in  std_logic;
60     Bus2IP_Reset                   : in  std_logic;
61     Bus2IP_Data                    : in  std_logic_vector(C_DWIDTH-1 downto 0);
62     Bus2IP_BE                      : in  std_logic_vector(C_DWIDTH/8-1 downto 0);
63     Bus2IP_RdCE                    : in  std_logic_vector(C_NUM_CE-1 downto 0);
64     Bus2IP_WrCE                    : in  std_logic_vector(C_NUM_CE-1 downto 0);
65     IP2Bus_Data                    : out std_logic_vector(C_DWIDTH-1 downto 0);
66     IP2Bus_Ack                     : out std_logic;
67     IP2Bus_Retry                   : out std_logic;
68     IP2Bus_Error                   : out std_logic;
69     IP2Bus_ToutSup                 : out std_logic
70     -- DO NOT EDIT ABOVE THIS LINE ---------------------
71   );
72 end entity user_logic;
73
74 ------------------------------------------------------------------------------
75 -- Architecture section
76 ------------------------------------------------------------------------------
77
78 architecture IMP of user_logic is
79
80   --USER signal declarations added here, as needed for user logic
81
82    component Completo is
83    port (
84       clk             : in  std_logic;
85       reset           : in  std_logic;
86       -- HEADERS DE ENTRADA
87       data_H_in_1     : in  std_logic_vector(31 downto 0);
88       data_H_in_2     : in  std_logic_vector(31 downto 0);
89       data_H_in_3     : in  std_logic_vector(31 downto 0);
90       -- hEADERS DE SALIDA
91       data_H_out_1    : in  std_logic_vector(31 downto 0);
92       data_H_out_2    : in  std_logic_vector(31 downto 0);
93       data_H_out_3    : in  std_logic_vector(31 downto 0);
94       -- REGLAS
95       Reg_1           : in  std_logic_vector(31 downto 0);
96       Reg_2           : in  std_logic_vector(31 downto 0);
97       Reg_3           : in  std_logic_vector(31 downto 0);
98       -- DIRECCION
99       dir             : in  std_logic_vector(3 downto 0);
100      -- REGISTRO DE INTERRUPCIONES
101      -- bita - wr_H_in - wr_H_out - wr_R_in - wr_R_out
102      --  4   -    3    -    2     -    1    -    0
103      interrupt       : in  std_logic_vector(4 downto 0);
104      -- SENALES DE FINALIZACION
105      rdy_in          : out std_logic_vector(1 downto 0);
106      rdy_out         : out std_logic_vector(1 downto 0);
107      bitac           : out std_logic_vector(79 downto 0)
108    );
109    end component Completo;
110
```

```
111  ------------------------------------------
112  -- Signals for user logic slave model s/w accessible register example
113  ------------------------------------------
114  signal slv_reg0                         : std_logic_vector(C_DWIDTH-1 downto 0);
115  signal slv_reg1                         : std_logic_vector(C_DWIDTH-1 downto 0);
116  signal slv_reg2                         : std_logic_vector(C_DWIDTH-1 downto 0);
117  signal slv_reg3                         : std_logic_vector(C_DWIDTH-1 downto 0);
118  signal slv_reg4                         : std_logic_vector(C_DWIDTH-1 downto 0);
119  signal slv_reg5                         : std_logic_vector(C_DWIDTH-1 downto 0);
120  signal slv_reg6                         : std_logic_vector(C_DWIDTH-1 downto 0);
121  signal slv_reg7                         : std_logic_vector(C_DWIDTH-1 downto 0);
122  signal slv_reg8                         : std_logic_vector(C_DWIDTH-1 downto 0);
123  signal slv_reg9                         : std_logic_vector(C_DWIDTH-1 downto 0);
124  signal slv_reg10                        : std_logic_vector(C_DWIDTH-1 downto 0);
125  signal slv_reg11                        : std_logic_vector(C_DWIDTH-1 downto 0);
126  signal slv_reg_write_select             : std_logic_vector(11 downto 0);
127  signal slv_reg_read_select              : std_logic_vector(11 downto 0);
128  signal slv_ip2bus_data                  : std_logic_vector(C_DWIDTH-1 downto 0);
129  signal slv_read_ack                     : std_logic;
130  signal slv_write_ack                    : std_logic;
131
132  signal rdyOut                           : std_logic_vector(1 downto 0);
133  signal bita                             : std_logic_vector(79 downto 0);
134  signal rst                              : std_logic;
135
136 begin
137
138  --USER logic implementation added here
139
140   U0: Completo
141   port map(
142      clk             => Bus2IP_Clk,
143      reset           => rst,
144      data_H_in_1     => slv_reg0,
145      data_H_in_2     => slv_reg1,
146      data_H_in_3     => slv_reg2,
147      data_H_out_1    => slv_reg0,
148      data_H_out_2    => slv_reg1,
149      data_H_out_3    => slv_reg2,
150      Reg_1           => slv_reg3,
151      Reg_2           => slv_reg4,
152      Reg_3           => slv_reg5,
153      dir             => slv_reg6(3 downto 0),
154      interrupt       => slv_reg7(4 downto 0),
155      rdy_in          => slv_reg8(1 downto 0),
156      rdy_out         => rdyOut,
157      bitac           => bita
158   );
159
160   slv_reg9 <= bita(79 downto 48);
161   slv_reg10<= bita(47 downto 16);
162   slv_reg11(15 downto 0)<= bita(15 downto 0);
163
164  ------------------------------------------
165  -- Example code to read/write user logic slave model s/w accessible registers
166  --
167  -- Note:
168  -- The example code presented here is to show you one way of reading/writing
169  -- software accessible registers implemented in the user logic slave model.
170  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
171  -- to one software accessible register by the top level template. For example,
172  -- if you have four 32 bit software accessible registers in the user logic, you
173  -- are basically operating on the following memory mapped registers:
174  --
175  --     Bus2IP_WrCE or    Memory Mapped
176  --         Bus2IP_RdCE    Register
```

```
177  --              "1000"   C_BASEADDR + 0x0
178  --              "0100"   C_BASEADDR + 0x4
179  --              "0010"   C_BASEADDR + 0x8
180  --              "0001"   C_BASEADDR + 0xC
181  --
182  ------------------------------------------
183  slv_reg_write_select <= Bus2IP_WrCE(11 downto 0);
184  slv_reg_read_select  <= Bus2IP_RdCE(11 downto 0);
185
186  slv_write_ack        <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1)
187  or Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or
188  Bus2IP_WrCE(5) or Bus2IP_WrCE(6) or Bus2IP_WrCE(7) or Bus2IP_WrCE(8)
189  or Bus2IP_WrCE(9) or Bus2IP_WrCE(10) or Bus2IP_WrCE(11);
190
191  slv_read_ack         <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1)
192  or Bus2IP_RdCE(2) or Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or
193  Bus2IP_RdCE(5) or Bus2IP_RdCE(6) or Bus2IP_RdCE(7) or Bus2IP_RdCE(8)
194  or Bus2IP_RdCE(9) or Bus2IP_RdCE(10) or Bus2IP_RdCE(11);
195
196  -- implement slave model register(s)
197  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
198  begin
199
200    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
201      if Bus2IP_Reset = '1' then
202        slv_reg0 <= (others => '0');
203        slv_reg1 <= (others => '0');
204        slv_reg2 <= (others => '0');
205        slv_reg3 <= (others => '0');
206        slv_reg4 <= (others => '0');
207        slv_reg5 <= (others => '0');
208        slv_reg6 <= (others => '0');
209        slv_reg7 <= (others => '0');
210      else
211        case slv_reg_write_select is
212          when "100000000000" =>
213            for byte_index in 0 to (C_DWIDTH/8)-1 loop
214              if ( Bus2IP_BE(byte_index) = '1' ) then
215                slv_reg0(byte_index*8+7 downto byte_index*8) <=
216                Bus2IP_Data(byte_index*8+7 downto byte_index*8);
217              end if;
218            end loop;
219          when "010000000000" =>
220            for byte_index in 0 to (C_DWIDTH/8)-1 loop
221              if ( Bus2IP_BE(byte_index) = '1' ) then
222                slv_reg1(byte_index*8+7 downto byte_index*8)
223                 <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
224              end if;
225            end loop;
226          when "001000000000" =>
227            for byte_index in 0 to (C_DWIDTH/8)-1 loop
228              if ( Bus2IP_BE(byte_index) = '1' ) then
229                slv_reg2(byte_index*8+7 downto byte_index*8)
230                <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
231              end if;
232            end loop;
233          when "000100000000" =>
234            for byte_index in 0 to (C_DWIDTH/8)-1 loop
235              if ( Bus2IP_BE(byte_index) = '1' ) then
236                slv_reg3(byte_index*8+7 downto byte_index*8)
237                <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
238              end if;
239            end loop;
240          when "000010000000" =>
241            for byte_index in 0 to (C_DWIDTH/8)-1 loop
242              if ( Bus2IP_BE(byte_index) = '1' ) then
```

```
243                 slv_reg4(byte_index*8+7 downto byte_index*8)
244                 <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
245               end if;
246             end loop;
247           when "000001000000" =>
248             for byte_index in 0 to (C_DWIDTH/8)-1 loop
249               if ( Bus2IP_BE(byte_index) = '1' ) then
250                 slv_reg5(byte_index*8+7 downto byte_index*8)
251                 <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
252               end if;
253             end loop;
254           when "000000100000" =>
255             for byte_index in 0 to (C_DWIDTH/8)-1 loop
256               if ( Bus2IP_BE(byte_index) = '1' ) then
257                 slv_reg6(byte_index*8+7 downto byte_index*8)
258                 <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
259               end if;
260             end loop;
261           when "000000010000" =>
262             for byte_index in 0 to (C_DWIDTH/8)-1 loop
263               if ( Bus2IP_BE(byte_index) = '1' ) then
264                 slv_reg7(byte_index*8+7 downto byte_index*8)
265                 <= Bus2IP_Data(byte_index*8+7 downto byte_index*8);
266               end if;
267             end loop;
268           when others => null;
269         end case;
270       end if;
271     end if;
272
273   end process SLAVE_REG_WRITE_PROC;
274
275   -- implement slave model register read mux
276   SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0, slv_reg1,
277   slv_reg2, slv_reg3,slv_reg4,slv_reg5, slv_reg6, slv_reg7, slv_reg8, slv_reg9,
278   slv_reg10, slv_reg11 ) is
279   begin
280
281     case slv_reg_read_select is
282       when "100000000000" => slv_ip2bus_data <= slv_reg0;
283       when "010000000000" => slv_ip2bus_data <= slv_reg1;
284       when "001000000000" => slv_ip2bus_data <= slv_reg2;
285       when "000100000000" => slv_ip2bus_data <= slv_reg3;
286       when "000010000000" => slv_ip2bus_data <= slv_reg4;
287       when "000001000000" => slv_ip2bus_data <= slv_reg5;
288       when "000000100000" => slv_ip2bus_data <= slv_reg6;
289       when "000000010000" => slv_ip2bus_data <= slv_reg7;
290       when "000000001000" => slv_ip2bus_data <= slv_reg8;
291       when "000000000100" => slv_ip2bus_data <= slv_reg9;
292       when "000000000010" => slv_ip2bus_data <= slv_reg10;
293       when "000000000001" => slv_ip2bus_data <= slv_reg11;
294       when others => slv_ip2bus_data <= (others => '0');
295     end case;
296
297   end process SLAVE_REG_READ_PROC;
298
299   ------------------------------------------
300   -- Example code to drive IP to Bus signals
301   ------------------------------------------
302   IP2Bus_Data        <= slv_ip2bus_data;
303
304   IP2Bus_Ack         <= slv_write_ack or slv_read_ack;
305   IP2Bus_Error       <= '0';
306   IP2Bus_Retry       <= '0';
307   IP2Bus_ToutSup     <= '0';
308
```

```
309 end IMP;
```

Archivo: `Completo.vhd`

```
 1
 2 library IEEE;
 3 use IEEE.STD_LOGIC_1164.ALL;
 4 use IEEE.STD_LOGIC_ARITH.ALL;
 5 use IEEE.STD_LOGIC_UNSIGNED.ALL;
 6
 7 entity Completo is
 8    port (
 9       clk             : in  std_logic;
10       reset           : in  std_logic;
11       -- HEADERS DE ENTRADA
12       data_H_in_1     : in  std_logic_vector(31 downto 0);
13       data_H_in_2     : in  std_logic_vector(31 downto 0);
14       data_H_in_3     : in  std_logic_vector(31 downto 0);
15       -- HEADERS DE SALIDA
16       data_H_out_1    : in  std_logic_vector(31 downto 0);
17       data_H_out_2    : in  std_logic_vector(31 downto 0);
18       data_H_out_3    : in  std_logic_vector(31 downto 0);
19       -- REGLAS
20       Reg_1           : in  std_logic_vector(31 downto 0);
21       Reg_2           : in  std_logic_vector(31 downto 0);
22       Reg_3           : in  std_logic_vector(31 downto 0);
23       -- DIRECCION
24       dir             : in  std_logic_vector(3 downto 0);
25       -- REGISTRO DE INTERRUPCIONES
26       -- bita - wr_H_in - wr_H_out - wr_R_in - wr_R_out
27       --  4   -    3    -    2     -    1    -    0
28       interrupt       : in  std_logic_vector(4 downto 0);
29       -- SENALES DE FINALIZACION
30       rdy_in          : out std_logic_vector(1 downto 0);
31       rdy_out         : out std_logic_vector(1 downto 0);
32       bitac           : out std_logic_vector(79 downto 0)
33     );
34 end entity Completo;
35
36 architecture pt of Completo is
37
38    component firewall is
39       port(
40          clk                :in  std_logic;
41          rcv_H_in           :in  std_logic;
42          rcv_H_out          :in  std_logic;
43          entradaHeader_in   :in  std_logic_vector(87 downto 0);
44          entradaHeader_out  :in  std_logic_vector(87 downto 0);
45          programacion       :in  std_logic;
46          regla_in           :in  std_logic_vector(92 downto 0);
47          regla_out          :in  std_logic_vector (92 downto 0);
48          dir_in             :in  std_logic_vector(3 downto 0);
49          dir_out            :in  std_logic_vector(3 downto 0);
50          readyIn            :out std_logic_vector(1 downto 0);
51          readyOut           :out std_logic_vector(1 downto 0)
52    );
53 end component firewall;
54
55    component bitacora is
56    port (
```

```
57        clk       :in std_logic;
58        reset     :in std_logic;
59        wr        :in std_logic;
60        rd        :in std_logic;
61        datosIn   :in std_logic_vector(63 downto 0);
62        datosOut  :out std_logic_vector(79 downto 0)
63    );
64    end component bitacora;
65
66 signal H_in, H_out : std_logic_vector(87 downto 0);
67 signal R_in, R_out : std_logic_vector(92 downto 0);
68 signal D_in, D_out : std_logic_vector(3 downto 0);
69
70 signal prog, wr_b  : std_logic;
71 signal rdyIn, rdyOut : std_logic_vector(1 downto 0);
72 signal int : std_logic_vector(4 downto 0);
73
74 begin
75
76    int <= interrupt;
77
78    process(clk, int)
79    begin
80       --se agrupan los datos de entrada para formar las reglas completas
81       if clk'event and clk='1' then
82          if int(3) = '1' then
83             H_in(87 downto 64)  <= data_H_in_1(23 downto 0);
84             H_in(63 downto 32)  <= data_H_in_2;
85             H_in(31 downto 0)   <= data_H_in_3;
86          end if;
87          if int(2) = '1' then
88             H_out(87 downto 64) <= data_H_out_1(23 downto 0);
89             H_out(63 downto 32) <= data_H_out_2;
90             H_out(31 downto 0)  <= data_H_out_3;
91          end if;
92          if int(1) = '1' then
93             R_in(92)            <= Reg_1(28);
94             R_in(91)            <= Reg_1(27);
95             R_in(90 downto 75)  <= Reg_1(23 downto 8);
96             R_in(74)            <= Reg_1(26);
97             R_in(73 downto 66)  <= Reg_1(7 downto 0);
98             R_in(65)            <= Reg_1(25);
99             R_in(64 downto 33)  <= Reg_2;
100            R_in(32)            <= Reg_1(24);
101            R_in(31 downto 0)   <= Reg_3;
102            D_in                <= dir;
103         end if;
104         if int(0) = '1' then
105            R_out(92)           <= Reg_1(28);
106            R_out(91)           <= Reg_1(27);
107            R_out(90 downto 75) <= Reg_1(23 downto 8);
108            R_out(74)           <= Reg_1(26);
109            R_out(73 downto 66) <= Reg_1(7 downto 0);
110            R_out(65)           <= Reg_1(25);
111            R_out(64 downto 33) <= Reg_2;
112            R_out(32)           <= Reg_1(24);
113            R_out(31 downto 0)  <= Reg_3;
114            D_out               <= dir;
115         end if;
116      end if;
117   end process;
118
119   prog <= '1' when int(1) = '1' or int(0) = '1' else '0';
120   -- se instancian los componentes
121   U0:firewall
122   port map(
```

```
123         clk                 => clk,
124         rcv_H_in            => int(3),
125         rcv_H_out           => int(2),
126         entradaHeader_in    => H_in,
127         entradaHeader_out   => H_out,
128         programacion        => prog,
129         regla_in            => R_in,
130         regla_out           => R_out,
131         dir_in              => D_in,
132         dir_out             => D_out,
133         readyIn             => rdyIn,
134         readyOut            => rdyOut
135     );
136
137     wr_b <= '1' when rdyIn = "11" or rdyOut = "11" else '0';
138
139     U1:bitacora
140     port map(
141         clk       => clk,
142         reset     => reset,
143         wr        => wr_b,
144         rd        => int(4),
145         datosIn => H_in(63 downto 0),
146         datosOut => bitac
147     );
148
149     rdy_in <= rdyIn;
150     rdy_out<= rdyOut;
151
152 end architecture pt;
```

Archivo: `bitacora.vhd`

```
1
2 bitacora.vhd
3 library IEEE;
4 use IEEE.STD_LOGIC_1164.ALL;
5 use IEEE.STD_LOGIC_ARITH.ALL;
6 use IEEE.STD_LOGIC_UNSIGNED.ALL;
7 use work.funciones.all;
8
9 entity bitacora is
10 port (
11     clk          :in std_logic; -- reloj
12     reset                :in std_logic;-- reset
13     wr                   :in std_logic;-- escribir
14     rd                   :in std_logic;-- leer
15     datosIn      :in std_logic_vector(63 downto 0);-- entrada de datos
16     datosOut     :out std_logic_vector(79 downto 0)-- salida de datos
17 );
18 end entity bitacora;
19
20 architecture pt of bitacora is
21 --memoria auxiliar
22 type RAM is array (0 to 31) of std_logic_vector(79 downto 0);
23 signal mem : RAM := (others => (others => '1'));
24
25 signal index,i   : natural range 0 to 31;
26 signal todo              : std_logic_vector(79 downto 0);
27
```

```
28 begin
29
30    todo <= datosIn & x"0001";
31
32    process (wr,reset)
33    begin
34       if reset = '1' then -- reinicia los valores
35          mem <= (others => (others => '1'));
36          index <= 0;
37       end if;
38       if wr = '1' then -- se almacenan la informacion en la memoria
39          if index = 0 then
40             mem(index) <= todo;
41             index <= index + 1;
42          else
43             if index < 31 then
44                mem(index) <= todo;
45                index <= index + 1;
46             end if;
47          end if;
48       end if;
49    end process;
50
51    process(clk)
52    begin
53       if clk'event and clk = '1' then
54          if rd = '1' then
55             if i < index then-- se extraen los datos de la bitacora
56                datosOut <= mem(i);
57                i <= i + 1;
58             else
59                datosOut <= (others => '1');
60                i <= 0;
61             end if;
62          end if;
63       end if;
64         end process;
65
66 end pt;
```

Archivo: `firewall.vhd`

```
1
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_arith.all;
5 use ieee.std_logic_unsigned.all;
6 use work.funciones.all;
7
8 entity firewall is
9 port(
10    clk               :in  std_logic;-- reloj
11    -- senales de habilitacion
12    rcv_H_in          :in  std_logic;
13    rcv_H_out         :in  std_logic;
14    --entradas de datos
15    entradaHeader_in  :in  std_logic_vector(87 downto 0);
16    entradaHeader_out :in  std_logic_vector(87 downto 0);
17    -- programacion de la memoria
18    programacion      :in  std_logic;
```

```
19    --entradas de las reglas
20    regla_in          :in  std_logic_vector(92 downto 0);
21    regla_out         :in  std_logic_vector (92 downto 0);
22    --entradas de las direcciones
23    dir_in            :in  std_logic_vector(3 downto 0);
24    dir_out           :in  std_logic_vector(3 downto 0);
25    --salida de control
26    readyIn           :out std_logic_vector(1 downto 0);
27    readyOut          :out std_logic_vector(1 downto 0)
28 );
29 end entity firewall;
30
31 architecture pt of firewall is
32
33    component filtro is
34    port(
35       clk            :in  std_logic;
36       rcv_H_in       :in  std_logic;
37       rcv_H_out      :in  std_logic;
38       datos_in       :in  std_logic_vector(87 downto 0);
39       -- datos del header
40       datos_out      :in  std_logic_vector(87 downto 0);
41       -- datos de las reglas
42       reglas_in      :in  std_logic_vector(92 downto 0);
43       reglas_out     :in  std_logic_vector(92 downto 0);
44       -- direccionde memoria a controlar
45       dir_mem_in     :out std_logic_vector(3 downto 0);
46       dir_mem_out    :out std_logic_vector(3 downto 0);
47       dir_max_mem_in :in  std_logic_vector(3 downto 0);
48       dir_max_mem_out:in  std_logic_vector(3 downto 0);
49       control_in     :out std_logic_vector(1 downto 0);
50       control_out    :out std_logic_vector(1 downto 0)
51    );
52    end component filtro;
53
54    component tabla is
55    port (
56       clk         :in  std_logic;
57       regla_in    :in  std_logic_vector(92 downto 0);
58       prog        :in  std_logic;
59       dir_memoria :in  std_logic_vector(3 downto 0);
60       dir_mprog   :in  std_logic_vector(3 downto 0);
61       dir_max_mem :out std_logic_vector(3 downto 0);
62       regla_sal   :out std_logic_vector(92 downto 0)
63    );
64 end component tabla;
65
66 signal reloj : std_logic;
67 signal dirmem_in,dirmem_out:std_logic_vector(3 downto 0);
68 signal regl_in,regl_out:std_logic_vector(92 downto 0);
69 signal headers_in,headers_out:std_logic_vector(87 downto 0);
70 signal mem_max_in,mem_max_out:std_logic_vector(3 downto 0);
71
72 begin
73
74    reloj<=clk;
75    headers_in<=entradaHeader_in;
76    headers_out<=entradaHeader_out;
77    --instanciado de componentes
78
79    elFiltro:filtro
80    port map(
81       clk            => reloj,
82       rcv_H_in       => rcv_H_in,
83       rcv_H_out      => rcv_H_out,
84       datos_in       => headers_in,
```

```
85        datos_out       => headers_out,
86        reglas_in       => regl_in,
87        reglas_out      => regl_out,
88        dir_mem_in      => dirmem_in,
89        dir_mem_out     => dirmem_out,
90        dir_max_mem_in  => mem_max_in,
91        dir_max_mem_out=> mem_max_out,
92        control_in      => readyIn,
93        control_out     => readyOut
94    );
95
96    tablaIn:tabla
97    port map(
98        clk          => reloj,
99        regla_in     => regla_in(92 downto 0),
100       prog         => programacion,
101       dir_memoria  => dirmem_in,
102       dir_mprog    => dir_in,
103       dir_max_mem  => mem_max_in,
104       regla_sal    => regl_in
105   );
106
107   tablaOut:tabla
108   port map(
109       clk          => reloj,
110       regla_in     => regla_out(92 downto 0),
111       prog         => programacion,
112       dir_memoria  => dirmem_out,
113       dir_mprog    => dir_out,
114       dir_max_mem  => mem_max_out,
115       regla_sal    => regl_out
116         );
117
118 end architecture pt;
```

Archivo: `tabla.vhd`

```
1
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_arith.all;
5 use ieee.std_logic_unsigned.all;
6 use work.funciones.all;
7
8 entity tabla is
9 port(
10    -- reloj
11    clk         :in  std_logic;
12    -- entrada de reglas
13    regla_in    :in  std_logic_vector(92 downto 0);
14    --progrramacion de la memoria
15    prog        :in  std_logic;
16    --direcciones para el manejo de la memoria
17    dir_memoria :in  std_logic_vector(3 downto 0);
18    dir_mprog   :in  std_logic_vector(3 downto 0);
19    dir_max_mem :out std_logic_vector(3 downto 0);
20    -- salida de reglas
21    regla_sal   :out std_logic_vector(92 downto 0)
22 );
23 end entity tabla;
24
25 architecture pt of tabla is
26
27 type unaTabla is array (0 to 15) of std_logic_vector(92 downto 0);
28 signal reglas : unaTabla :=( -- valores por defecto
29    0  => "1" & x"00000000000000000000000",
30    1  => "1" & x"00000000000000000000000",
31    2  => "1" & x"00000000000000000000000",
32    3  => "1" & x"00000000000000000000000",
33    4  => "1" & x"00000000000000000000000",
34    5  => "1" & x"00000000000000000000000",
35    6  => "1" & x"00000000000000000000000",
36    7  => "1" & x"00000000000000000000000",
37    8  => "1" & x"00000000000000000000000",
38    9  => "1" & x"00000000000000000000000",
39    10 => "1" & x"00000000000000000000000",
40    11 => "1" & x"00000000000000000000000",
41    12 => "1" & x"00000000000000000000000",
42    13 => "1" & x"00000000000000000000000",
43    14 => "1" & x"00000000000000000000000",
44    15 => "1" & x"00000000000000000000000");
45 signal max_mem, dir_m :std_logic_vector(3 downto 0);
46
47 begin
48    --obtiene la direccion maxima de memoria
49    max_mem<=dir_mprog;
50    dir_m <= dir_memoria;
51
52    process(clk, prog)
53    variable index: natural;
54    begin
55       if(clk'event and clk='1') then
56          if(prog='1') then-- carga a memoria
57             reglas(bit_vector2integer(dir_mprog))<=regla_in;
58             dir_max_mem<=max_mem;
59          end if;
```

```
60          if(prog='0') then --extrae de memoria
61             regla_sal<=reglas(bit_vector2integer(dir_m));
62          end if;
63       end if;
64    end process;
65
66 end architecture pt;
```

Archivo: `filtro.vhd`

```
 1
 2 library ieee;
 3 use ieee.std_logic_1164.all;
 4 use ieee.std_logic_arith.all;
 5 use ieee.std_logic_unsigned.all;
 6 use work.funciones.all;
 7
 8 entity filtro is
 9 port(
10    clk             :in  std_logic;--reloj
11    --entradas para habilitacion
12    rcv_H_in        :in  std_logic;
13    rcv_H_out       :in  std_logic;
14    --entradas de datos
15    datos_in        :in  std_logic_vector(87 downto 0);
16    datos_out       :in  std_logic_vector(87 downto 0);
17    -- entrada de reglas
18    reglas_in       :in  std_logic_vector(92 downto 0);
19    reglas_out      :in  std_logic_vector(92 downto 0);
20    --direcciones de memoria a llamar
21    dir_mem_in      :out std_logic_vector(3 downto 0);
22    dir_mem_out     :out std_logic_vector(3 downto 0);
23    --direccion de memoria maxima
24    dir_max_mem_in :in  std_logic_vector(3 downto 0);
25    dir_max_mem_out:in  std_logic_vector(3 downto 0);
26    --senales de control
27    control_in      :out std_logic_vector(1 downto 0);
28    control_out     :out std_logic_vector(1 downto 0)
29 );
30 end entity filtro;
31
32 architecture pt of filtro is
33
34    component maquina_edos is
35    port(
36       clk          :in  std_logic;
37       datos        :in  std_logic_vector(87 downto 0);
38       regla        :in  std_logic_vector (92 downto 0);
39       en           :in  std_logic;
40       dir_mem_max :in  std_logic_vector(3 downto 0);
41       dir_mem      :out std_logic_vector(3 downto 0);
42       control      :out std_logic_vector(1 downto 0)
43    );
44    end component maquina_edos;
45
46 signal reloj : std_logic;
47 signal max_mem_in,max_mem_out:std_logic_vector(3 downto 0);
48
49 begin
50
```

```
51    reloj<=clk;
52    max_mem_in<=dir_max_mem_in;
53    max_mem_out<=dir_max_mem_out;
54    --instanciado de componentes
55
56    maquinaIn:maquina_edos
57    port map(
58        clk          => reloj,
59        datos        => datos_in,
60        regla        => reglas_in,
61        en           => rcv_H_in,
62        dir_mem_max  => max_mem_in,
63        dir_mem      => dir_mem_in,
64        control      => control_in
65    );
66
67    maquinaOut:maquina_edos
68    port map(
69        clk          => reloj,
70        datos        => datos_out,
71        regla        => reglas_out,
72        en           => rcv_H_out,
73        dir_mem_max  => max_mem_out,
74        dir_mem      => dir_mem_out,
75        control      => control_out
76    );
77
78 end architecture pt;
```

Archivo: `maquina_edos.vhd`

```
1
2 library ieee;
3
4 use ieee.std_logic_1164.all;
5 use ieee.std_logic_arith.all;
6 use ieee.std_logic_unsigned.all;
7 use work.funciones.all;
8
9 entity maquina_edos is
10 port(
11    --reloj
12    clk          : in  std_logic;
13    --datos de los encabezados
14    datos        : in  std_logic_vector(87 downto 0);
15    --reglas para el firewall
16    regla        : in  std_logic_vector (92 downto 0);
17    --habilitacion
18    en           : in  std_logic;
19    --maxima direccion de memoria
20    dir_mem_max  : in  std_logic_vector(3 downto 0);
21    --direccion de memoria a carga
22    dir_mem      : out std_logic_vector(3 downto 0);
23    --control
24    control      : out std_logic_vector(1 downto 0)
25 );
26 end entity maquina_edos;
27
28 architecture pt of maquina_edos is
29
```

```
30 signal estado:integer range 0 to 6;
31 signal edo:integer range 0 to 1;
32 signal mem_max:std_logic_vector(3 downto 0);
33 signal memoria:std_logic_vector(3 downto 0):="0000";
34 signal ctrl: std_logic_vector(1 downto 0):= "00";
35 signal tmp1:std_logic_vector(92 downto 0);
36 signal tmp2:std_logic_vector(87 downto 0);
37 signal resultado:std_logic_vector(87 downto 0);
38 signal cmp:std_logic:='0';
39
40 begin
41
42    control <= ctrl;
43    mem_max<=dir_mem_max;
44
45    process(clk,en)
46    begin
47       if(clk'event and clk='1')then
48          if en='0'then   -- reinicio
49             estado<=0;
50          end if;
51          if en='1' then
52             case estado is
53             when 0=>
54                -- limpiar direcion de memoria
55                memoria<="0000";
56                estado<=1;
57             when 1=>
58                edo <= 0;
59                -- solicita una nueva direccion
60                dir_mem<=memoria;
61                estado<=2;
62             when 2=>
63                ctrl<="00";
64                tmp1<=regla;
65                -- carga reglas y datos a registros internos
66                tmp2<=datos;
67                estado<=3;
68             when 3=>
69                if(tmp1(32)='0')then   --ip destino
70                   --si el campo no se toma en cuenta se cargan con '1'
71                   --para no afectar el resultado final
72                   resultado(31 downto 0)<="11111111111111111111111111111111";
73                else
74                   --si el campo se va a tomar en cuenta, se hace
75                   --una comparacion entre la regla y el contenido
76                   -- de los datos se realiza lo mismo para todos los campos
77                   resultado(31 downto 0)<=tmp1(31 downto 0)xnor tmp2(31 downto 0);
78                end if;
79                if (tmp1(65)='0') then--ip origen
80                   resultado(63 downto 32)<="11111111111111111111111111111111";
81                else
82                   resultado(63 downto 32)<=tmp1(64 downto 33)xnor tmp2(63 downto 32);
83                end if;
84                if(tmp1(74)='0')then -- protocolo
85                   resultado(71 downto 64)<="11111111";
86                else
87                   resultado(71 downto 64)<=tmp1(73 downto 66)xnor tmp2(71 downto 64);
88                end if;
89                if(tmp1(91)='0') then --puerto
90                   resultado(87 downto 72)<="1111111111111111";
91                else
92                   resultado(87 downto 72)<=tmp1(90 downto 75)xnor tmp2(87 downto 72);
93                end if;
94                estado<=4;
95             when 4=>
```

```
96              -- comprueba si el encabezado coincide con la regla, en caso
97              --de que todos los bits sean '1' y si hay mas reglas
98              cmp<=operacionand(resultado);
99              edo <= comprueba(memoria,mem_max);
100             estado<=5;
101          when 5=>
102             -- prepara la nueva direccion de memoria
103             memoria<=memoria+1;
104             --acepta o rechaza el paquete segun lo indeque la regla, y
105             --segun coincida o no el paquete con la regla
106             if(tmp1(92)='0' and cmp='1') or (tmp1(92)='1' and cmp='0') then
107                ctrl<="11"; -- rechazo
108                estado <= 0;
109             else
110                if(edo = 0) then
111                   ctrl<="10"; -- acepta
112                   estado <= 0;
113                else
114                   estado<= 1;
115                end if;
116             end if;
117          when others =>
118             null;
119          end case;
120       end if;
121    end if;
122 end process;
```

Archivo: `funciones.vhd`

```
1
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_arith.all;
5 use ieee.std_logic_unsigned.all;
6
7 package funciones is
8
9    function bit_vector2integer(A:std_logic_vector(3 downto 0)) return integer;
10   function integer2bit_vector(B:integer range 0 to 15)
11      return std_logic_vector;
12   function operacionand(C:std_logic_vector(87 downto 0)) return std_logic;
13   function comprueba(D:std_logic_vector(3 downto 0);
14      M:std_logic_vector(3 downto 0)) return integer;
15
16 end funciones;
17
18 package body funciones is
19
20   function bit_vector2integer(A:std_logic_vector(3 downto 0)) return integer is
21      variable var:std_logic_vector(3 downto 0):=A;
22      variable ent:integer range 0 to 15:=0;
23      begin-- conversion de vector a entero
24         for i in 0 to 3 loop
25            if (var(0)='1') then
26               ent:=ent+(2**i);
27            end if;
28            var:='0'&var(3 downto 1);
29         end loop;
30      return ent;
```

```
31    end  bit_vector2integer;
32
33    function integer2bit_vector(B:integer range 0 to 15) return std_logic_vector is
34       variable v1:std_logic_vector (3 downto 0):="0000";
35       variable op:integer range 0 to 15:=B;
36       begin-- conversion de entero a vector
37          for i in 0 to 3 loop
38             if(op MOD 2)=0 then
39                v1(i):='0';
40             else
41                v1(i):='1';
42             end if;
43             op:=op/2;
44          end loop;
45       return v1;
46    end integer2bit_vector;
47
48    function operacionand(C:std_logic_vector(87 downto 0))return std_logic is
49       variable var:std_logic:='1';
50       begin-- se realiza una AND entre todos los bits
51          for i in 0 to 87 loop
52             var:=var and c(i);
53          end loop;
54       return var;
55    end operacionand;
56
57    function comprueba(D:std_logic_vector(3 downto 0);M:
58       std_logic_vector(3 downto 0))return integer is
59       variable edo:integer range 0 to 1;
60       begin-- compara ambas direcciones de memoria
61          if(D=M)then
62             edo:=0;
63          else
64             edo:=1;
65          end if;
66       return edo;
67    end function comprueba;
68
69 end funciones;
```

Apéndice C

Código del software que interactua con el firewall

Archivo: `main.c`

```
1#include "redireccion.h"
2
3void* redireccion_thread(void * arg)
4{
5    pthread_t hilo_recv_eth, hilo_vacia_cola, hilo_button;
6    int r;
7
8    config_emac ();
9    config_sl();
10
11   FIREWALL_BITACORA_mReset(XPAR_FIREWALL_BITACORA_0_BASEADDR);
12
13   /* se lanza el hilo para ver el estado de button*/
14   r = pthread_create (&hilo_button, NULL, button, NULL);
15        if(r != 0)print("Error al crear hilo recv_eth\r\n");
16
17   /* se lanza el hilo de recepcion eth*/
18   r = pthread_create (&hilo_recv_eth, NULL, recv_eth_thread, NULL);
19   if(r != 0)print("Error al crear hilo recv_eth\r\n");
20
21   /* se lanza el hilo que vacia la cola*/
22   r = pthread_create (&hilo_vacia_cola, NULL, ExtraerPack, NULL);
23   if(r != 0)print("Error al crear el hilo\r\n");
24
25   // NUNCA LLEGA
26   pthread_join (hilo_button, NULL); // espera a el hilo de recepcion eth
27   pthread_join (hilo_recv_eth, NULL); // espera a el hilo de recepcion eth
28   pthread_join (hilo_vacia_cola, NULL); // espera a el hilo que vacia la cola
29
30   exit(0);
31}
32
33int main()
34{
35   xilkernel_main();
36}
```

Archivo: `redireccion.h`

```
1#include "xmk.h"
2#include "pthread.h"
3#include "stdio.h"
4#include "xparameters.h"
5#include "xemac_l.h"
6#include "xutil.h"
7#include "stdlib.h"
8#include "firewall_bitacora.h"
9
10#include "eth.h"
11#include "sl.h"
12
13 void concatena (Xuint8 *buff, Xuint8 *res, int len);
14 void * button (void * arg);
15 void * recv_eth_thread(void * arg);
16 void * ExtraerPack(void * arg);
17 void procesa_paquete (Xuint8 * buff, int len);
```

Archivo: `redireccion.c`

```
1#include "redireccion.h"
2#include "Cola_Packete.h"
3#include "led_button.h"
4
5 struct Cola cola;
6 int b = 0,cont =0;
7
8 /*Concatena los datos obtenidos de la cabecera IP*/
9 void concatena (Xuint8 *buff, Xuint8 *res, int len)
10 {
11    int i;
12    for(i=1;i<=len;i++)
13    *(res++)=*buff++;
14 }
15
16 /* Hilo ocupado para permitir la comunicacion serial*/
17 void * button (void * arg)
18 {
19    Xuint32 DataRead;
20    Xuint8 *RcvBuffer, reg1[4],reg2[4],reg3[4];
21    Xuint32 i=0,*res,value;
22    while(1)
23    {
24       //revisa el estado de los botones
25       Input(XPAR_PUSHBUTTONS_5BIT_DEVICE_ID, &DataRead);
26       for (i = 0; i < LED_DELAY*4; i++); // retardo
27
28       // si hubo algun cambio ene el estado de los botones
29       if(DataRead != 0x1F)
30       {
31          b = 1;
32          bzero(RcvBuffer, sizeof(RcvBuffer));
33          serial_recive((int)stdin, RcvBuffer, 14);
34
```

```
35          if (RcvBuffer[0] == 0x02) // si peticion de bitacora
36          {
37              // HABILITA bitacora
38              FIREWALL_BITACORA_mWriteSlaveReg7(XPAR_FIREWALL_BITACORA_0_BASEADDR,
39              0x00000010);
40              // DESHABILITA bitacora
41              FIREWALL_BITACORA_mWriteSlaveReg7(XPAR_FIREWALL_BITACORA_0_BASEADDR,
42              0x00000000);
43              value=0;
44              do
45              {
46                  value = FIREWALL_BITACORA_mReadSlaveReg9(XPAR_FIREWALL_BITACORA_0_BASEADDR);
47              }
48              while (value == 0);
49
50              //envia direccion origen
51              serial_send((int)stdin, (Xuint8*)value, 4);
52              value=FIREWALL_BITACORA_mReadSlaveReg9(XPAR_FIREWALL_BITACORA_0_BASEADDR);
53              serial_send((int)stdin, (Xuint8*)value, 4); //envia direccion destino
54              value=FIREWALL_BITACORA_mReadSlaveReg10(XPAR_FIREWALL_BITACORA_0_BASEADDR);
55              //envia el numedo de ocurrencias
56              serial_send((int)stdin, (Xuint8*)value, 4);
57              value=FIREWALL_BITACORA_mReadSlaveReg11(XPAR_FIREWALL_BITACORA_0_BASEADDR);
58
59              if(value=0xFFFFFFFF) //si la bitacora esta vacia
60                  b=0;
61                  continue;
62          }
63
64          while(RcvBuffer[0] == 0x01)
65          {
66              reg1[0]=RcvBuffer[2]; reg1[1]=RcvBuffer[3]; reg1[2]=RcvBuffer[4];
67              reg1[3]=RcvBuffer[5]; reg2[0]=RcvBuffer[6]; reg2[1]=RcvBuffer[7];
68              reg2[2]=RcvBuffer[8]; reg2[3]=RcvBuffer[9]; reg3[0]=RcvBuffer[10];
69              reg3[1]=RcvBuffer[11];reg3[2]=RcvBuffer[12];reg3[3]=RcvBuffer[13];
70
71              // DIRECCION
72              FIREWALL_BITACORA_mWriteSlaveReg6(XPAR_FIREWALL_BITACORA_0_BASEADDR,i);
73
74              // REGLA
75              concatena(reg1,(Xuint8*)&res,4);
76              FIREWALL_BITACORA_mWriteSlaveReg3(XPAR_FIREWALL_BITACORA_0_BASEADDR, res);
77              concatena(reg2,(Xuint8*)&res,4);
78              FIREWALL_BITACORA_mWriteSlaveReg4(XPAR_FIREWALL_BITACORA_0_BASEADDR, res);
79              concatena(reg3,(Xuint8*)&res,4);
80              FIREWALL_BITACORA_mWriteSlaveReg5(XPAR_FIREWALL_BITACORA_0_BASEADDR, res);
81
82              // HABILITA ESCRIBIR REGLAS
83              FIREWALL_BITACORA_mWriteSlaveReg7(XPAR_FIREWALL_BITACORA_0_BASEADDR,
84              0x00000002);
85              // DESHABILITA ESCRIBIR REGLAS
86              FIREWALL_BITACORA_mWriteSlaveReg7(XPAR_FIREWALL_BITACORA_0_BASEADDR,
87              0x00000000);
88
89              i++;
90              bzero(RcvBuffer, sizeof(RcvBuffer));
91              serial_recive((int)stdin, RcvBuffer, 14);
92          }
93          b=0;
94          i=0;
95          print("Reglas programadas\r\n");
96      }
97  }
98 }
99
100 void * ExtraerPack(void * arg)
```

```
101 {
102    struct Nodo *aux;
103    while(1)
104    {
105       if(cola.pri != NULL)
106       {
107          if (b == 0)
108          { //si no hay comunicacion serial
109             procesa_paquete (cola.pri->dato.buff, cola.pri->dato.len);
110             aux = cola.pri;
111             cola.pri=(*cola.pri).proximo;
112             free(aux);
113          }
114       }
115       else
116       {
117          cola.ult = NULL;
118       }
119    }
120 }
121
122 void * recv_eth_thread(void * arg)
123 {
124    struct Paquete x;
125
126    while(1)
127    {
128       x.len=0;
129       bzero(x.buff,sizeof(x.buff));
130       x.len = XEmac_RecvFrame(EMAC_BASEADDR, x.buff);
131       if(x.len <= 0)continue;
132       cola.ult=CreaCola(cola.ult,x);  //se encola
133       // Si es la 1 pasada pongo en pri el valor del primer nodo
134       if(cola.pri==NULL) cola.pri=cola.ult;
135    }
136 }
137
138
139 void procesa_paquete (Xuint8 * buff, int len)
140 {
141    int r,i;
142    Xuint8 rev1[4], rev2[4], rev3[4];
143    Xuint32 value, *res;
144
145    if(buff[12] == 0x08 && buff[13] == 0x00 ) // si es un IP
146    {
147       rev1[0]=0x00;      rev1[1]=buff[36]; rev1[2]=buff[37]; rev1[3]=buff[23];
148       rev2[0]=buff[26]; rev2[1]=buff[27]; rev2[2]=buff[28]; rev2[3]=buff[29];
149       rev3[0]=buff[30]; rev3[1]=buff[31]; rev3[2]=buff[32]; rev3[3]=buff[33];
150
151       xil_printf("\r\n\r\nPAQUETE NUMERO %d\r\n",cont++);
152       xil_printf("\r\nDireccion origen = %d.%d.%d.%d\r\n",rev2[0],
153       rev2[1],rev2[2],rev2[3]);
154       xil_printf("Direccion destino = %d.%d.%d.%d\r\n",rev3[0],rev3[1],
155       rev3[2],rev3[3]);
156       xil_printf("Protocolo de cabecera IP = %X\r\n",rev1[3]);
157       xil_printf("Puerto destino = %X %X\r\n",rev1[1],rev1[2]);
158       PrintPacketInHex(buff, len);
159
160       concatena(rev1,(Xuint8*)&res,4);
161       FIREWALL_BITACORA_mWriteSlaveReg0(XPAR_FIREWALL_BITACORA_0_BASEADDR,res);
162       concatena(rev2,(Xuint8*)&res,4);
163       FIREWALL_BITACORA_mWriteSlaveReg1(XPAR_FIREWALL_BITACORA_0_BASEADDR, res);
164       concatena(rev3,(Xuint8*)&res,4);
165       FIREWALL_BITACORA_mWriteSlaveReg2(XPAR_FIREWALL_BITACORA_0_BASEADDR, res);
166
```

```
167         // HABILITA ESCRIBIR HEADER_IN
168         FIREWALL_BITACORA_mWriteSlaveReg7(XPAR_FIREWALL_BITACORA_0_BASEADDR,
169         0x00000008);
170         value=0;
171         do
172         {
173             value = FIREWALL_BITACORA_mReadSlaveReg8(XPAR_FIREWALL_BITACORA_0_BASEADDR);
174         }
175         while (value == 0);
176
177         if (value == 3)print("Paquete Denegado!!!\r\n\r\n");
178         if (value == 2)print("Paquete Aceptado!!!\r\n\r\n");
179
180         // DESHABILITA ESCRIBIR HEADER_IN
181         FIREWALL_BITACORA_mWriteSlaveReg7(XPAR_FIREWALL_BITACORA_0_BASEADDR,
182         0x00000000);
183         Output(XPAR_LEDS_4BIT_DEVICE_ID,4,0x0);
184
185     }
186     if(buff[12] == 0x08 && buff[13] == 0x06 )
187     {
188         //mandar a la otra interfaz
189         printf("\r\nPAQUETE ARP\r\n");
190         PrintPacketInHex(buff, len);
191     }
192 }
```

Archivo: `eth.h`

```
1#include "xemac_l.h"
2#include "xparameters.h"
3
4#define EMAC_BASEADDR    XPAR_ETHERNET_MAC_BASEADDR
5#define MAX_FRAME_SIZE   1518      /* Max size of the Frame bits*/
6
7
8
9 void config_emac ();
10 void PrintPacketInHex(char *packet, int len);
```

Archivo: `eth.h`

```
1#include "xemac_l.h"
2#include "xparameters.h"
3
4#define EMAC_BASEADDR    XPAR_ETHERNET_MAC_BASEADDR
5#define MAX_FRAME_SIZE   1518      /* Max size of the Frame bits*/
6
7 void config_emac ();
8 void PrintPacketInHex(char *packet, int len);
```

Archivo: `eth.c`

```
1#include "eth.h"
2
3 Xuint8 LocalAddress[6] =
4 {
5    0x06, 0x05, 0x04, 0x03, 0x02, 0x01
6 };
7
8
9 void config_emac ()
10 {
11    Xuint32 Control;
12
13    Control = XEmac_mReadReg(EMAC_BASEADDR, XEM_ECR_OFFSET);
14    XEmac_mWriteReg(EMAC_BASEADDR, XEM_ECR_OFFSET,
15                    Control | XEM_ECR_UNICAST_ENABLE_MASK
16                            | XEM_ECR_MULTI_ENABLE_MASK
17                            | XEM_ECR_BROAD_ENABLE_MASK
18                            | XEM_ECR_PROMISC_ENABLE_MASK);
19
20    XEmac_mSetMacAddress(EMAC_BASEADDR, LocalAddress);
21
22    XEmac_mEnable(EMAC_BASEADDR);
23 }
24
25 void PrintPacketInHex(char *packet, int len)
26 {
27    unsigned char *p = packet;
28
29    while(len--)
30    {
31       if (len % 30 == 0)
32       print("\r\n");
33          xil_printf("%0.2x ", *p);
34          p++;
35    }
36 }
```

Archivo: `sl.h`

```
1#include "xutil.h"
2#include "uartlite.h"
3#define UART_BASEADDR              XPAR_RS232_UART_1_BASEADDR
4#define UARTLITE_DEVICE_ID   XPAR_RS232_UART_1_DEVICE_ID
5
6
7 XUartLite UartLite; /* Instance of the UartLite Device */
8
9 void config_sl();
10 int  serial_send(XUartLite* UartLite, Xuint8 *data, int size);
11 int  serial_recive(XUartLite* UartLite, Xuint8 *data, int size);
```

Archivo: `sl.c`

```
1#include "xstatus.h"
2#include "stdlib.h"
3#include "sl.h"
```

```
4
5
6 void config_sl()
7 {
8    XStatus Status;
9    /*Inicializa el UART para su uso*/
10   Status = XUartLite_Initialize(&UartLite, UARTLITE_DEVICE_ID);
11   if (Status != XST_SUCCESS)
12   {
13      exit(0);
14   }
15 }
16
17 int serial_recive(XUartLite* UartLite, Xuint8 *data, int size)
18 {
19   int count=0, n;
20   do
21   {
22      if ((n=XUartLite_Recv(UartLite, &data[count], size-count)) == -1)
23      {
24         printf("Error al leer los datos: ");
25         return -1;
26      }
27      count+=n;
28   }
29   while (count<size);
30
31   return count;
32 }
33
34 int serial_send(XUartLite* UartLite, Xuint8 *data, int size)
35 {
36   int count=0, n;
37   do
38   {
39      if ((n=XUartLite_Recv(UartLite, &data[count], size-count)) == -1)
40      {
41         printf("Error al escribir los datos: ");
42         return -1;
43      }
44      count+=n;
45   }
46   while (count<size);
47
48   return count;
49 }
```

Archivo: `Cola_Packete.h`

```
1#include "xutil.h"
2#include "eth.h"
3
4 struct Paquete
5 {
6   Xuint8 buff[MAX_FRAME_SIZE];
7   int len;
8 };
9 struct Nodo
10 {
11   struct Paquete dato;
12   struct Nodo *proximo;
```

```
13 };
14
15 struct Cola
16 {
17    struct Nodo *pri;
18    struct Nodo *ult;
19 };
20
21
22 struct Nodo *NuevoNodo();
23 struct Nodo *CreaCola(struct Nodo *ult, struct Paquete x);
24 int colavacia(struct Nodo *pri);
```

Archivo: `Cola_Packete.c`

```
1#include <stdio.h>
2#include <stdlib.h>
3#include <string.h>
4#include "Cola_Packete.h"
5
6
7 struct Nodo *NuevoNodo()
8 {
9    struct Nodo *p;
10
11    p=(struct Nodo *)malloc(sizeof(struct Nodo));
12    if(p==NULL)
13    {
14       printf("Memoria RAM Llena\r\n");
15       exit(0);
16    }
17    return p;
18 }
19
20 struct Nodo *CreaCola(struct Nodo *ult, struct Paquete x)
21 {
22    struct Nodo *p;
23    p=NuevoNodo();
24    (*p).dato=x;
25    (*p).proximo=NULL;
26
27    if(ult!=NULL) (*ult).proximo=p;
28    // Si hay nodo anterior en prox pongo la direccion del nodo actual
29    return p;
30 }
31
32 int colavacia(struct Nodo *pri)
33 {
34    if(pri==NULL) return 1;
35    else return 0;
36 }
```

Archivo: `led_button.h`

```
1#include "xparameters.h"
2#include "xgpio.h"
```

```
 3#include "stdio.h"
 4#include "xbasic_types.h"
 5#include "xstatus.h"
 6
 7#define LED_DELAY     1000000
 8
 9#define LED_CHANNEL 1
10
11 XGpio GpioOutput;
12 XGpio GpioInput;
13
14 XStatus Output(Xuint16 DeviceId, Xuint32 GpioWidth, Xuint8 value);
15 XStatus Input(Xuint16 DeviceId, Xuint32 *DataRead)
```

Archivo: `led_button.c`

```
 1#include "led_button.h"
 2
 3
 4 XStatus Output(Xuint16 DeviceId, Xuint32 GpioWidth, Xuint8 value)
 5 {
 6    XStatus Status;
 7    volatile int Delay;
 8
 9    Status = XGpio_Initialize(&GpioOutput, DeviceId);
10    if (Status != XST_SUCCESS)
11    {
12       return XST_FAILURE;
13    }
14
15    XGpio_SetDataDirection(&GpioOutput, LED_CHANNEL, 0x0);
16
17    XGpio_DiscreteWrite(&GpioOutput, LED_CHANNEL, value);
18
19    for (Delay = 0; Delay < LED_DELAY; Delay++);
20
21    XGpio_DiscreteWrite(&GpioOutput, LED_CHANNEL, 0xF);
22
23    return XST_SUCCESS;
24 }
25
26
27 XStatus Input(Xuint16 DeviceId, Xuint32 *DataRead)
28 {
29    XStatus Status;
30
31    Status = XGpio_Initialize(&GpioInput, DeviceId);
32    if (Status != XST_SUCCESS)
33    {
34       return XST_FAILURE;
35    }
36
37    XGpio_SetDataDirection(&GpioInput, LED_CHANNEL, 0xFFFFFFFF);
38
39    *DataRead = XGpio_DiscreteRead(&GpioInput, LED_CHANNEL);
40
41    return XST_SUCCESS;
42 }
```

Apéndice D

Código Interfaz Gráfica de Usuario

Archivo: `main.cpp`

```
1 Generado automaticamente por el Qt3 Designer
2 #include <qapplication.h>
3 #include "GUI.h"
4
5 int main( int argc, char ** argv )
6 {
7     QApplication a( argc, argv );
8     Form1 w;
9     w.show();
10    a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
11    return a.exec();
12 }
```

Archivo: `GUI.h`

```
1
2 Generado automaticamente por el Qt3 Designer
3 #ifndef FORM1_H
4 #define FORM1_H
5
6 #include <qvariant.h>
7 #include <qpixmap.h>
8 #include <qmainwindow.h>
9
10 class QVBoxLayout;
11 class QHBoxLayout;
12 class QGridLayout;
13 class QSpacerItem;
14 class QAction;
15 class QActionGroup;
16 class QToolBar;
17 class QPopupMenu;
18 class QTabWidget;
19 class QWidget;
20 class QButtonGroup;
```

```
21 class QRadioButton;
22 class QGroupBox;
23 class QLabel;
24 class QLineEdit;
25 class QCheckBox;
26 class QPushButton;
27
28 class Form1 : public QMainWindow
29 {
30     Q_OBJECT
31
32 public:
33     Form1( QWidget* parent = 0, const char* name = 0, WFlags fl = WType_TopLevel );
34     ~Form1();
35
36     QTabWidget* pestanas;
37     QWidget* tab;
38     QButtonGroup* grupoProtocolos;
39     QRadioButton* botonTCP;
40     QRadioButton* botonUDP;
41     QRadioButton* botonICMP;
42     QButtonGroup* grupoServicios;
43     QRadioButton* botonFTP;
44     QRadioButton* botonWHOIS;
45     QRadioButton* botonSMTP;
46     QRadioButton* botonDNS;
47     QRadioButton* botonHTTP;
48     QRadioButton* botonSSH;
49     QGroupBox* grupoIPs;
50     QLabel* etiquetaOrigen;
51     QLabel* etiquetaDestino;
52     QLineEdit* IPDestino;
53     QLineEdit* IPOrigen;
54     QCheckBox* aceptarPaquete;
55     QPushButton* botonFinalizar;
56     QPushButton* botonCancelar;
57     QCheckBox* entradasalida;
58     QPushButton* botonNueva;
59     QWidget* tab_2;
60     QGroupBox* groupBox2;
61     QLineEdit* lineEdit5;
62     QPushButton* pushButton8;
63     QGroupBox* groupBox3;
64     QMenuBar *MenuBar;
65     QPopupMenu *fileMenu;
66     QPopupMenu *editMenu;
67     QPopupMenu *helpMenu;
68     QToolBar *toolBar;
69     QAction* fileNewAction;
70     QAction* fileOpenAction;
71     QAction* fileSaveAction;
72     QAction* fileSaveAsAction;
73     QAction* filePrintAction;
74     QAction* fileExitAction;
75     QAction* editUndoAction;
76     QAction* editRedoAction;
77     QAction* editCutAction;
78     QAction* editCopyAction;
79     QAction* editPasteAction;
80     QAction* editFindAction;
81     QAction* helpContentsAction;
82     QAction* helpIndexAction;
83     QAction* helpAboutAction;
84
85 public slots:
86     virtual void fileNew();
```

```
 87     virtual void fileOpen();
 88     virtual void fileSave();
 89     virtual void fileSaveAs();
 90     virtual void filePrint();
 91     virtual void fileExit();
 92     virtual void editUndo();
 93     virtual void editRedo();
 94     virtual void editCut();
 95     virtual void editCopy();
 96     virtual void editPaste();
 97     virtual void editFind();
 98     virtual void helpIndex();
 99     virtual void helpContents();
100     virtual void helpAbout();
101     virtual void nuevaRegla();
102     virtual void finalizar();
103     virtual void cancelar();
104
105 protected:
106
107 protected slots:
108     virtual void languageChange();
109
110 private:
111     QPixmap image0;
112     QPixmap image1;
113     QPixmap image2;
114     QPixmap image3;
115     QPixmap image4;
116     QPixmap image5;
117     QPixmap image6;
118     QPixmap image7;
119     QPixmap image8;
120     QPixmap image9;
121
122 };
123
124#endif // FORM1_H
```

Archivo: `GUI.cpp`

```
 1 Generado automaticamente por el Qt3Designer
 2#ifndef FORM1_H
 3#define FORM1_H
 4
 5#include <qvariant.h>
 6#include <qpixmap.h>
 7#include <qmainwindow.h>
 8
 9 class QVBoxLayout;
10 class QHBoxLayout;
11 class QGridLayout;
12 class QSpacerItem;
13 class QAction;
14 class QActionGroup;
15 class QToolBar;
16 class QPopupMenu;
17 class QTabWidget;
18 class QWidget;
19 class QButtonGroup;
20 class QRadioButton;
```

```
21 class QGroupBox;
22 class QLabel;
23 class QLineEdit;
24 class QCheckBox;
25 class QPushButton;
26
27 class Form1 : public QMainWindow
28 {
29     Q_OBJECT
30
31 public:
32     Form1( QWidget* parent = 0, const char* name = 0, WFlags fl = WType_TopLevel );
33     ~Form1();
34
35     QTabWidget* pestanas;
36     QWidget* tab;
37     QButtonGroup* grupoProtocolos;
38     QRadioButton* botonTCP;
39     QRadioButton* botonUDP;
40     QRadioButton* botonICMP;
41     QButtonGroup* grupoServicios;
42     QRadioButton* botonFTP;
43     QRadioButton* botonWHOIS;
44     QRadioButton* botonSMTP;
45     QRadioButton* botonDNS;
46     QRadioButton* botonHTTP;
47     QRadioButton* botonSSH;
48     QGroupBox* grupoIPs;
49     QLabel* etiquetaOrigen;
50     QLabel* etiquetaDestino;
51     QLineEdit* IPDestino;
52     QLineEdit* IPOrigen;
53     QCheckBox* aceptarPaquete;
54     QPushButton* botonFinalizar;
55     QPushButton* botonCancelar;
56     QCheckBox* entradasalida;
57     QPushButton* botonNueva;
58     QWidget* tab_2;
59     QGroupBox* groupBox2;
60     QLineEdit* lineEdit5;
61     QPushButton* pushButton8;
62     QGroupBox* groupBox3;
63     QMenuBar *MenuBar;
64     QPopupMenu *fileMenu;
65     QPopupMenu *editMenu;
66     QPopupMenu *helpMenu;
67     QToolBar *toolBar;
68     QAction* fileNewAction;
69     QAction* fileOpenAction;
70     QAction* fileSaveAction;
71     QAction* fileSaveAsAction;
72     QAction* filePrintAction;
73     QAction* fileExitAction;
74     QAction* editUndoAction;
75     QAction* editRedoAction;
76     QAction* editCutAction;
77     QAction* editCopyAction;
78     QAction* editPasteAction;
79     QAction* editFindAction;
80     QAction* helpContentsAction;
81     QAction* helpIndexAction;
82     QAction* helpAboutAction;
83
84 public slots:
85     virtual void fileNew();
86     virtual void fileOpen();
```

```
 87     virtual void fileSave();
 88     virtual void fileSaveAs();
 89     virtual void filePrint();
 90     virtual void fileExit();
 91     virtual void editUndo();
 92     virtual void editRedo();
 93     virtual void editCut();
 94     virtual void editCopy();
 95     virtual void editPaste();
 96     virtual void editFind();
 97     virtual void helpIndex();
 98     virtual void helpContents();
 99     virtual void helpAbout();
100     virtual void nuevaRegla();
101     virtual void finalizar();
102     virtual void cancelar();
103
104 protected:
105
106 protected slots:
107     virtual void languageChange();
108
109 private:
110     QPixmap image0;
111     QPixmap image1;
112     QPixmap image2;
113     QPixmap image3;
114     QPixmap image4;
115     QPixmap image5;
116     QPixmap image6;
117     QPixmap image7;
118     QPixmap image8;
119     QPixmap image9;
120
121 };
122
123 #endif // FORM1_H
```

Archivo: `moc_form1.cpp`

```
 1 Generado automaticamente por el Qt3 Designer
 2
 3 #undef QT_NO_COMPAT
 4 #include "form1.h"
 5 #include <qmetaobject.h>
 6 #include <qapplication.h>
 7
 8 #include <private/qucomextra_p.h>
 9 #if !defined(Q_MOC_OUTPUT_REVISION) || (Q_MOC_OUTPUT_REVISION != 26)
10 #error "This file was generated using the moc from 3.3.8b. It"
11 #error "cannot be used with the include files from this version of Qt."
12 #error "(The moc has changed too much.)"
13 #endif
14
15 const char *Form1::className() const
16 {
17     return "Form1";
18 }
19
20 QMetaObject *Form1::metaObj = 0;
21 static QMetaObjectCleanUp cleanUp_Form1( "Form1", &Form1::staticMetaObject );
```

```
22
23#ifndef QT_NO_TRANSLATION
24 QString Form1::tr( const char *s, const char *c )
25 {
26     if ( qApp )
27         return qApp->translate( "Form1", s, c, QApplication::DefaultCodec );
28     else
29         return QString::fromLatin1( s );
30 }
31#ifndef QT_NO_TRANSLATION_UTF8
32 QString Form1::trUtf8( const char *s, const char *c )
33 {
34     if ( qApp )
35         return qApp->translate( "Form1", s, c, QApplication::UnicodeUTF8 );
36     else
37         return QString::fromUtf8( s );
38 }
39#endif // QT_NO_TRANSLATION_UTF8
40
41#endif // QT_NO_TRANSLATION
42
43 QMetaObject* Form1::staticMetaObject()
44 {
45     if ( metaObj )
46         return metaObj;
47     QMetaObject* parentObject = QMainWindow::staticMetaObject();
48     static const QUMethod slot_0 = {"fileNew", 0, 0 };
49     static const QUMethod slot_1 = {"fileOpen", 0, 0 };
50     static const QUMethod slot_2 = {"fileSave", 0, 0 };
51     static const QUMethod slot_3 = {"fileSaveAs", 0, 0 };
52     static const QUMethod slot_4 = {"filePrint", 0, 0 };
53     static const QUMethod slot_5 = {"fileExit", 0, 0 };
54     static const QUMethod slot_6 = {"editUndo", 0, 0 };
55     static const QUMethod slot_7 = {"editRedo", 0, 0 };
56     static const QUMethod slot_8 = {"editCut", 0, 0 };
57     static const QUMethod slot_9 = {"editCopy", 0, 0 };
58     static const QUMethod slot_10 = {"editPaste", 0, 0 };
59     static const QUMethod slot_11 = {"editFind", 0, 0 };
60     static const QUMethod slot_12 = {"helpIndex", 0, 0 };
61     static const QUMethod slot_13 = {"helpContents", 0, 0 };
62     static const QUMethod slot_14 = {"helpAbout", 0, 0 };
63     static const QUMethod slot_15 = {"languageChange", 0, 0 };
64     static const QMetaData slot_tbl[] = {
65         { "fileNew()", &slot_0, QMetaData::Public },
66         { "fileOpen()", &slot_1, QMetaData::Public },
67         { "fileSave()", &slot_2, QMetaData::Public },
68         { "fileSaveAs()", &slot_3, QMetaData::Public },
69         { "filePrint()", &slot_4, QMetaData::Public },
70         { "fileExit()", &slot_5, QMetaData::Public },
71         { "editUndo()", &slot_6, QMetaData::Public },
72         { "editRedo()", &slot_7, QMetaData::Public },
73         { "editCut()", &slot_8, QMetaData::Public },
74         { "editCopy()", &slot_9, QMetaData::Public },
75         { "editPaste()", &slot_10, QMetaData::Public },
76         { "editFind()", &slot_11, QMetaData::Public },
77         { "helpIndex()", &slot_12, QMetaData::Public },
78         { "helpContents()", &slot_13, QMetaData::Public },
79         { "helpAbout()", &slot_14, QMetaData::Public },
80         { "languageChange()", &slot_15, QMetaData::Protected }
81     };
82     metaObj = QMetaObject::new_metaobject(
83         "Form1", parentObject,
84         slot_tbl, 16,
85         0, 0,
86#ifndef QT_NO_PROPERTIES
87         0, 0,
```

```
88          0, 0,
89 #endif // QT_NO_PROPERTIES
90          0, 0 );
91      cleanUp_Form1.setMetaObject( metaObj );
92      return metaObj;
93 }
94
95 void* Form1::qt_cast( const char* clname )
96 {
97      if ( !qstrcmp( clname, "Form1" ) )
98          return this;
99      return QMainWindow::qt_cast( clname );
100 }
101
102 bool Form1::qt_invoke( int _id, QUObject* _o )
103 {
104      switch ( _id - staticMetaObject()->slotOffset() ) {
105      case 0: fileNew(); break;
106      case 1: fileOpen(); break;
107      case 2: fileSave(); break;
108      case 3: fileSaveAs(); break;
109      case 4: filePrint(); break;
110      case 5: fileExit(); break;
111      case 6: editUndo(); break;
112      case 7: editRedo(); break;
113      case 8: editCut(); break;
114      case 9: editCopy(); break;
115      case 10: editPaste(); break;
116      case 11: editFind(); break;
117      case 12: helpIndex(); break;
118      case 13: helpContents(); break;
119      case 14: helpAbout(); break;
120      case 15: languageChange(); break;
121      default:
122          return QMainWindow::qt_invoke( _id, _o );
123      }
124      return TRUE;
125 }
126
127 bool Form1::qt_emit( int _id, QUObject* _o )
128 {
129      return QMainWindow::qt_emit(_id,_o);
130 }
131 #ifndef QT_NO_PROPERTIES
132
133 bool Form1::qt_property( int id, int f, QVariant* v)
134 {
135      return QMainWindow::qt_property( id, f, v);
136 }
137
138 bool Form1::qt_static_property( QObject* , int , int , QVariant* ){ return FALSE; }
139 #endif // QT_NO_PROPERTIES
```

Archivo: `moc_GUI.cpp`

```
1 Generado automaticamente por el Qt3 Designer
2 #undef QT_NO_COMPAT
3 #include "GUI.h"
4 #include <qmetaobject.h>
5 #include <qapplication.h>
6
```

```
 7#include <private/qucomextra_p.h>
 8#if !defined(Q_MOC_OUTPUT_REVISION) || (Q_MOC_OUTPUT_REVISION != 26)
 9#error "This file was generated using the moc from 3.3.8b. It"
10#error "cannot be used with the include files from this version of Qt."
11#error "(The moc has changed too much.)"
12#endif
13
14 const char *Form1::className() const
15 {
16     return "Form1";
17 }
18
19 QMetaObject *Form1::metaObj = 0;
20 static QMetaObjectCleanUp cleanUp_Form1( "Form1", &Form1::staticMetaObject );
21
22#ifndef QT_NO_TRANSLATION
23 QString Form1::tr( const char *s, const char *c )
24 {
25     if ( qApp )
26         return qApp->translate( "Form1", s, c, QApplication::DefaultCodec );
27     else
28         return QString::fromLatin1( s );
29 }
30#ifndef QT_NO_TRANSLATION_UTF8
31 QString Form1::trUtf8( const char *s, const char *c )
32 {
33     if ( qApp )
34         return qApp->translate( "Form1", s, c, QApplication::UnicodeUTF8 );
35     else
36         return QString::fromUtf8( s );
37 }
38#endif // QT_NO_TRANSLATION_UTF8
39
40#endif // QT_NO_TRANSLATION
41
42 QMetaObject* Form1::staticMetaObject()
43 {
44     if ( metaObj )
45         return metaObj;
46     QMetaObject* parentObject = QMainWindow::staticMetaObject();
47     static const QUMethod slot_0 = {"fileNew", 0, 0 };
48     static const QUMethod slot_1 = {"fileOpen", 0, 0 };
49     static const QUMethod slot_2 = {"fileSave", 0, 0 };
50     static const QUMethod slot_3 = {"fileSaveAs", 0, 0 };
51     static const QUMethod slot_4 = {"filePrint", 0, 0 };
52     static const QUMethod slot_5 = {"fileExit", 0, 0 };
53     static const QUMethod slot_6 = {"editUndo", 0, 0 };
54     static const QUMethod slot_7 = {"editRedo", 0, 0 };
55     static const QUMethod slot_8 = {"editCut", 0, 0 };
56     static const QUMethod slot_9 = {"editCopy", 0, 0 };
57     static const QUMethod slot_10 = {"editPaste", 0, 0 };
58     static const QUMethod slot_11 = {"editFind", 0, 0 };
59     static const QUMethod slot_12 = {"helpIndex", 0, 0 };
60     static const QUMethod slot_13 = {"helpContents", 0, 0 };
61     static const QUMethod slot_14 = {"helpAbout", 0, 0 };
62     static const QUMethod slot_15 = {"nuevaRegla", 0, 0 };
63     static const QUMethod slot_16 = {"finalizar", 0, 0 };
64     static const QUMethod slot_17 = {"cancelar", 0, 0 };
65     static const QUMethod slot_18 = {"languageChange", 0, 0 };
66     static const QMetaData slot_tbl[] = {
67         { "fileNew()", &slot_0, QMetaData::Public },
68         { "fileOpen()", &slot_1, QMetaData::Public },
69         { "fileSave()", &slot_2, QMetaData::Public },
70         { "fileSaveAs()", &slot_3, QMetaData::Public },
71         { "filePrint()", &slot_4, QMetaData::Public },
72         { "fileExit()", &slot_5, QMetaData::Public },
```

```
73          { "editUndo()", &slot_6, QMetaData::Public },
74          { "editRedo()", &slot_7, QMetaData::Public },
75          { "editCut()", &slot_8, QMetaData::Public },
76          { "editCopy()", &slot_9, QMetaData::Public },
77          { "editPaste()", &slot_10, QMetaData::Public },
78          { "editFind()", &slot_11, QMetaData::Public },
79          { "helpIndex()", &slot_12, QMetaData::Public },
80          { "helpContents()", &slot_13, QMetaData::Public },
81          { "helpAbout()", &slot_14, QMetaData::Public },
82          { "nuevaRegla()", &slot_15, QMetaData::Public },
83          { "finalizar()", &slot_16, QMetaData::Public },
84          { "cancelar()", &slot_17, QMetaData::Public },
85          { "languageChange()", &slot_18, QMetaData::Protected }
86      };
87      metaObj = QMetaObject::new_metaobject(
88          "Form1", parentObject,
89          slot_tbl, 19,
90          0, 0,
91  #ifndef QT_NO_PROPERTIES
92          0, 0,
93          0, 0,
94  #endif // QT_NO_PROPERTIES
95          0, 0 );
96      cleanUp_Form1.setMetaObject( metaObj );
97      return metaObj;
98  }
99
100 void* Form1::qt_cast( const char* clname )
101 {
102     if ( !qstrcmp( clname, "Form1" ) )
103         return this;
104     return QMainWindow::qt_cast( clname );
105 }
106
107 bool Form1::qt_invoke( int _id, QUObject* _o )
108 {
109     switch ( _id - staticMetaObject()->slotOffset() ) {
110     case 0: fileNew(); break;
111     case 1: fileOpen(); break;
112     case 2: fileSave(); break;
113     case 3: fileSaveAs(); break;
114     case 4: filePrint(); break;
115     case 5: fileExit(); break;
116     case 6: editUndo(); break;
117     case 7: editRedo(); break;
118     case 8: editCut(); break;
119     case 9: editCopy(); break;
120     case 10: editPaste(); break;
121     case 11: editFind(); break;
122     case 12: helpIndex(); break;
123     case 13: helpContents(); break;
124     case 14: helpAbout(); break;
125     case 15: nuevaRegla(); break;
126     case 16: finalizar(); break;
127     case 17: cancelar(); break;
128     case 18: languageChange(); break;
129     default:
130         return QMainWindow::qt_invoke( _id, _o );
131     }
132     return TRUE;
133 }
134
135 bool Form1::qt_emit( int _id, QUObject* _o )
136 {
137     return QMainWindow::qt_emit(_id,_o);
138 }
```

```
139#ifndef QT_NO_PROPERTIES
140
141 bool Form1::qt_property( int id, int f, QVariant* v)
142 {
143     return QMainWindow::qt_property( id, f, v);
144 }
145
146 bool Form1::qt_static_property( QObject* , int , int , QVariant* ){ return FALSE; }
147#endif // QT_NO_PROPERTIES
```

Archivo: `GUI.ui.h`

```
 1#include "FuncionesConversion.h"
 2#include "FuncionesConversion.c"
 3#include "sl.h"
 4#include "sl.c"
 5 int numReglas=0;
 6 char reglasE[13][10];
 7 char regla[13];
 8 char habilE[5];//guarda cuales son los campo habilitados
 9 void Form1::nuevaRegla()
10 {
11     char iporigen[16];
12     char ipdestino[16];
13     char *iporigenhex;
14     char *ipdestinohex;
15     struct unPuerto puertohex;
16     char en;
17     char entradaSalida;
18     unsigned char puerto1;
19     unsigned char protocolo1;
20     /*se comprueba si alguno de los protocoles esta seleccionado
21 si alguno esta seleccionado, se guarda un valor en el vector habil, inidicandolo,
22 se realiza la conversion*/
23     if(botonTCP->isChecked()){
24     protocolo1=CampotoHex(6);
25     habilE[2]='1';
26     }
27     if(botonUDP->isChecked()){
28     protocolo1=CampotoHex(17);
29     habilE[2]='1';
30     }
31     if(botonICMP->isChecked()){
32     protocolo1=CampotoHex(1);
33     habilE[2]='1';
34     }
35     /*se comprueba si la IP contiene un valor distinto de nulo
36 de ser asi se realiza la conversion y se carga un valor en el vector habil*/
37    if(IPOrigen->text()!='\0'){
38     strcpy(iporigen,IPOrigen->text());
39     printf("%s\t\n", iporigen);
40     iporigenhex=ConvierteIp(iporigen);
41     habilE[1]='1';
42     }
43    else
44        strcpy(iporigenhex,"0000");
45     if(IPDestino->text()!='\0'){
46     strcpy(ipdestino, IPDestino->text());
47     printf("%s\t\n", ipdestino);
48     ipdestinohex=ConvierteIp(ipdestino);
49     habilE[0]='1';
```

```
50      }
51      else
52      strcpy(ipdestinohex,"0000");
53      /*se comprueba si alguno de los puertos esta seleccionado
54 si alguno esta seleccionado, se guarda un valor en el vector habil, inidicandolo
55 se realiza la conversion*/
56      if(botonFTP->isChecked()){
57      puertohex=puerto2Hex(21);
58      habilE[3]='1';
59      }
60      if(botonWHOIS->isChecked()){
61      puertohex=puerto2Hex(43);
62      habilE[3]='1';
63      }
64      if(botonSMTP->isChecked()){
65      puertohex=puerto2Hex(25);
66      habilE[3]='1';
67      }
68      if(botonSSH->isChecked()){
69      puertohex=puerto2Hex(22);
70      printf("%c", puerto1);
71      printf("\n %x",puerto1);
72      }
73      if(botonDNS->isChecked()){
74      puertohex=puerto2Hex(53);
75      habilE[3]='1';
76      }
77      if(botonHTTP->isChecked()){
78      puertohex=puerto2Hex(80);
79      habilE[3]='1';
80      }
81      /*comprueba si la regla sera para aceptar o rechazar*/
82      if(aceptarPaquete->isChecked())
83      habilE[4]='1';
84      /*comprueba si la regla sera de entrada o de salida*/
85      if(entradasalida->isChecked())
86      entradaSalida='1';
87      else
88      entradaSalida='0';
89      /*convierte la informacion en el vector habil al valor hexadecimal correspondiente*/
90      en=HabilitadostoHex(habilE);
91      if(numReglas<10){
92      /*Carga los distintos campos de las reglas al vector que las contendra*/
93      reglasE[0][numReglas]=entradaSalida;
94      reglasE[1][numReglas]=en;
95      reglasE[2][numReglas]=40;
96      reglasE[3][numReglas]=40;
97      reglasE[4][numReglas]=protocolo1;
98      reglasE[5][numReglas]=iporigenhex[0];
99      reglasE[6][numReglas]=iporigenhex[1];
100     reglasE[7][numReglas]=iporigenhex[2];
101     reglasE[8][numReglas]=iporigenhex[3];
102     reglasE[9][numReglas]=ipdestinohex[0];
103     reglasE[10][numReglas]=ipdestinohex[1];
104     reglasE[11][numReglas]=ipdestinohex[2];
105     reglasE[12][numReglas]=ipdestinohex[3];
106     numReglas++;
107     }
108 }
109 void Form1::finalizar()
110 {
111     char conexion[]={"conexion"};
112     speed_t vel;
113     int conect;
114     vel=56;
115     /*Crea la conexion*/
```

```
116     conect=serial_open(conexion,vel);
117     for(int z=0;z<numReglas;z++){
118     /*Extrae reglas del vector*/
119     for(int x=0;x<13;x++){
120         regla[x]=reglasE[x][numReglas];
121     }
122     /*Envia reglas*/
123     serial_send(conect, regla,13);
124     }
125     /*cierra conexion*/
126     serial_close(conect);
127 }
128
129
130 void Form1::cancelar()
131 {
132     /*Limpia todo*/
133     botonTCP->setChecked(false);
134     botonUDP->setChecked(false);
135     botonICMP->setChecked(false);
136     botonFTP->setChecked(false);
137     botonWHOIS->setChecked(false);
138     botonSMTP->setChecked(false);
139     botonSSH->setChecked(false);
140     botonDNS->setChecked(false);
141     botonHTTP->setChecked(false);
142     aceptarPaquete->setChecked(false);
143     entradasalida->setChecked(false);
144     IPOrigen->setText('\0');
145     IPDestino->setText('\0');
146     numReglas=0;
147 }
148
149
150 void Form1::limpiar()
151 {
152     /*Limpia  el texto*/
153     bitacora->setText(" ");
154 }
155 void Form1::recibir()
156 {
157     char conexion[]={"conexion"};
158     char datos[200];
159     int conect;
160     speed_t vel=56;
161     conect=serial_open(conexion,vel); //abre conexion
162     serial_recive(conect,datos,200);// Recibe datos
163     serial_close(conect);//cierra conexion
164     bitacora->setText(datos);//Escribe el texto
165 }
```

Archivo: `sl.h`

```
#ifndef sl
        #define sl
        #include <termios.h>
        #include <unistd.h>
        #include <fcntl.h>
        #include <string.h>
        #include <stdlib.h>
        #include <stdio.h>

        int  serial_open(char *serial_name, speed_t baud);
        int serial_send(int serial_fd, char *data, int size);
        int  serial_recive(int serial_fd, char *data, int size);
        void serial_close(int fd);
#endif
```

Archivo: `sl.c`

```
#include "sl.h"


int serial_open(char *serial_name, speed_t baud)
{
    struct termios newtermios;
    int fd;

    if ((fd = open(serial_name,O_RDWR | O_NOCTTY)) == -1)
    {
        perror("Error al abrir el puerto serie: ");
        exit(-1);
    }

    /* Configurar atributos del puerto */

    newtermios.c_cflag= CBAUD | CS8 | CLOCAL | CREAD | CRTSCTS;
    newtermios.c_iflag=IGNPAR;
    newtermios.c_oflag=0;
    newtermios.c_lflag=0;
    newtermios.c_cc[VMIN]=1;
    newtermios.c_cc[VTIME]=0;


    cfsetospeed(&newtermios,baud);
    cfsetispeed(&newtermios,baud);

    if (tcflush(fd,TCIFLUSH)==-1)
    {
        perror("Error en tcflush: ");
        return -1;
    }

    if (tcflush(fd,TCOFLUSH)==-1)
    {
        perror("Error en tcflush: ");
        return -1;
```

```
38      }
39
40      if (tcsetattr(fd,TCSANOW,&newtermios)==-1)
41      {
42          perror("Error en tcsetattr: ");
43          return -1;
44      }
45
46    return fd;
47 }
48
49 int serial_send(int fd, char *data, int size)
50 {
51      int count=0, n;
52      do
53      {
54          if ((n=write (fd, &data[count], size-count)) == -1)//escribe
55          {
56              perror("Error al escribir los datos: ");
57              return -1;
58          }
59          count+=n;
60      }
61      while (count<size);
62
63      return count;
64 }
65
66
67 int serial_recive(int fd, char *data, int size)
68 {
69      int count=0, n;
70      do
71      {
72          if ((n=read (fd, &data[count], size-count)) == -1)//lee
73          {
74              perror("Error al leer los datos: ");
75              return -1;
76          }
77          count+=n;
78      }
79      while (count<size);
80
81      return count;
82 }
83
84 void serial_close(int fd)
85 {
86    close(fd);//cierra
87 }
```

Archivo: `FuncionesConversion.h`

```
1#ifndef funciones
2   #define funciones
3   #include<stdio.h>
4   #include<stdlib.h>
5   #include<string.h>
6   #include<ctype.h>
7    struct unPuerto{
8     char h;
```

```
 9    char l;};
10   char HabilitadostoHex(char *);
11   char CampotoHex(int);
12   char *ConvierteIp(char*);
13   struct   unPuerto  puerto2Hex(int);
14#endif
```

Archivo: `Funciones Conversión.c`

```
1#include"FuncionesConversion.h"
2char HabilitadostoHex(char *habilitados){
3     char hex='\0';
4          /*Hace un enmascaramiento para pasar las
5  * banderas de un vector a un solo byte*/
6     if(habilitados[4]=='1')
7          hex=hex|0x01;
8     if(habilitados[3]=='1')
9          hex=hex|0x02;
10    if(habilitados[2]=='1')
11         hex=hex|0x04;
12    if(habilitados[1]=='1')
13         hex=hex|0x8;
14    if(habilitados[0]=='1')
15         hex=hex|0x16;
16    return hex;
17}
18char CampotoHex(int i){
19char hex;
20/*Se convierte de int a char*/
21if(i>255)
22         hex='\0';
23return hex=i;
24}
25char *ConvierteIp(char *dirIp){
26   char *pch;
27   char arreglo[4];
28   int i=0;
29   arreglo[0]='\0';
30/*Separa tokens*/
31   pch= strtok (dirIp,".");
32   arreglo[0]=CampotoHex(atoi(pch));
33   for(i=1;i<4;i++){
34     pch= strtok (NULL,".");
35/*convierte de cadena a entero*/
36     arreglo[i]=CampotoHex(atoi(pch));
37   }
38   return arreglo;
39}
40struct unPuerto  puerto2Hex(int numero_puerto){
41    struct unPuerto regresa;
42    char hex[5];
43    char tmp[10];
44    char h[3],l[3];
45    char *nptr;
46    int longitud, i;
47    h[2]='\0';l[2]='\0';
48         /*pasa de entero en formato hexadecimal a una cadena*/
49    sprintf(hex,"%x",numero_puerto);
50    longitud=strlen(hex);
51         /*Se completan los digitos hexadecimales para que sean 4*/
52    longitud=4-longitud;
```

```
53      for(i=0;i<longitud;i++){
54          tmp[i]='0';
55      }
56      strcat(hex,tmp);
57          /*se asignan los valores*/
58      h[0]=hex[2];
59      h[1]=hex[3];
60      l[0]=hex[0];
61      l[1]=hex[1];
62      regresa.h=strtol(h,&nptr,16);
63      regresa.l=strtol(l,&nptr,16);
64      return regresa;
65  }
```

Apéndice E

Manual para la Interfaz Gráfica de Usuario

En este apéndice se describirán los pasos a seguir para compilar, ejecutar y utilizar la Interfaz Gráfica, que es la herramienta que le permitirá al usuario la creación de las reglas y la visualización de los resultados producidos por el *firewall*.

E.1. Bibliotecas requeridas

Se requieren instaladas las siguientes bibliotecas para la compilación y ejecución de la Interfaz Gráfica de Usuario:

- qt3-3.3 o superior
- qt3-devel-3.3 o superior

E.2. Compilación y ejecución

Desde una terminal, como usuario normal, se entra al directorio GUI que contienen los archivos con los códigos fuente para la Interfaz Gráfica.

1. $ cd GUI/
2. $ qmake
3. $ make
   Una vez compilada la aplicación, se ejecutara con el siguiente comando:
4. ./GUI

E.3. Uso de la Interfaz Gráfica de Usuario

E.3.1. Creación de las reglas

Después de ejecutar la interfaz, aparecerá la ventana mostrada en la siguiente figura.

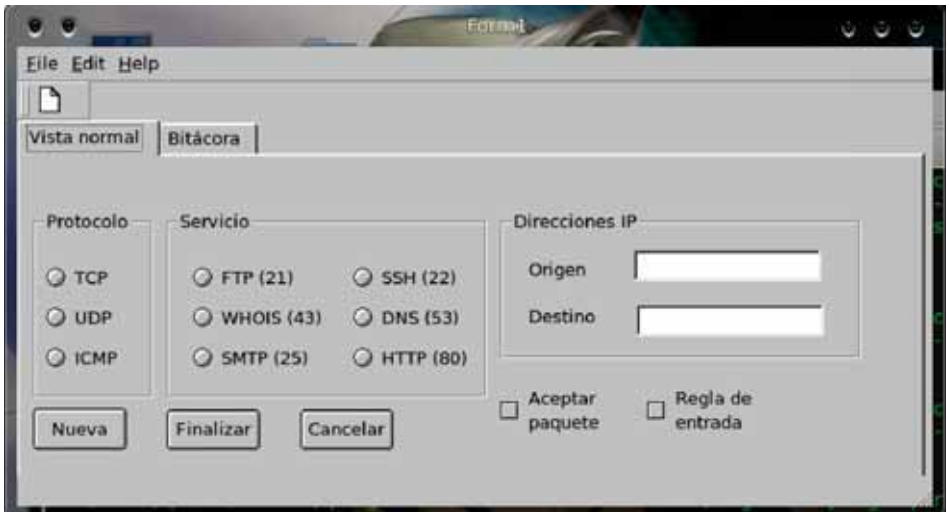


Figura E.1: Interfaz para la carga de reglas.

Se debe de hacer clic en los botones circulares contenidos bajo los cuadros con los nombre "Protocolo", "Servicio" o "Direcciones IP". Por ejemplo si se necesita que la regla aplique a paquetes con el protocolo TCP, se deberá hacer 'clic' en el botón circular que se encuentra al lado de TCP, como se muestra en la siguiente figura.

Figura E.2: Ejemplo de utilización 1.

Se procede de forma análoga con el campo de servicio.
Para las Direcciones IP, tanto de entrada como de salida, se escribirá la dirección ip como 4 números enteros, con valor de entre o y 255, separados por puntos (.), esta será la dirección IP sobre la cuál se aplicará la regla.

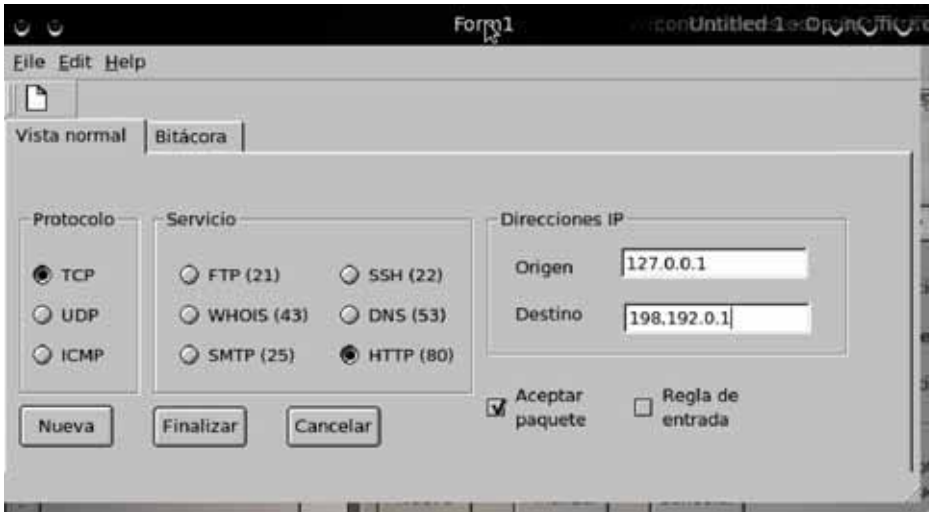


Figura E.3: Ejemplo de utilización 2.

No es necesario elegir una opción de cada campo, sólo aquellas que se van a utilizar; por ejemplo, si en la regla que estamos creando no importa la dirección de origen, entonces se deberá de dejar en blanco, lo mismo para los servicios y los protocolos.

Posteriormente se define se la regla será para aceptar o rechazar, seleccionando o no "Aceptar paquete" Aceptar significa que a los paquetes que cumplan con las características seleccionadas se les permitirá el paso.

Si Aceptar paquete no está seleccionado a dichos paquetes no se les permitirá el paso. Por ultimo se debe de determinar si la regla aplica a las entradas o a las salidas, seleccionando o no la casilla "Regla de entrada". Si la casilla está seleccionada, se considera regla de entrada, de lo contrario es una regla de salida.

Una regla de entrada aplicará a los paquetes que vengan desde la red hacia la computadora, una regla de salida es aquella que aplica a los paquetes que van desde la computadora hacia la red. Una vez que se haya diseñado la regla, se hará 'clic' en el botón "Nueva" para que la regla se cree. Sólo se permiten un máximo de 10 reglas sin importar si son de entrada o de salida. Para programar las reglas al *firewall*, se tiene que habilitar este para que reciba las reglas y se hace clic en el botón Finalizar. Para borrar las reglas creadas sin programar el *firewall*, se hace 'clic' en el botón "Cancelar".

E.3.2. Utilización de la bitácora

Para tener acceso a la bitácora del *firewall* para conocer las direcciones de los paquetes rechazados y el número de los mismos, se hace 'clic' en la pestaña "Bitácora" ubicada al lado de la "Vista normal". Al hacer 'clic' aparecerá la ventana mostrada en la siguiente figura.

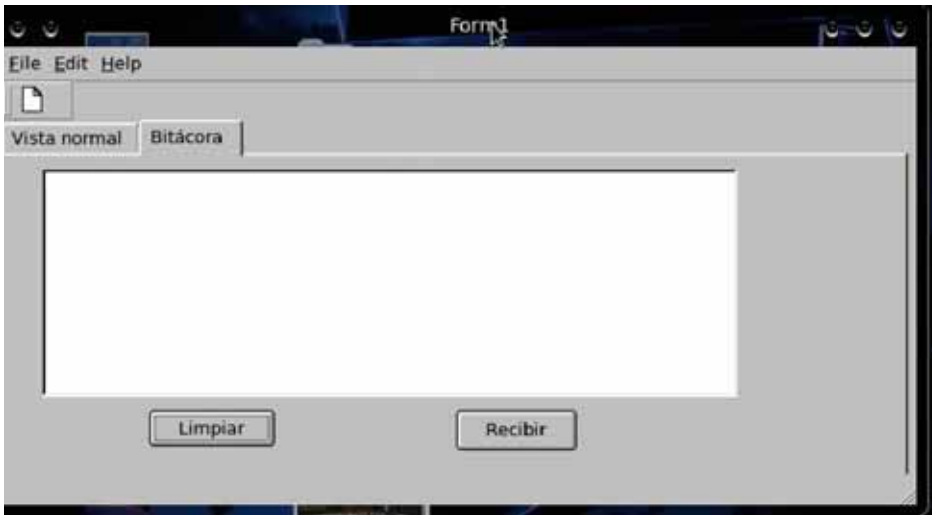


Figura E.4: Ejemplo de utilización 3.

Para recibir la información del la bitácora, se habilita el *firewall*, para que transmita información, y se hace 'clic' en el botón recibir. Los datos de la bitácora se escribirán en el cuadro de texto.
Para limpiar el cuadro de texto y borrar la información que se haya mostrado allí, se hará 'clic' en el botón limpiar.
Para salir de la aplicación se hace 'clic' en el botón cerrar ubicado en la esquina superior derecha, o con la combinación de teclas 'ctr+c' desde la terminal en dónde se ejecutó la aplicación.

Apéndice F

Manual de Instalación del Entorno de desarrollo

F.1. Objetivo

El objetivo de este manual es guiar al usuario durante la instalación del entorno de desarrollo *Xilinx*, tanto EDK como ISE ambos con la version 8.2i. Ademas de solucionar algunos de los problemas que se presentan al utilizar éstas herramientas.

F.2. Instalación de requerimientos en Linux

En este manual se considerara que ya se tiene instalado un sistema operativo linux especialmente Debian o Fedora.

Lo primero que se tiene que realizar es instalar los compiladores de C y C++, ya que aunque las herramientas de *Xilinx* son auto contenidas no esta por demás tenerlas. Esto se realiza de la siguiente manera:

Como usuario root se ejecutan las siguientes lineas en Debian:

- # apt-get install gcc
- # apt-get install g++
- # apt-get install vim vim-full vim-gtk vim-gui-common

Como usuario root se ejecutan las siguientes lineas en Fedora:

- # yum install gcc

- # yum install g++
- # yum install gvim

La ultima linea es para instalar un editor de texto.

Debido a que la mayoría de las herramientas de *Xilinx* se desarrollaron en C++ es necesario instalar las bibliotecas con las clases desarrolladas y compiladas con versiones de g++ en versiones 3. Esto lo hacemos de la siguiente forma:

En Debian

- # apt-get install libstdc++5

En Fedora:

- # yum install libstdc++ o

- # yum install compat-libstdc++-33//

F.3. Instalación de ISE 8.2i

Una ves instalados los compiladores y las bibliotecas, procedemos a Instalar ISE 8.2i.

Es posible que la configuración por defecto del HAL, no permita la ejecución del script desde la ruta creada de forma dinámica, en ese caso se podría detener el HAL y montar manualmente el DVD.

Para detener el HAL en Debian ejecutar:

- # /etc/init.d/hal stop

Para Fedora

- # /etc/init.d/haldaemon stop

Ya detenido el HaL procedemos a montar el DVD.

Para montar DVD en Debian y Fedora se utiliza:

- # mkdir /mnt/disco
- # mount -t iso9660 /dev/dvd /mnt/disco

Para desmontar el DVD se utiliza en ambos:

- # umount /mnt/disco

Ya montado el DVD nos dirigimos a la ruta de montura para ejecutar el script de instalación esto es:

- $ ./setup

La instalación creará un directorio en tu home llamado *Xilinx*. Lo anterior no es recomendable para servidores multiusuario, ya que el software se instalaría para cada usuario. Para esta situación es preferible instalarlo como root en el directorio raíz y dejar que el administrador se haga cargo de la instalación y el otorgamiento de privilegios.

Al ejecutar la linea anterior se abrirá la siguiente pantalla:

Figura F.1: Pantalla de instalación.

Hacer clic en next.

Figura F.2: Pantalla de instalación.

Aceptar los términos de la licencia y hacer clic en next.

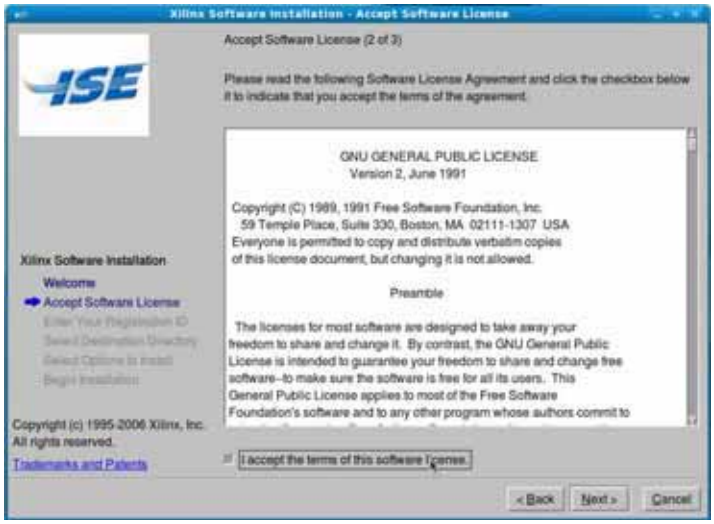


Figura F.3: Pantalla de instalación.

Aceptar los téminos y dar clic en next.

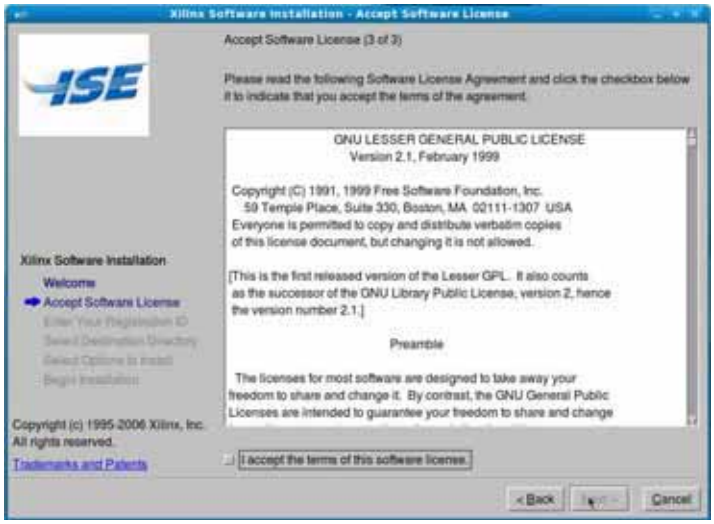


Figura F.4: Pantalla de instalación.

Aceptar los términos y dar clic en next.

Figura F.5: Pantalla de instalación.

Ingresa el número de registro y da clic en next.

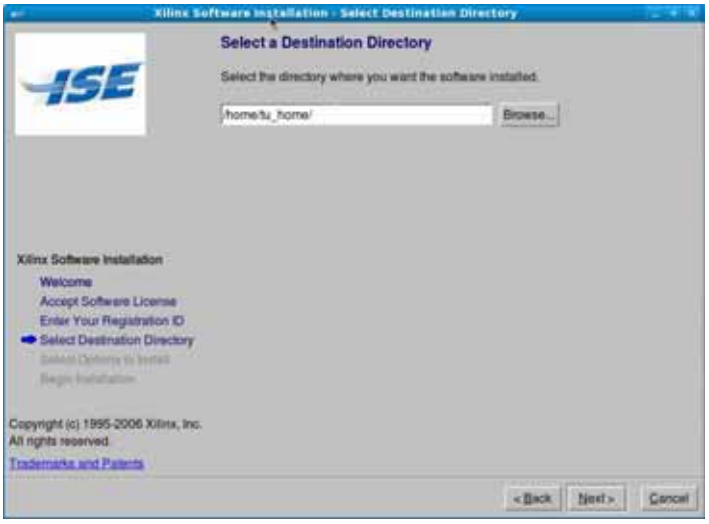


Figura F.6: Pantalla de instalación.

Seleccionas como directorio destino para la instalación /home/TU_ HOME/Xilinx y das clic en next.

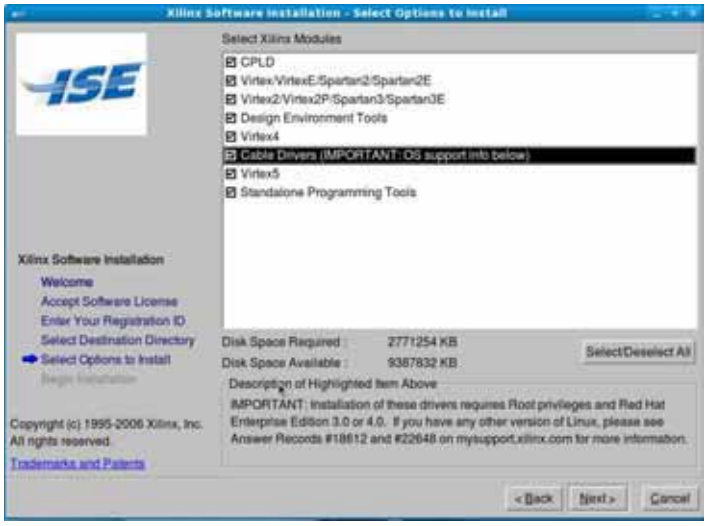


Figura F.7: Pantalla de instalación.

Seleccionamos los módulos a instalar:

- Virtex2/Virtex2P/Spartan3/Spartan3E
- Design Environment Tools
- Cable Drivers
- Standalone Programming tools

Y das clic en next.

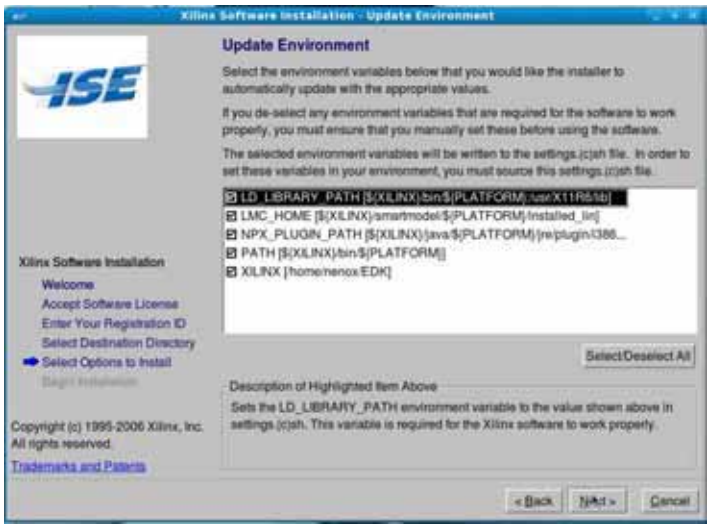


Figura F.8: Pantalla de instalación.

Se seleccionan todas las variables de ambiente y se da clic en next.

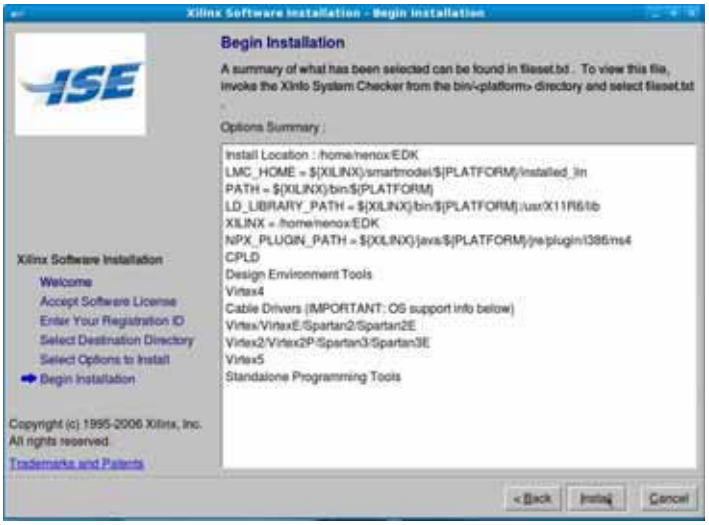


Figura F.9: Pantalla de instalación.

Se da clic en install.

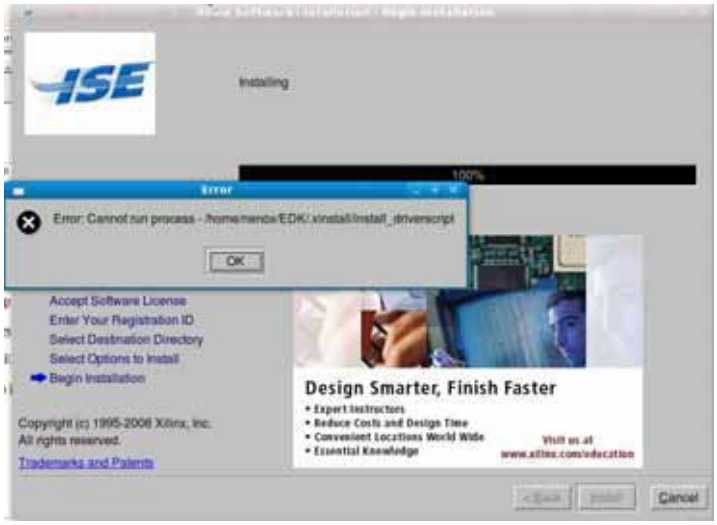


Figura F.10: Pantalla de instalación.

Después de terminada la instalación aparecerá el siguiente mensaje de error, se da clic en ok ya que nosotros utilizaremos otros drivers.

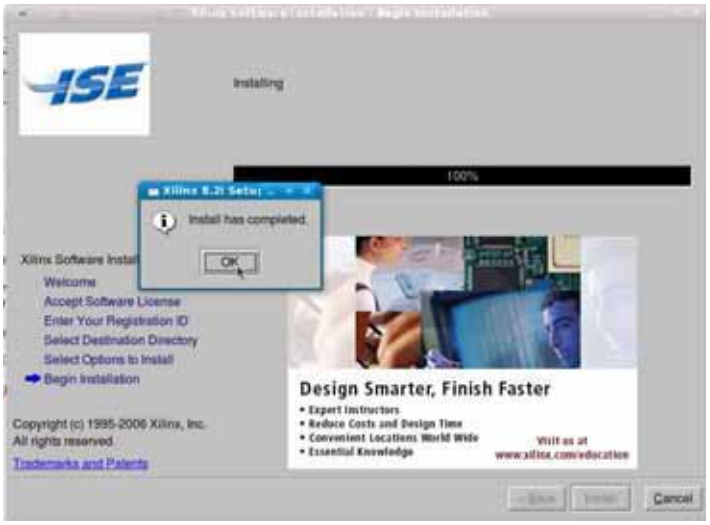


Figura F.11: Pantalla de instalación.

Clic en ok y termina el proceso de instalación de ISE 8.2i.

F.4. Instalación de EDK 8.2i

Se monta el DVD de la misma manera que el DVD de ISE y se ejecuta el script de instalación del DVD de EDK.

Al ejecutar el script se presenta la siguiente pantalla:

Figura F.12: Pantalla de instalación.

Dar clic en next.

Figura F.13: Pantalla de instalación.

Se aceptan los términos de la licencia y se da clic en next.

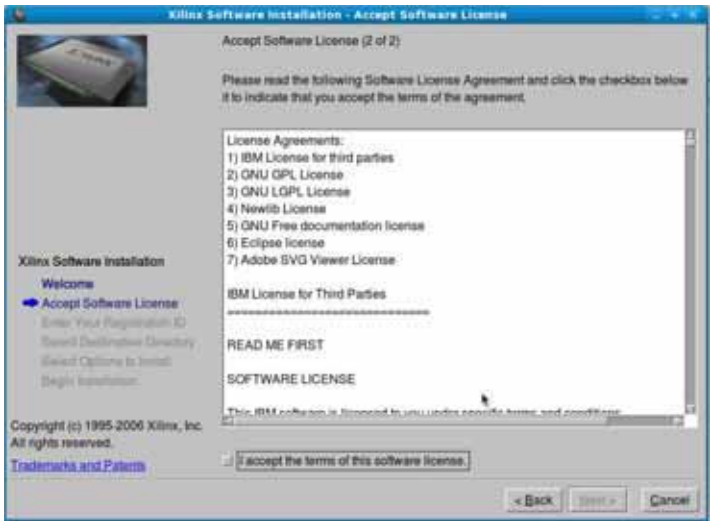


Figura F.14: Pantalla de instalación.

Se aceptan los términos y se da clic en next.

Figura F.15: Pantalla de instalación.

Se introduce el numero de registro y se da clic en next.

Figura F.16: Pantalla de instalación.

Se elige como directorio destino de la instalación un directorio llamado EDK en tu home de usuario y se da clic en next.

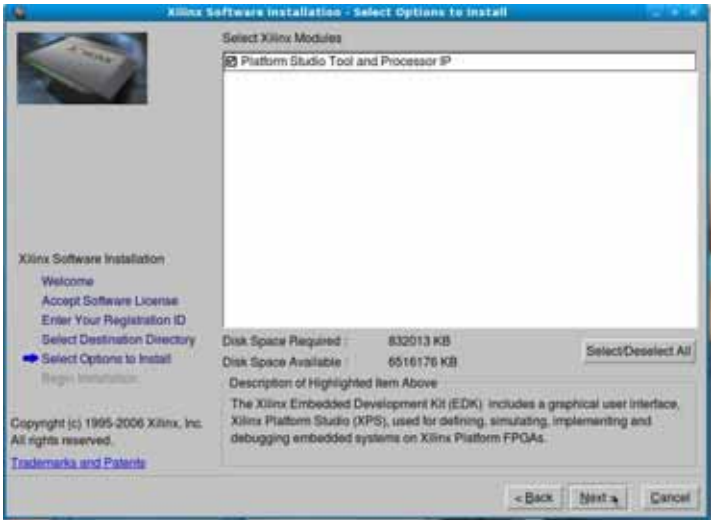


Figura F.17: Pantalla de instalación.

Se selecciona el módulo Platform Studio Tool and Processor IP y se da clic en next.

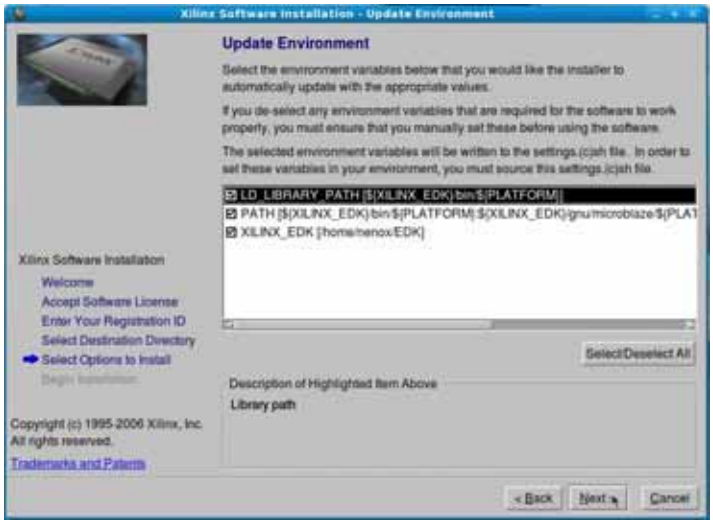


Figura F.18: Pantalla de instalación.

Se eligen todas las variables de ambiente y se da clic en next.

Figura F.19: Pantalla de instalación.

Se da clic en install.

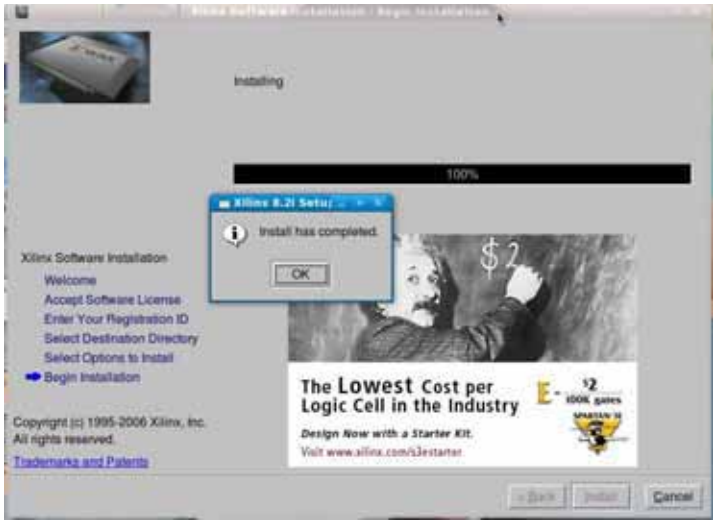


Figura F.20: Pantalla de instalación.

Clic en ok y finaliza la instalación de EDK.

F.5. Instalación de Drivers

Como algunas herramientas de EDK, requieren de la biblioteca dinámica: libdb-4.1.so, la cual contiene módulos para el acceso a bases de datos desde C++, la instalaremos de la siguiente manera:

proceso de instalación en Debian:

- # apt-get install libdb4.6++-dev
- # ln -s /usr/lib/libdb-4.6.so /usr/lib/libdb-4.1.so

proceso en Fedora:

- # yum install libdbi-devel
- # ln -s /usr/lib/libdb-4.x.so /usr/lib/libdb-4.1.so

Donde x es el numero de la versión de la biblioteca.

Ahora instalaremos la biblioteca libusb que es la biblioteca que nos permite crear drivers de sistema con acceso a USB. Esto se hace de la siguiente manera:

En Debian:

- # apt-get install libusb-dev

En fedora:

- # yum install libusb-devel libusb-static libusb1-devel libusb1-devel-doc libusb1-static

Después instalaremos el driver que nos permitirá bajar programas a la tarjeta así como configurar la tarjeta. Para lo que se deben bajar las fuentes de usb-driver y hacer lo siguiente:

Primero se extraen las fuentes, entramos al directorio donde han sido extraídas, se ejecuta make y se crea una variable de ambiente para que las herramientas de *Xilinx* puedan utilizarla.

- $ tar zxvf usb-driver-HEAD.tar.gz

- $ cd usb-driver
- $ make
- $ export LD_ PRELOAD=/path/to/libusb-driver.so

Para utilizar el dispositivo como un usuario ordinario, creamos el archivo llamado libusb-driver.rules de la siguiente forma:

- # gvim /etc/udev/rules.d/libusb-driver.rules

Y se le agrega la siguiente información:

- ACTION=="add", BUS=="usb", ATTRS{idVendor} =="$03fd$", MODE="666"

Como cuando vallamos compilar un programa en EDK el compilador de este llama al compilador gmake y en debian gmake no existe, entonces debemos crear un enlace simbólico de make en /usr/bin/ llamado gmake para que EDK lo encuentre.

- # ln -s /usr/bin/gmake /usr/bin/make

Ya tenemos instaladas la herramientas y librerías necesarias para poder ocupar ISE y EDK, ahora lo que sigue es instalar una hyperterminal para poder visualizar las salidas de los programas que ejecutemos en la tarjeta para lo que hay que instalar minicom.

En Debian:

- # apt-get install minicom

En Fedora:

- # yum install minicom

F.6. Solución a problemas

F.6.1. Simulación ISE

Al realizar alguna aplicación en ISE se tiene un problema con el simulador, ya que muestra un mensaje al querer simular que indica un problema con el compilador. Este problema se soluciona realizando algunos cambios en diferentes archivos.

En ruta RUTA_ISE/gnu/gcc/3.2.3/lin/bin/gcc-lib/i686-pc-linux-gnu/3.2.3 se realizaron cambios al archivo specs. Se remplazo la palabra -lc por -lcxil.

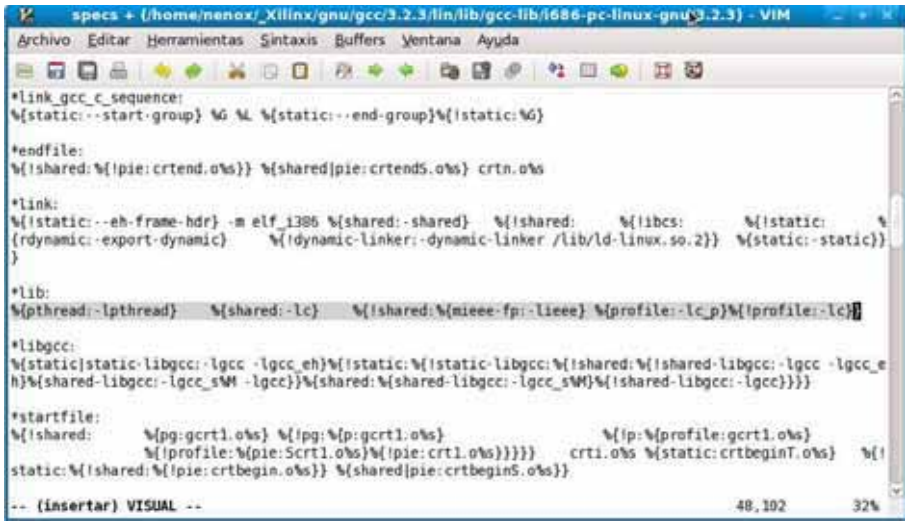


Figura F.21: Archivo specs original.

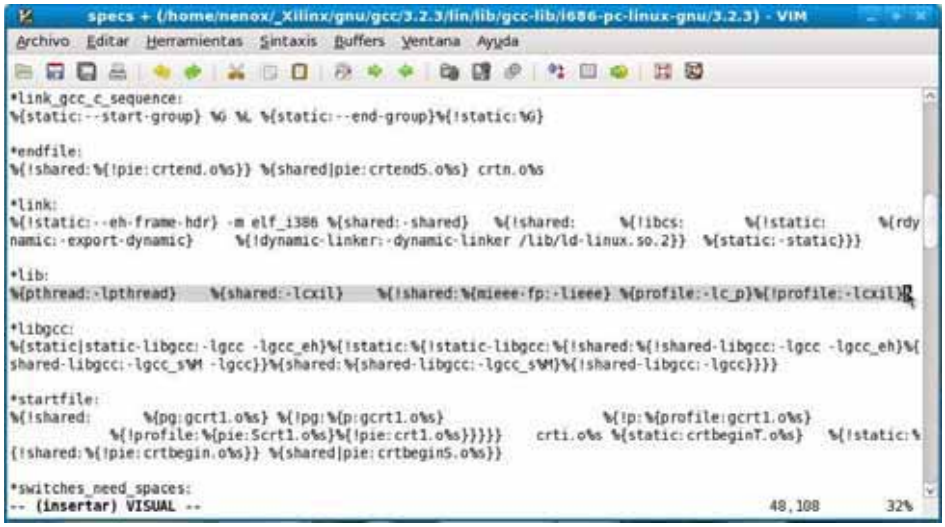


Figura F.22: Archivo specs modificado.

Después se copio la biblioteca libc.so a la ruta anterior con el nombre de libcxil.so. Al archivo libcxil.so se le quito el parámetro AS_NEEDED y los paréntesis que encierran la ruta que aparece delante del parámetro AS_NEEDED:

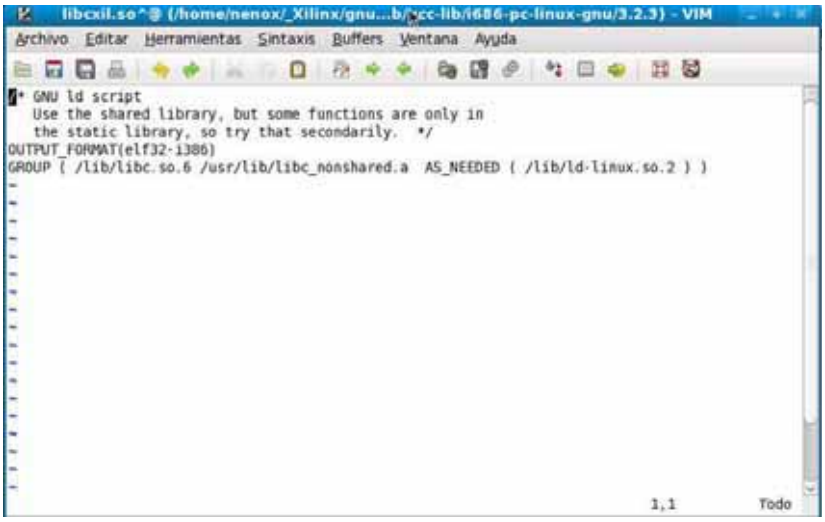


Figura F.23: Archivo libcxil.so original.

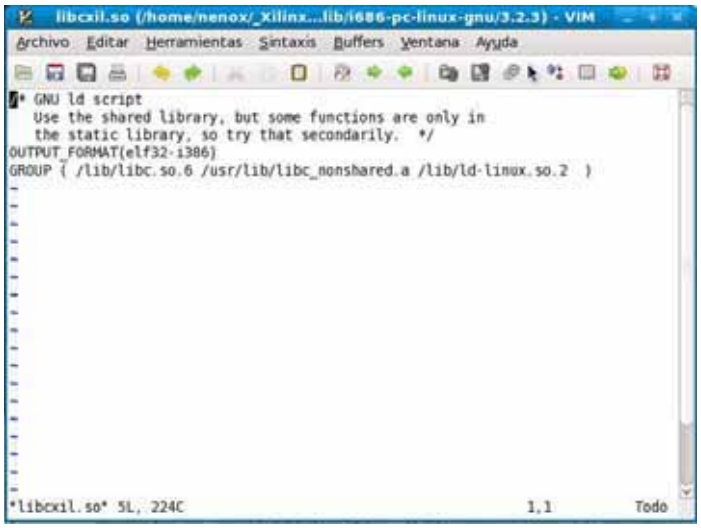


Figura F.24: Archivo libcxil.so modificado.

F.6.2. Problema con el módulo ethernet EDK

Este es un problema con la licencia del modulo *plb_ethernet* en el EDK, este problema sale al querer agregar el modulo en un proyecto. La solución que encontramos fue cambiar algunos parámetros de la duración de la licencia del modulo.

Los archivos de licencia se encuentran en la ruta RUTA_EDK/data/core-licenses por ahora solo tenemos que modificar el archivo plb_ethernet_v1_flexlm.lic que se encuentra en esa ruta. Los cambios al archivo se muestran en las siguientes imagenes:

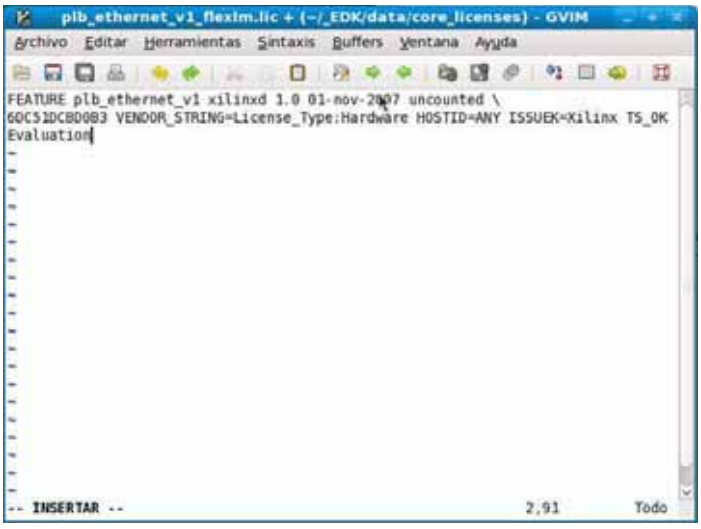


Figura F.25: Archivo de licencia original.

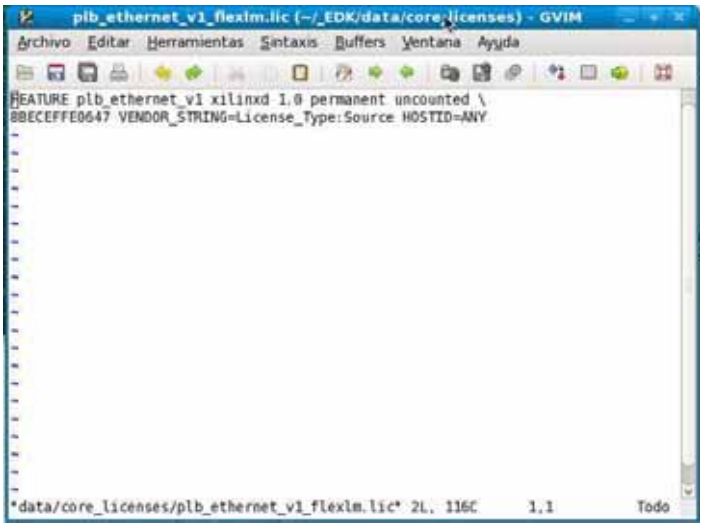


Figura F.26: Archivo de licencia modificado.

Apéndice G

Manual de Creación de un sistema mínimo

G.1. Objetivo

Esta práctica tiene como objetivo guiar en la creación de un sencillo sistema para el procesador PowerPC orientado a la tarjeta de desarrollo XUP Virtex-II pro, a través del Xilinx Platform Studio (XPS). Además de mostrar los pasos y soluciones a los posibles errores que se presentan al bajar el sistema a la tarjeta y verificar los resultados en la hyperteminal (minicom).

G.2. Preámbulo

Las practicas a realizar tendrán como objetivo hacer un sistema completo, por eso que cada práctica se basara en la practica anterior. El sistema completo se muestra en la siguiente figura y las partes a realizar en esta práctica se encuentran sombreadas.

En esta práctica, Utilizaremos el BSB de el XPS para crear un sistema de procesador que consiste en el siguiente procesador IP.

- PPC405 (Procesador PowerPC)
- Proc_Sys_Reset (circuito de sistema de reset)
- JTAG_PPC (interfaz al circuito interno JTAG del FPGA)
- DCM (Genera diversas frecuencias de reloj usadas por el procesador, buses, y periféricos)
- PLB bus (Bus de alto rendimiento)

- PLB BRAM controller (PLB controlador de memoria que se conecta a la memoria del FPGA)
- BRAM (Memoria del FPGA)
- PLB2OPB (Puente entre los buses PLB y OPB)
- OPB bus
- OPB UART Lite (Versión lite de UART, que esta incluida en la instalación de EDK)

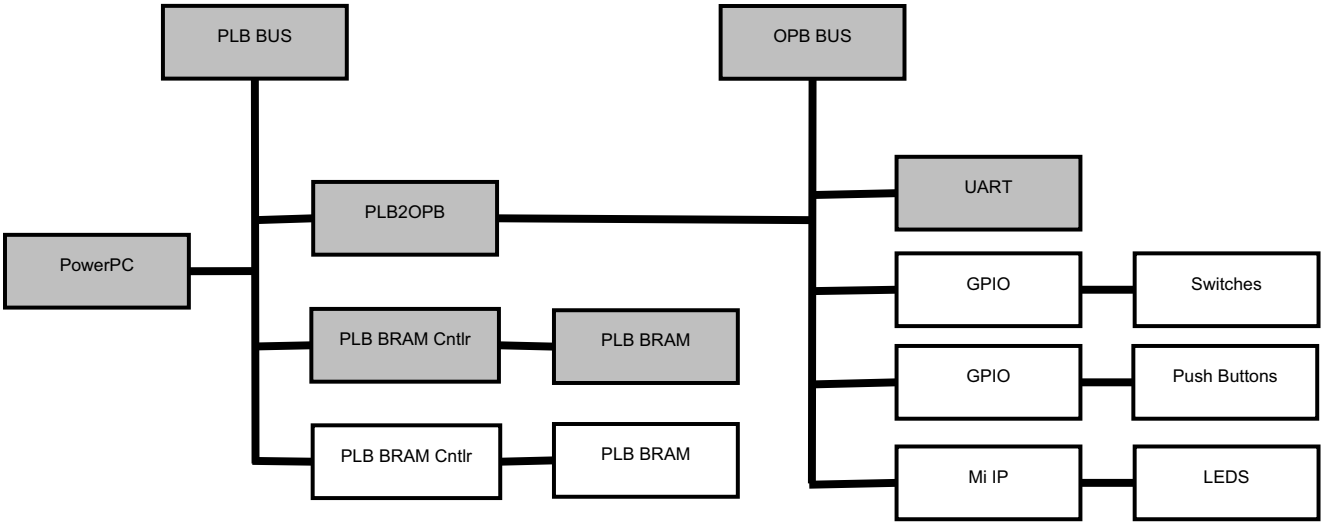


Figura G.1: Sistema Completo.

G.3. Procedimiento

La práctica consta de dos pasos: Crear el proyecto usando Base System Builder y Bajar el sistema a la tarjeta de desarrollo.

G.3.1. Preparar entorno

Para poder utilizar el XPS o ISE y bajar un programa a la tarjeta se tienen que hacer lo siguiente:

1. Cargar las variables de entorno
   - #source <RUTA_XILINX>/settings.sh
   - #source <RUTA_EDK>/settings.sh

2. Cargar la biblioteca del driver USB para que la pueda utilizar Xilinx
   ◦ #export LD_PRELOAD=<RUTA_USB-DIVER>/usb-driver/libusb-driver.so

3. Revisar que el firmware de la tarjeta se halla cargado correctamente
   ◦ Conectar la tarjeta (USB y RS232)
   ◦ Con #lsusb verificar que el firmware de la tarjeta sea: 03fd:**0008** Xilinx, Inc.
   ◦ Si el firmware es 03fd:**0009** Xilinx, Inc, como en la figura. Entonces tenemos que cargarlo correctamente.

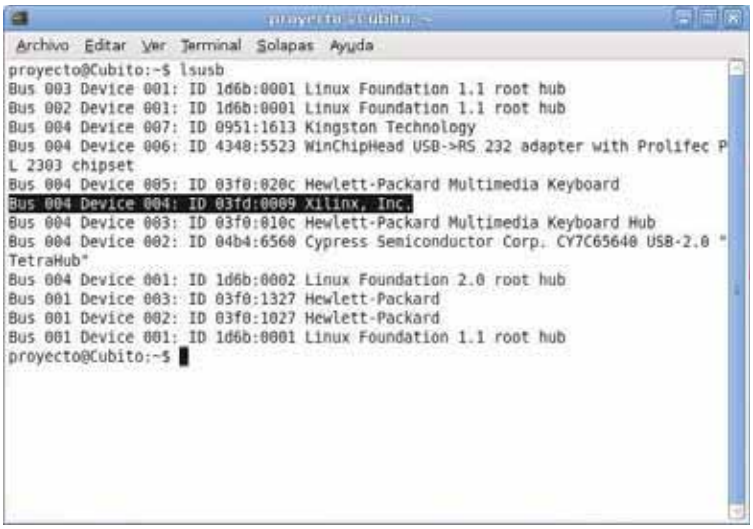


Figura G.2: Firmware mal cargado.

4. Para cargar el firmware correctamente
   ◦ #/sbin/fxload -v -t fx2 -I /usr/share/usb/xusbdfwu.hex -D /dev/bus/usb/***bus***/***dispositivo***
   Los parametros en negritas se pueden encontrar al ejecutar #lsusb.

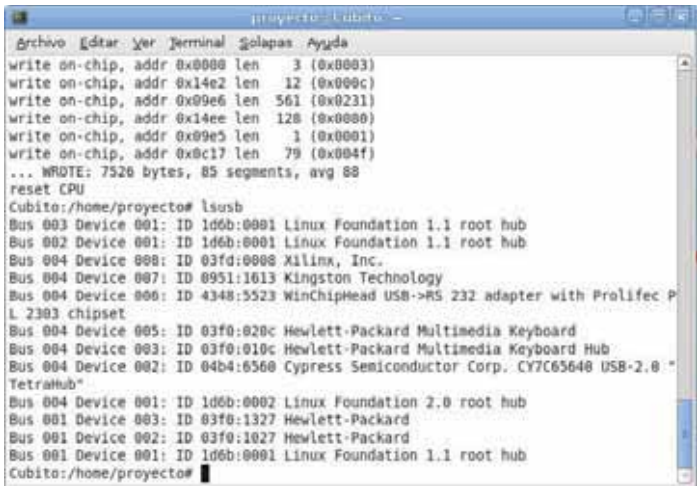


Figura G.3: Firmware cargado correctamente.

G.3.2. Paso 1: Crear Proyecto

1. Abrir la aplicación XPS:
   - #xps

2. Seleccionar Base System Builder Wizard y hacer clic en Ok.

Figura G.4: Creación de proyecto con Base System Builder Wizard.

3. Indicar la ruta en donde se guardara el proyecto, seleccionar use repository paths y dar clic en Ok. Los repository paths se localizan en el CD Virtex-II pro en la carpeta lib.

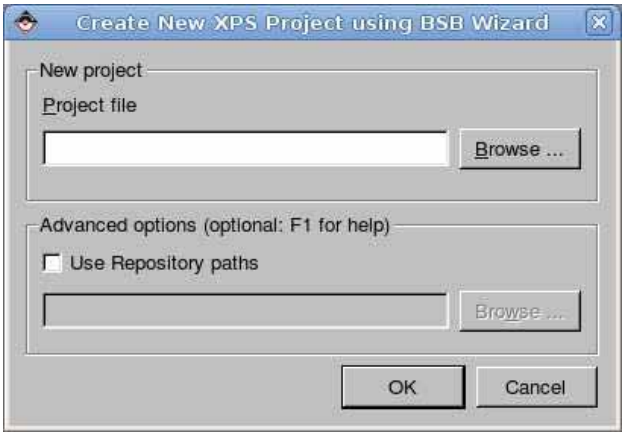


Figura G.5: Creación de proyecto con Base System Builder Wizard Dialog Box.

4. Seleccionar la opción I would like to create a new design y hacer clic en next.

Figura G.6: Creación de proyecto con Base System Builder Wizard.

5. seleccionar Board Vendor: Xilinx, Board Name: XUP Virtex-II Pro Development System, Board Revision: C y hacer clic en next.

Figura G.7: Select Board dialog box.

6. Seleccionar como procesador PowerPC y hacer clic en next en el cuadro de dialogo seleccionar procesador.

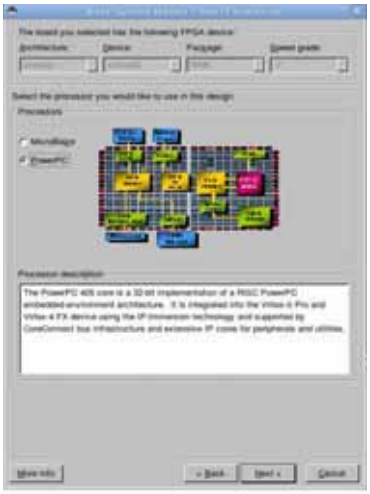


Figura G.8: Select Processor dialog box.

7. Especificar las opciones de configuración del procesador y dar clic en next.

   - Processor Clock Frequency: 100 MHz
   - Bus Clock Frequency: 100 MHz
   - JTAG Debug Interface: FPGA JTAG
   - On-Chip Memory (OCM) – Data: NONE
   - On-Chip Memory (OCM) – Instruction: NONE
   - Cache: Disabled

Figura G.9: Configure PowerPC dialog box.

8. En el cuadro de dialogo para configurar las interfaces IO
   - Seleccionar solo la opción RS232_Uart_1
   - Peripheral:OPB UARTLITE
   - Baudrate: 9600
   - Data bits: 8
   - Parity: None

   Hacer clic en next hasta llegar al cuadro de dialogo Add Internal Peripherals.

Figura G.10: Configure Additional IO Interfaces dialog box.

9. Seleccionar 16KB como tamaño de memoria y hacer clic en next.

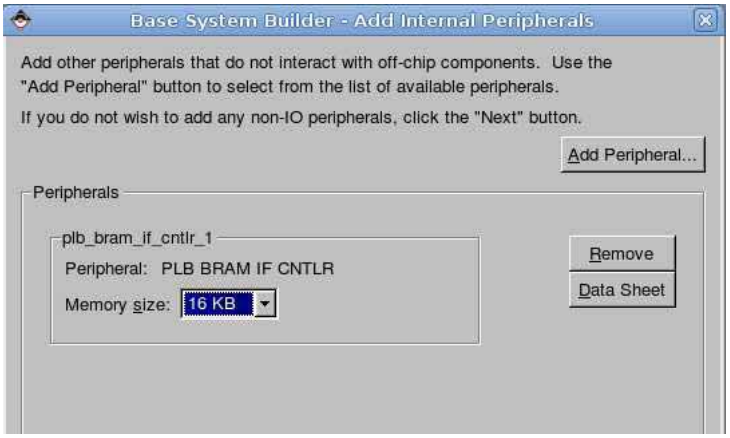


Figura G.11: Add Internal peripherals dialog box.

10. Seleccionar RS232_Uart_1 como STDIN y STDOUT, habilitar Memory test y dar clic en next.

Figura G.12: Software setup dialog box.

11. Hacer clic en next para pasar al cuadro de dialogo System Created, en el cual se muestra un resumen del sistema que ha sido creado.

Figura G.13: Configure memory test application dialog box.

12. Hacer clic en Generate para crear el proyecto.

Figura G.14: System Created dialog box.

13. Aparece un cuadro de dialogo que nos indica que el sistema ha sido generado con exito, hacer clic en finish.

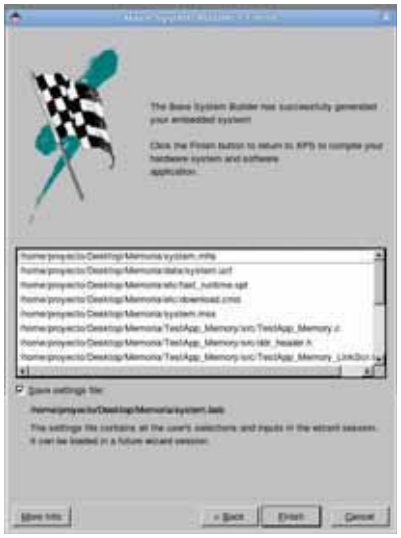


Figura G.15: Finish dialog box.

14. Hacer clic en start using Platform Studio.

Figura G.16: The next step dialog box.

G.3.3. Paso 2: Bajar la aplicación

1. En la pestaña Applications, hacer clic derecho en TestApp_Peripheral software project y seleccionar "Mark to Inicialize BRAM".

2. Hacer clic derecho en TestApp_Memory software project y desmarcar "Mark to Inicialize BRAM".

3. Seleccionar Device Configuration ⟶ Update Bitstream.

   Nota: Esto generara las siguientes acciones: ejecutara el platform generator → generara el bitstream → generara las bibliotecas → compilara el codigo del SW → mezclara el ejecutable con el bitstream.

4. Abrir la hyperteminal para ver la salida de la aplicación.

   a) Como la salida de nuestra aplicación será por el puerto serie (RS232) debemos asegurarnos que la maquina vea el cable, para esto utilizamos desde una terminal $ls /dev el cable es el ttyUSBN donde N es el numero asignado por el sistema.

   b) Para abrir la hyperteminal, en nuestro caso minicom ejecutamos: #minicom -s, lo que nos mostrara lo siguiente:

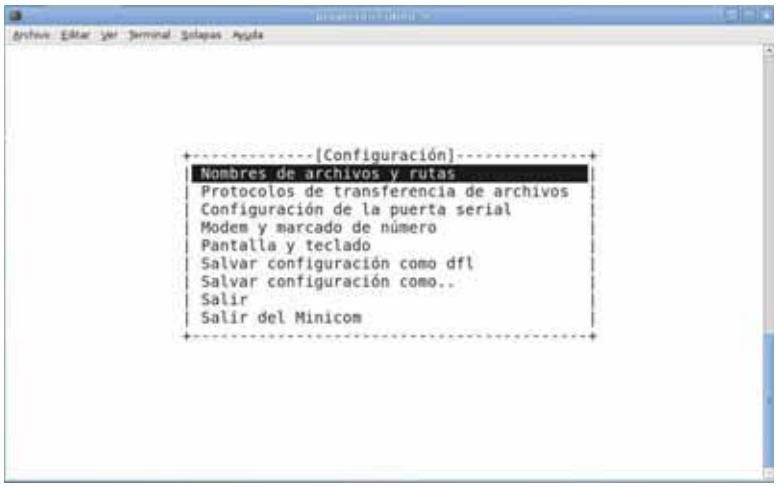


Figura G.17: Pantalla de configuracion de minicom.

c) Seleccionamos la opción de configurar la puerta serial y nos muestra:

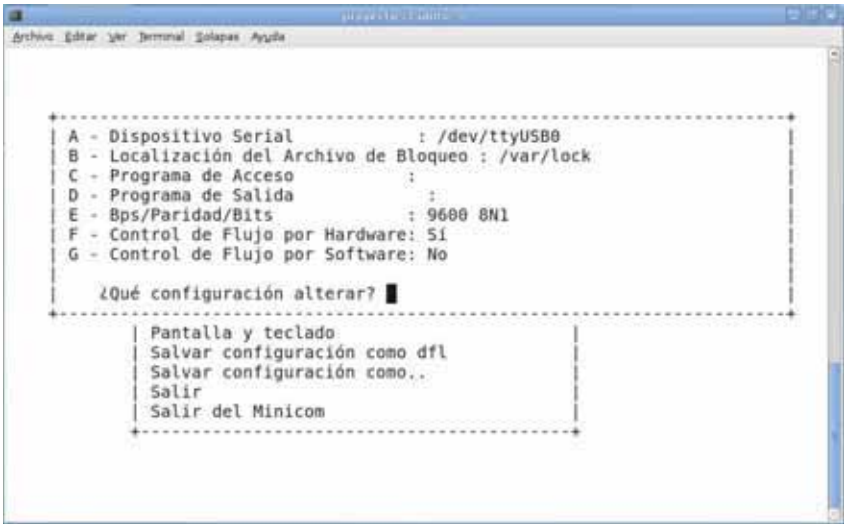


Figura G.18: Pantalla de configuracion de la puerta serial.

- Pulsamos la tecla A, ponemos /dev/ttyUSB0 y pulsamos enter.
- Pulsamos E, ponemos 9600 8N1 y pulsamos enter.
- Damos enter y despues seleccionamos la opcion salir.

5. Seleccionamos Device Configuration ⟶ Download Bitstream y en la terminal deberá aparecer:

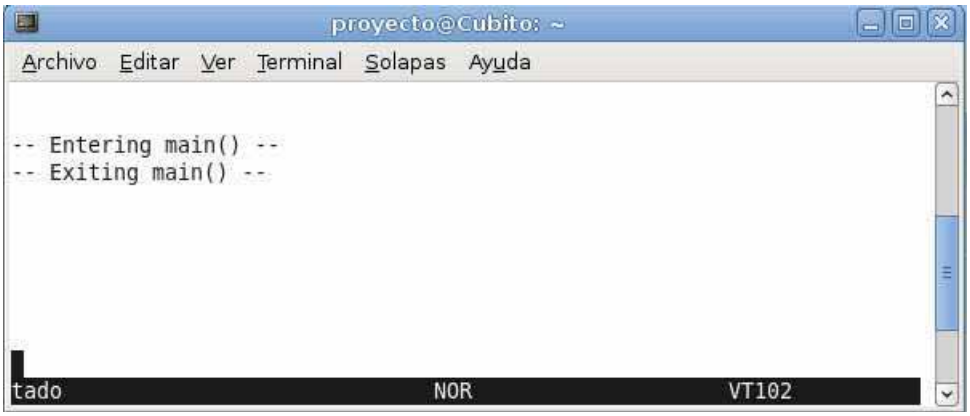


Figura G.19: Salida en minicom.

G.4. Posibles errores al bajar la aplicación

Es posible que salgan algunos errores al bajar la aplicación y aquí trataremos de explicar cuales son y cuales son las soluciones.

Uno de los errores es al hacer Download Bitstream, podria aparecer: USB_Transfer: -110 (error sending control messaje: connection time out).

Este error se soluciona pulsando el boton de reset de la tarjeta y volviendo a cargar el firmware como se explico anteriormente.

Otro de los errores es que en la ventana de salida del xps aparezca lo siguiente:

Figura G.20: Error al bajar la aplicación.

La solucion a este error es, en una terminal lanzamos los settings como se hizo en el paso 1 y 2 del apartado preparar el entorno y ejecutamos:

- #impact -batch
- >setMode -bscan
- >cleancablelock

Esta solución sirve también cuando aparece Reussing ###### Key.

Apéndice H

Manual para Agregar IP a un Diseño de Hardware

H.1. Objetivo

El objetivo de la practica es extender el diseño de hardware de la práctica 1, mostrando dos métodos para agregar IPs adicionales, ya sea, Usando el IP Catalog o modificando el archivo MHS con un editor de textos e Implementar el diseño utilizando Xflow.

H.2. Preámbulo

En la práctica 1 se incluyeron: procesador PowerPC, bus PLB, JTAG_PPC, proc_Sys_Reset, DCM, PLB2OPB, UART, PLB RAM controller, PLB BRAM y un componente OPB UART lite. En esta práctica agregaremos los mismos componentes excepto por una instancia para los LEDs llamada MYIP, para extender el diseño.

En la práctica, usaremos el modo de dialogo del sistema XPS y el modo texto para agregar los siguientes IPs al sistema de procesador existente.

- PLB_BRAM_CNTLR para datos.
- BRAM para datos.
- OPB GPIO para Push button y Dip switches.

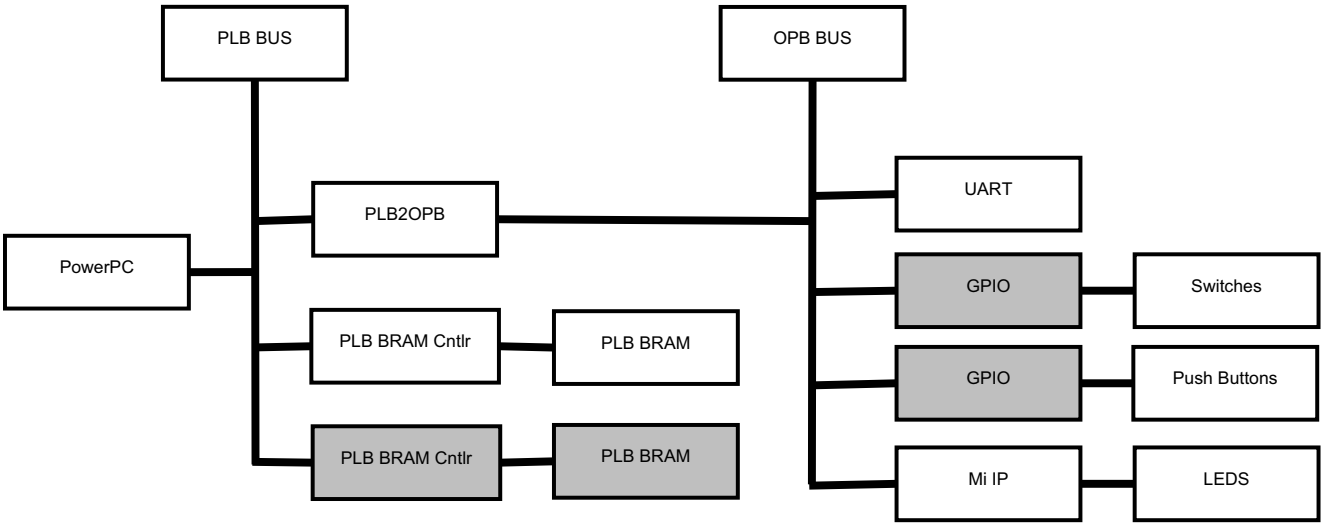


Figura H.1: Sistema Completo.

H.3. Procedimiento

La práctica consta de varios pasos, incluyendo agregar IP al sistema de procesador diseñado en la práctica 1 y crear un proyecto ISE usado para implementar el diseño.

NOTA: Recuerda que antes de realizar la práctica debes preparar el entorno como se hizo en la práctica 1.

H.3.1. Paso 1: Abrir el Proyecto

Como vamos a utilizar como base el sistema creado en el manual entonces debemos:

1. Crear un directorio llamado Practica_02.
2. Copiar el contenido del directorio de la práctica 1 dentro del directorio Practica_02.
3. Abrimos XPS.
    - #xps
4. Seleccionar Open A Recent Project, clic en Ok y buscar el directorio de la práctica 2.

Figura H.2: Ventana de opciones de proyecto.

5. Seleccionar system.xmp para abrir el proyecto.

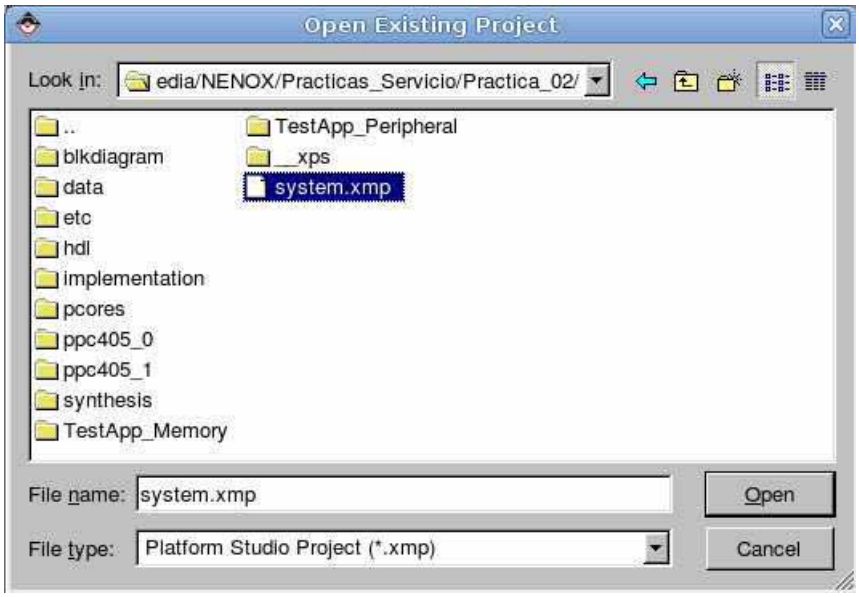


Figura H.3: Ventana open Existing Project.

H.3.2. Paso 2: Extender el Hardware del Sistema

Agregar los siguientes IP al sistema de procesador via IP Catalog:

- PLB BRAM interface controler.
- BRAM.
- OPB_GPIO (dos instancias).

1. Seleccionar el IP Catalog, que muestra una lista de todos los IP disponibles para uso en el sistema.

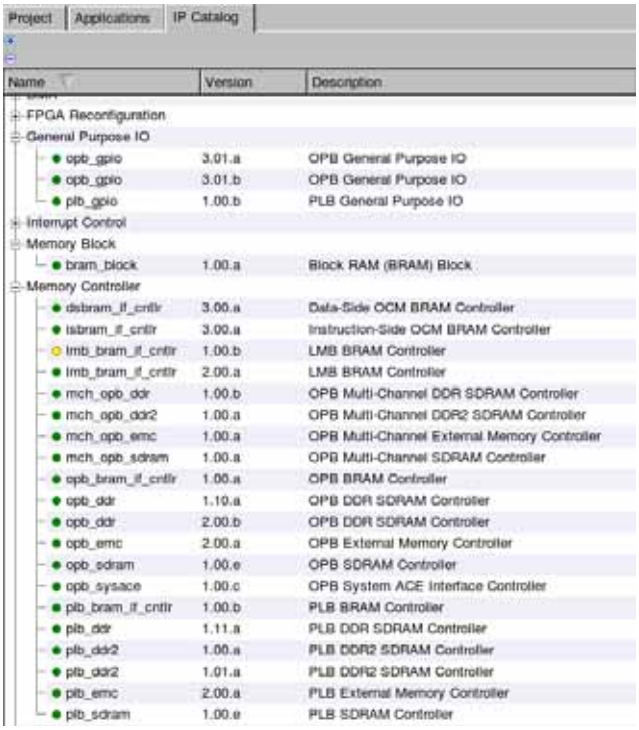


Figura H.4: IP Catalog.

2. Agregar los siguientes periféricos al diseño:
   - plb_bram_if_cntlr (1.00.b)
   - bram_block (1.00.a)
   - opb_gpio (3.01.b) – agregar dos instancias

3. Cambiar los nombres de las instancias de los periféricos que se acaban de agregar de acuerdo a la tabla de abajo.

Nombre por defecto	**Nuevo nombre**
opb_gpio_0	dip1
opb_gpio_1	push1
plb_bram_if_cntlr_0	plb_bram_if_cntlr_2
bram_block_0	plb_bram_if_cntlr_2_bram

Tabla H.1: Tabla de nombres.

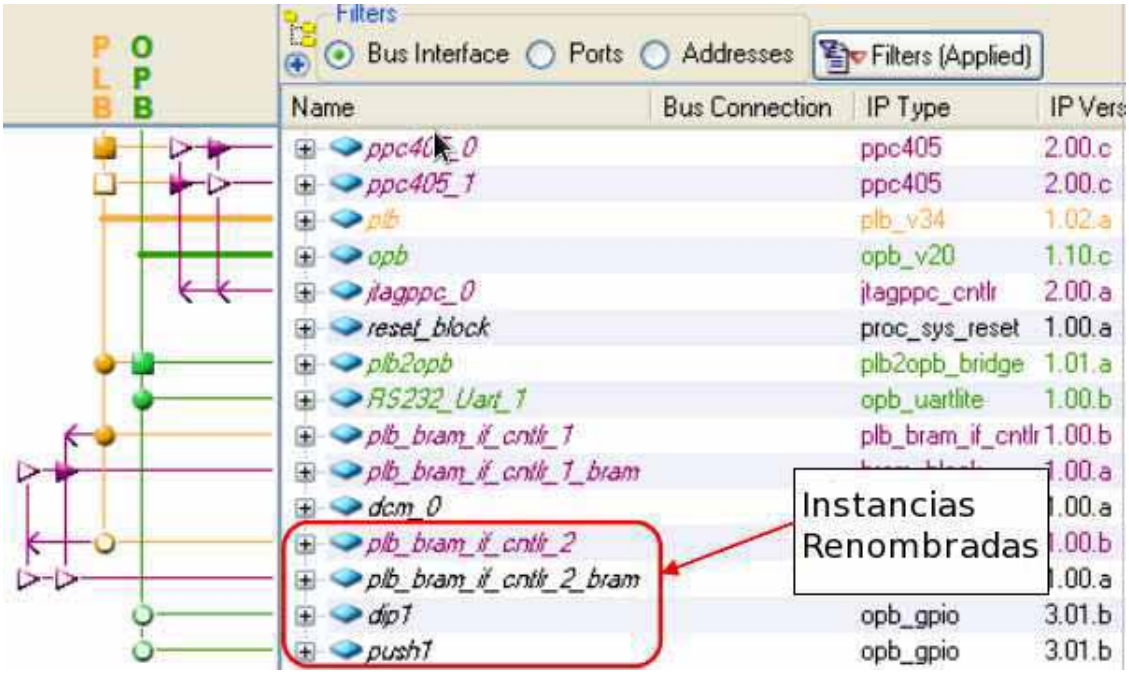


Figura H.5: Nombres cambiados.

4. Establecer las conexiones de bus como se listan en la siguiente tabla, Expandiendo cada periférico y seleccionar la apropiada conexión de bus.

Nombre de la instancia	**plb**	**opb**
plb_bram_if_cntlr_2	•	
dip1		•
push1		•

Tabla H.2: Conexiones de bus de los periféricos.

NOTA:El dispositivo dip_push es un dispositivo esclavo conectado a el bus OPB. El plb_bram contrler es un dispositivo esclavo conectado a el bus PLB.

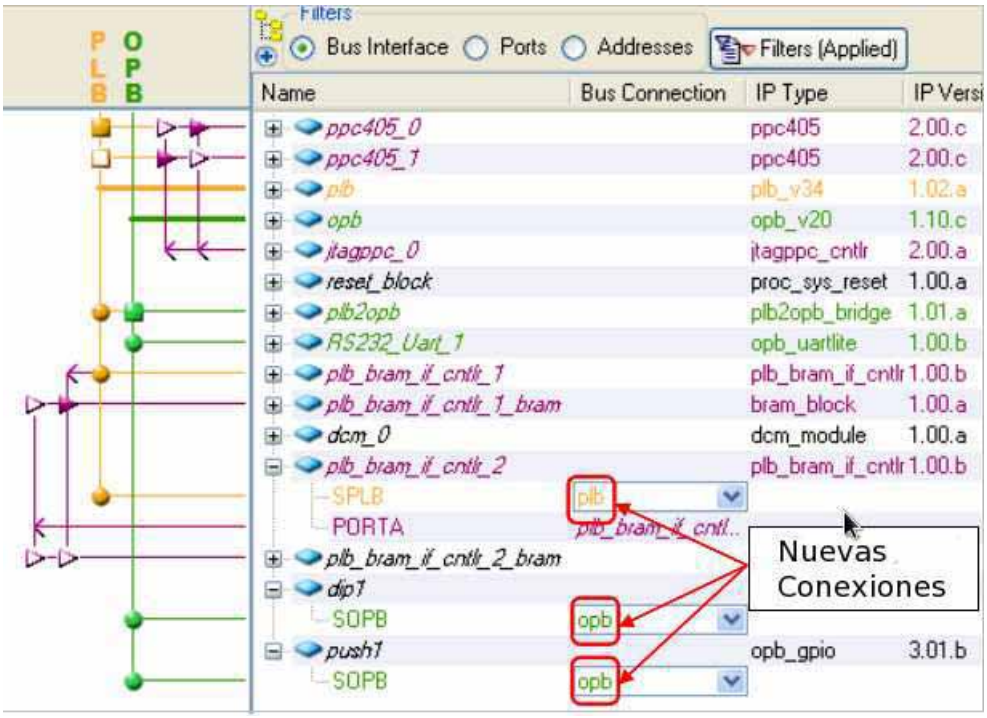


Figura H.6: Conexiones Asignadas.

5. Conecta el PORT A de el control de memoria plb_bram_if_cntlr_2 a el PORT A de la memoria BRAM, selecciona bus PORT A bajo la columna de conexion de la instancia de plb_bram_if_cntlr_2_bram y selecciona plb_bram_if_cntlr_2_PORTA.

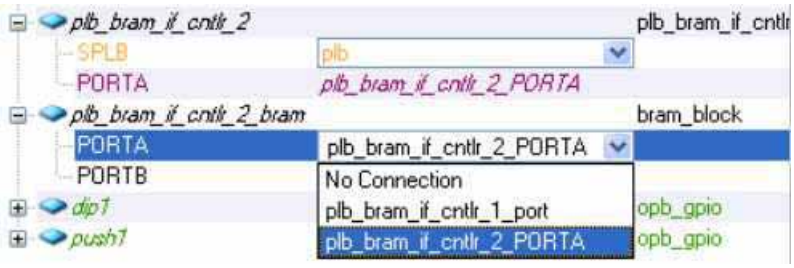


Figura H.7: Conexión entre el bloque de RAM y el control de memoria.

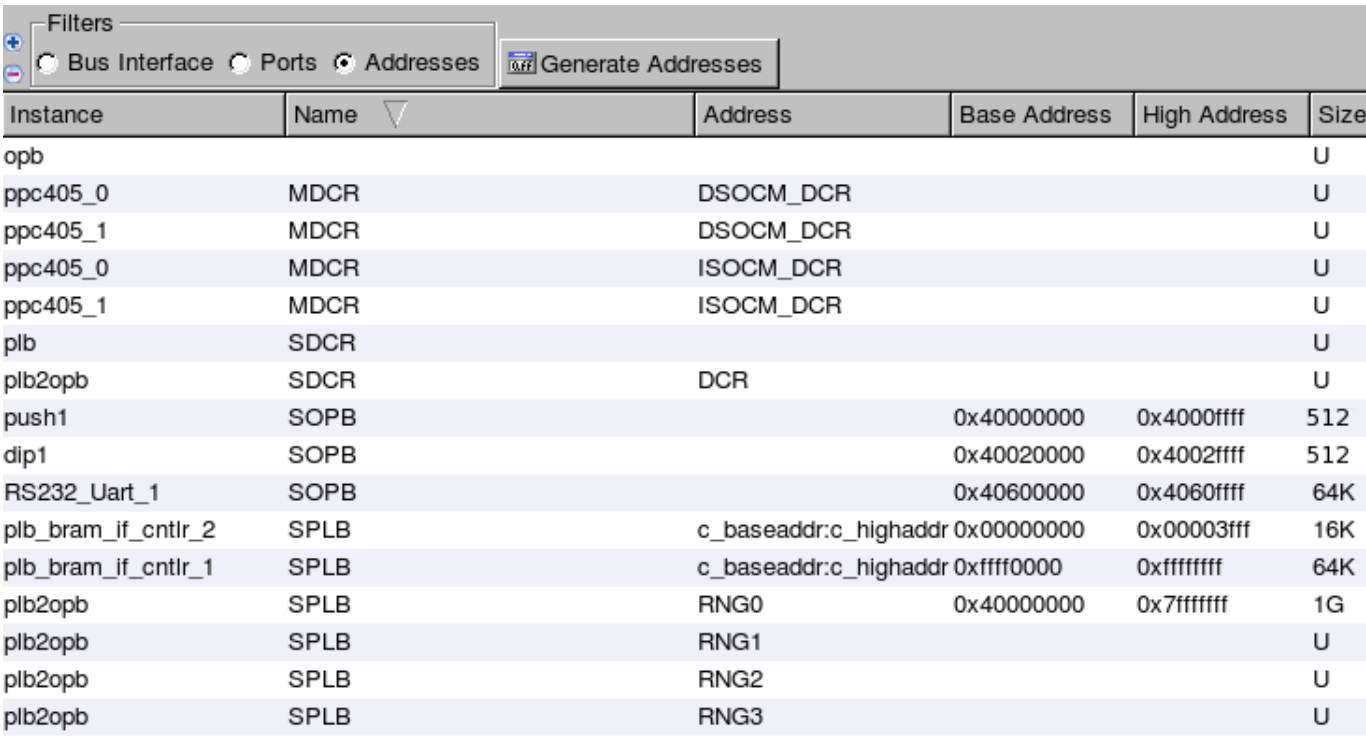
6. Renombra el conector entre el PORT A de el control de memoria plb_bram_if_cntlr_1_PORTA y el PORT A de el bloque de RAM plb_bram_if_cntlr_1_bram como plb_bram_if_cntlr_1_PORTA y selecciona No Connection en el PORT B.

7. Selecciona la pestaña de Addresses e introduce los tamaños y la dirección base para los periféricos de acuerdo a la tabla de abajo.

Nombre de la instancia	C_BASEADDR	Tamaño
plb_bram_if_cntlr_2	0x00000000	16K
dip1	0x40700000	512
push1	0x40900000	512

Tabla H.3: Mapa de Memoria de los Perifericos.

NOTA: Puedes asignar manualmente la dirección base y la dirección mas alta de tus periféricos o dejar que XPS lo haga haciendo clic en el botón Generate Addresses.



Instance	Name	Address	Base Address	High Address	Size
opb					U
ppc405_0	MDCR	DSOCM_DCR			U
ppc405_1	MDCR	DSOCM_DCR			U
ppc405_0	MDCR	ISOCM_DCR			U
ppc405_1	MDCR	ISOCM_DCR			U
plb	SDCR				U
plb2opb	SDCR	DCR			U
push1	SOPB		0x40000000	0x4000ffff	512
dip1	SOPB		0x40020000	0x4002ffff	512
RS232_Uart_1	SOPB		0x40600000	0x4060ffff	64K
plb_bram_if_cntlr_2	SPLB	c_baseaddr:c_highaddr	0x00000000	0x00003fff	16K
plb_bram_if_cntlr_1	SPLB	c_baseaddr:c_highaddr	0xffff0000	0xffffffff	64K
plb2opb	SPLB	RNG0	0x40000000	0x7fffffff	1G
plb2opb	SPLB	RNG1			U
plb2opb	SPLB	RNG2			U
plb2opb	SPLB	RNG3			U

Figura H.8: Asignación con Generate Addresses.

8. Seleccionar Project ⟶ Generate and View Block Diagram, para ver el diagrama de bloques del diseño construido.

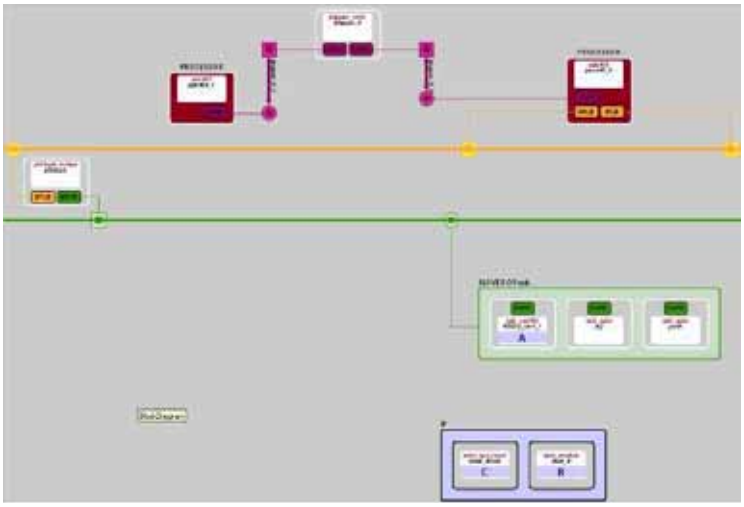


Figura H.9: Vista de Diagrama de Bloques despues de agregar los periféricos.

9. Cierra la vista de diagrama de bloques.

Configurar los periféricos recién agregados.

1. Hacer doble-clic en la instancia dip1 y poner los parámetros de acuerdo a las siguientes figuras.

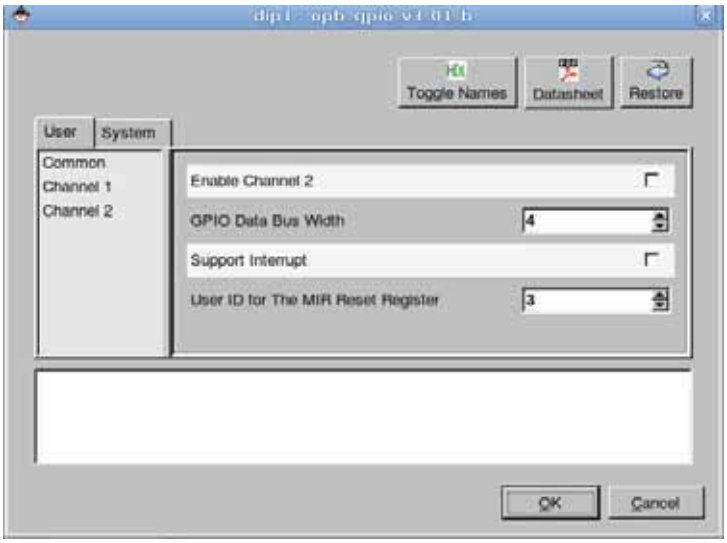


Figura H.10: Parámetros comunes para la instancia dip1.

Figura H.11: Parámetros de canal 1 para la instancia dip1.

2. Hacer doble-clic en la instancia push1 y poner los parámetros de acuerdo a las siguientes figuras.

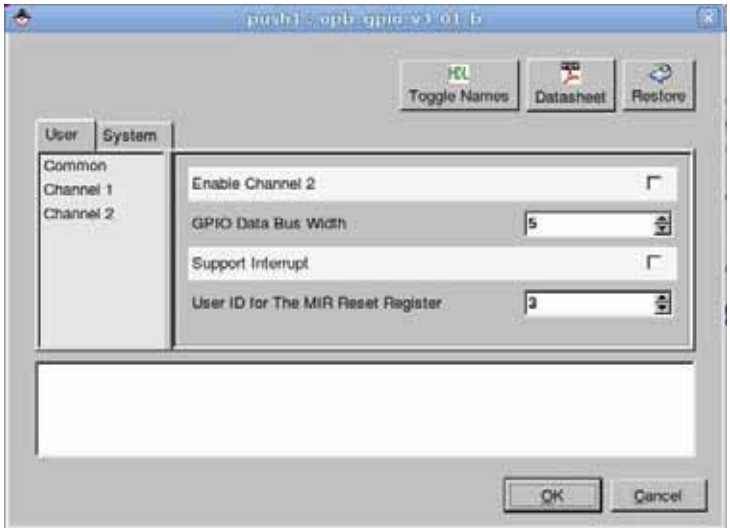
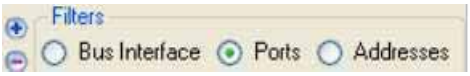


Figura H.12: Parámetros comunes para la instancia push1.

Figura H.13: Parámetros de canal 1 para la instancia push1.

Agregar puertos adicionales a los periféricos y conectarlos. Esos puertos coinciden con los pins externos del FPGA que lo conectan a los componentes de la tarjeta incluyendo los push buttons y los DIP switches.

1. Seleccionar el filtro de puertos en el System Assembly View.



2. Expandir dip1 y push1 para ver los puertos disponibles.

3. Cambiar las señales de conexión escribiendo los nuevos valores de acuerdo a la tabla de abajo.

Nombre de la instancia	**Puerto**	**Conexión**
dip1	GPIO_in	DIP
push1	GPIO_in	PUSH

Tabla H.4: Puertos adicionales que se agregaron a los periféricos.

4. Después de entrar en las conexiones para los puertos como se describe en el paso anterior, elegir la opción Make External de la lista para hacer las mismas conexiones externas (conexiones hacia los pines de I/O de el FPGA).

5. En External Ports Connections, cambiar los nombres de los puertos, los net names, polarity, y rangos como se muestra en la figura de abajo. Borrar los pines externos que están puestos a tierra (net_gnd).

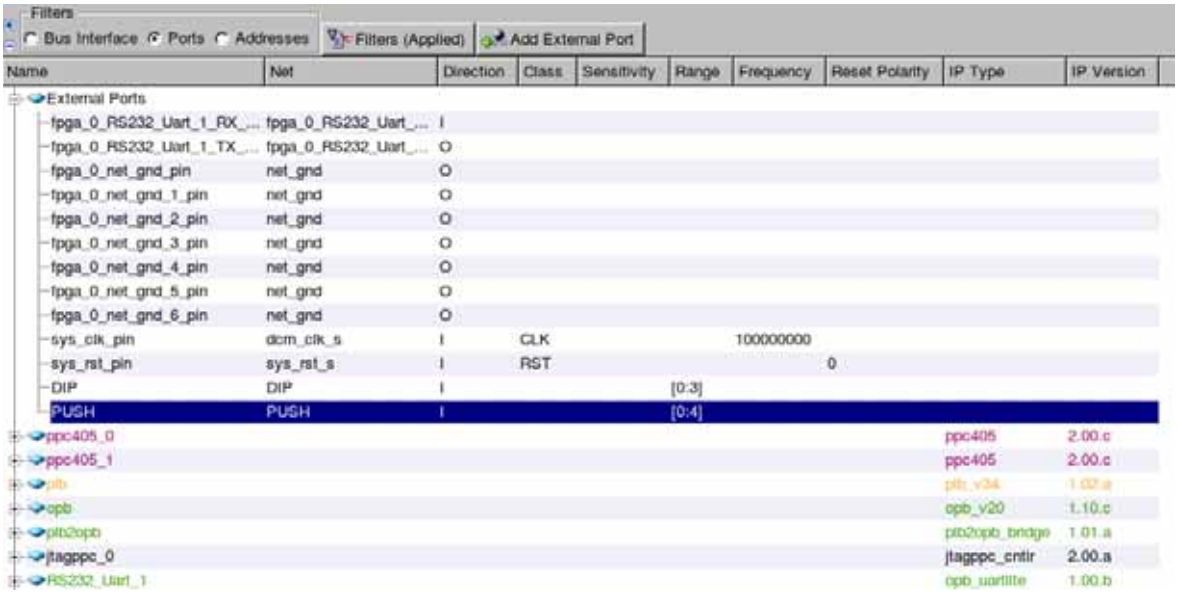


Figura H.14: Filtro de puertos despues de hacer dip_push GPIO externos.

Agregar el codigo que implementa la funcionalidad de los push buttons y de los dip switches.

1. Hacer clic en la pestaña de Applications y hacer doble-clic en el archivo TestApp_Memory.c.

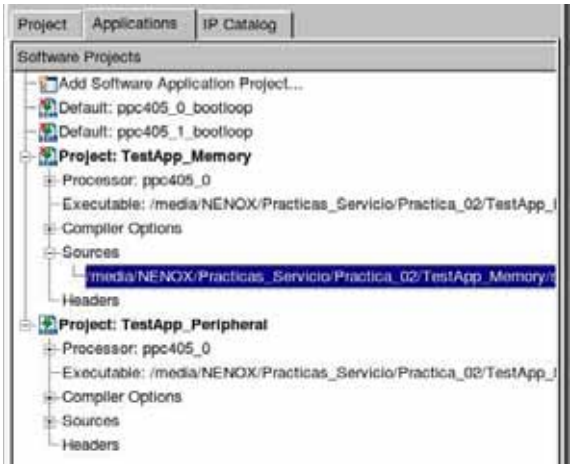


Figura H.15: Pestaña de application que contiene el código fuente.

2. Abrir el archivo TestApp_Memory.c y agregar el siguiente código:

```c
#include "xparameters.h"
#include "xgpio.h"
#include "stdio.h"

#include "xutil.h"

//----------------------------------------------------

int main (void) {

        XGpio dip,push;
        int psb_check,dip_check;

        print("-- Start of the program --\r\n");

        XGpio_Initialize(&dip, XPAR_DIP1_DEVICE_ID);
        XGpio_SetDataDirection(&dip,1,0xffffffff);

        XGpio_Initialize(&push, XPAR_PUSH1_DEVICE_ID);
        XGpio_SetDataDirection(&push,1,0xffffffff);

        while(1){
                psb_check = XGpio_DiscreteRead(&push,1);
                xil_printf("push buttons status %x\r\n",psb_check);
                dip_check = XGpio_DiscreteRead(&dip,1);
                xil_printf("DIP switch status %x\r\n",dip_check);
                sleep(1);
        }
}
```

Figura H.16: Código fuente de la aplicación.

3. Hacer clic en el menú Software ⟶ Generate Libraries, lo que generara el archivo xparameters.h.

4. Hacer clic en el archivo system.ucf que se encuentra en la pestaña Project y agregar el siguiente código para asignar pins a push buttons y dip switches. También Actializa el IOSTANDARD de sys_rst_pin con LVCMOS25.

```
Net sys_clk_pin LOC=AJ15;
Net sys_clk_pin IOSTANDARD = LVCMOS25;
Net sys_rst_pin LOC=AH5;
Net sys_rst_pin IOSTANDARD = LVCMOS25;
## System level constraints
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 10000 ps;
Net sys_rst_pin TIG;
NET "C405RSTCORERESETREQ" TPTHRU = "RST_GRP";
NET "C405RSTCHIPRESETREQ" TPTHRU = "RST_GRP";
NET "C405RSTSYSRESETREQ" TPTHRU = "RST_GRP";
TIMESPEC "TS_RST1" = FROM CPUS THRU RST_GRP TO FFS  TIG;

## IO Devices constraints

#### Module RS232_Uart_1 constraints

Net fpga_0_RS232_Uart_1_RX_pin LOC=AJ8;
Net fpga_0_RS232_Uart_1_RX_pin IOSTANDARD = LVCMOS25;
Net fpga_0_RS232_Uart_1_TX_pin LOC=AE7;
Net fpga_0_RS232_Uart_1_TX_pin IOSTANDARD = LVCMOS25;
Net fpga_0_RS232_Uart_1_TX_pin SLEW = SLOW;
Net fpga_0_RS232_Uart_1_TX_pin DRIVE = 12;

net push<0> LOC = AH4;
net push<1> LOC = AH1;
net push<2> LOC = AG5;
net push<3> LOC = AH2;
net push<4> LOC = AG3;

net dip<0> LOC = AC11;
net dip<1> LOC = AD11;
net dip<2> LOC = AF8;
net dip<3> LOC = AF9;
```

Figura H.17: Código fuente del archivo system.ucf después de asignar pins.

5. Compila el código fuente, haciendo clic en el icono.

H.3.3. Paso 3: Bajar el Bitstream

Configurar y abrir una hiperterminal. Generar el bitstream y bajarlo vía EDK, y verifica la operación en la tarjeta.

1. Seleccionar Device Configuration ⟶ Update Bitstream.

   Nota: Esto generara las siguientes acciones: ejecutara el platform generator → generara el bitstream → generara las bibliotecas → compilara el codigo del SW → mezclara el ejecutable con el bitstream.

2. Abrir la hyperteminal para ver la salida de la aplicación.

   a) Como la salida de nuestra aplicación será por el puerto serie (RS232) debemos asegurarnos que la maquina vea el cable, para esto utilizamos desde una terminal $ls /dev el cable es el ttyUSBN donde N es el numero asignado por el sistema.

   b) Para abrir la hyperteminal, en nuestro caso minicom ejecutamos: #minicom -s, lo que nos mostrara lo siguiente:

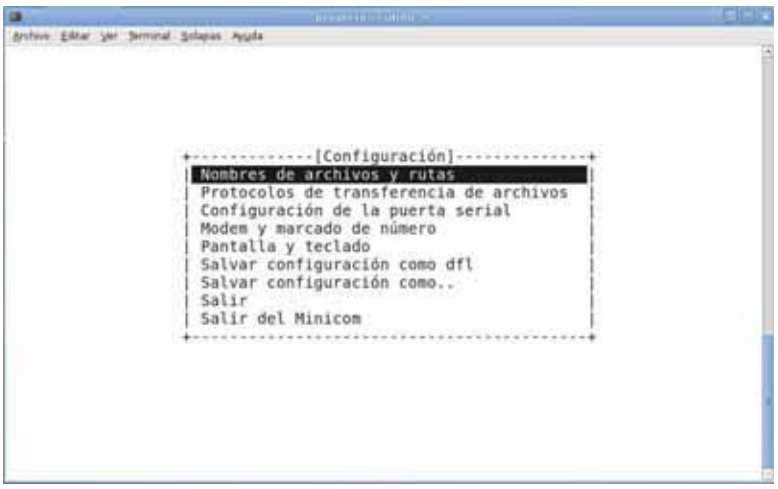


Figura H.18: Pantalla de configuracion de minicom.

   c) Seleccionamos la opción de configurar la puerta serial y nos muestra:

   - Pulsamos la tecla A, ponemos /dev/ttyUSB0 y pulsamos enter.
   - Pulsamos E, ponemos 9600 8N1 y pulsamos enter.

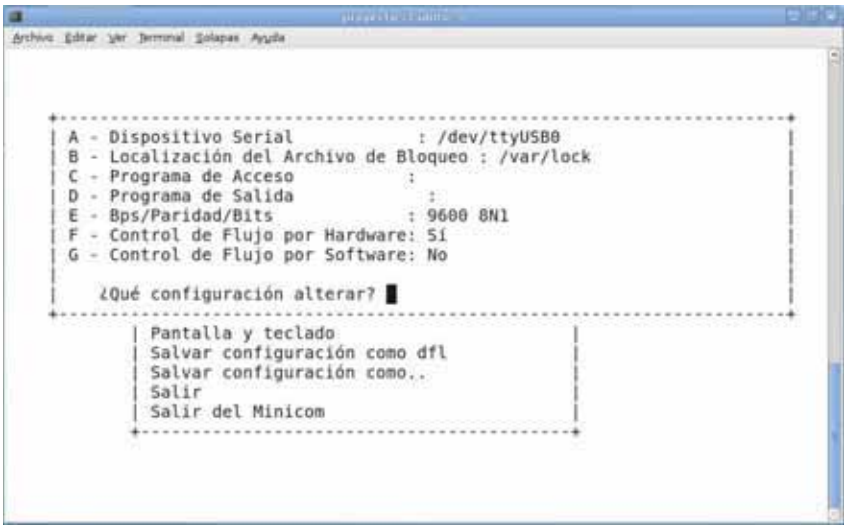


Figura H.19: Pantalla de configuracion de la puerta serial.

- Damos enter y despues seleccionamos la opcion salir.

3. Seleccionamos Device Configuration ⟶ Download Bitstream y en la terminal deberá aparecer:

Figura H.20: Salida en minicom.

4. Presiona los botones y los switches y nota los cambios de valores en la hiperterminal.

Apéndice I

Manual Agregar un IP Personalizado a un Diseño de Hardware

I.1. Objetivo

El objetivo de esta práctica es mostrar los pasos para crear y agregar un periférico OPB personalizado al sistema embebido que se actualizo en la práctica 2 usando el Create/Import Peripheral Wizard. Actualizar el código de la practica 2 para escribir a este nuevo periférico, así como descargarlo y probarlo en la tarjeta.

I.2. Preámbulo

Con esta práctica se pretende completar el diseño de hardware empezado en la práctica 1 y extendido en la práctica 2. En la práctica 1 se incluyeron los componentes: PPC, PLB bus, JTAG_PPC, proc_Sys_Reset, DCM, PLB2OPB, RS232_Uart_1, PLB RAM controller, and PLB BRAM. En la práctica 2 utilizaron los mismos IP, con excepción de una instancia MYIP para los LEDs, para extender el diseño de hardware.

En esta práctica, usaremos el Create and Import Peripheral Wizard de Xilinx Platform Studio (XPS) para crear un periférico de usuario para un modulo HDL, agregar una instancia de el periférico importado, y modificar el archivo system.ucf para proveer una interfaz para el modulo de LED.

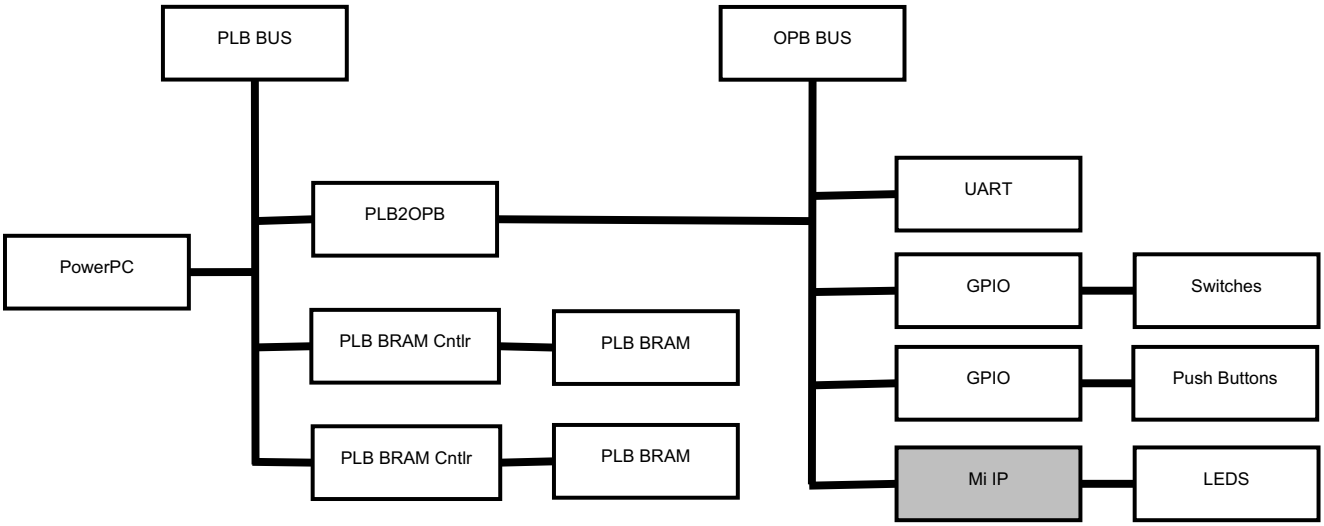


Figura I.1: Sistema Completo.

I.3. Procedimiento

Esta practica comprende varios pasos que implican la creación de un OPB IP personalizado (una simple salida 4-bit para manejar los LEDs) y la adición de un periférico OPB personalizado. Aunque el cambio en el hardware es sencillo, en esta práctica se ilustra la integración de un periférico de usuario a través del Create and Import Peripheral Wizard. Esta práctica también ilustra el uso de un periférico existente para proveer una interfaz al bus OPB.

I.3.1. Paso 1: Creación de un IP Personalizado

Como tomaremos como base el trabajo realizado en la práctica 2, entonces:

1. Crear un directorio llamado Practica_03.
2. Copiar el contenido del directorio Practica_02 al directorio creado.
3. Abrir Xilinx Platform Studio (XPS).
    - #xps
4. Hacer clic en Cancel.
5. Hecr clic en Hardware $\longrightarrow$ Create or Import Peripheral Wizard, se abrirá la siguiente ventana, se da clic en Next:

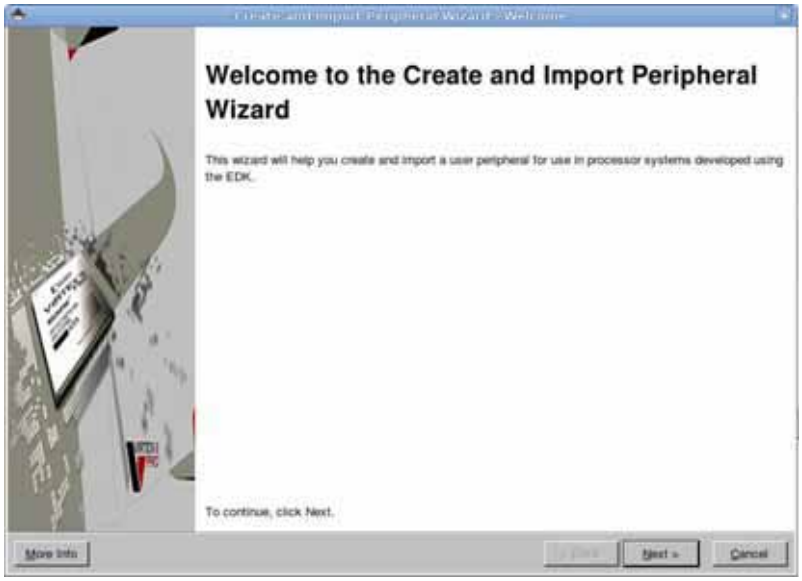


Figura I.2: Create and Import Peripheral Wizard-Welcome.

6. En la sección Select Flow, elegir Create templates for a new peripheral y haz clic en Next.

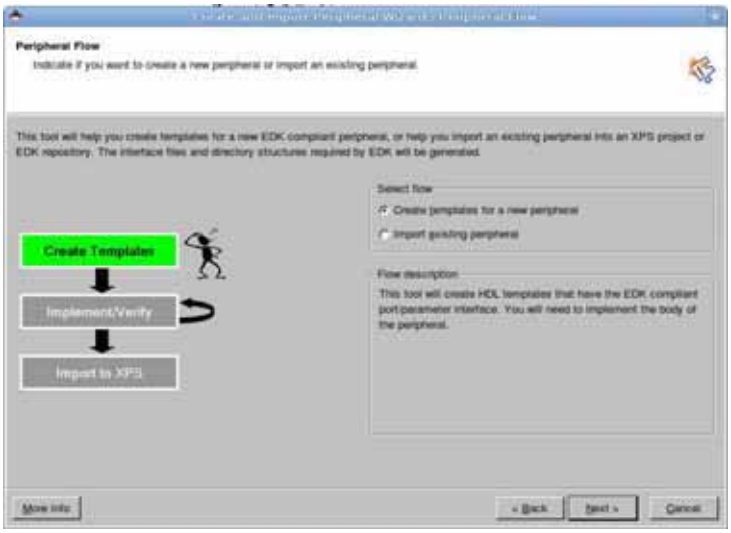


Figura I.3: Create and Import Peripheral Wizard-Peripheral Flow.

7. En el panel Repository or Project, elegir To an XPS Project, seleccionar el directorio de proyecto que seria Practica_03 y hacer clic en Next.

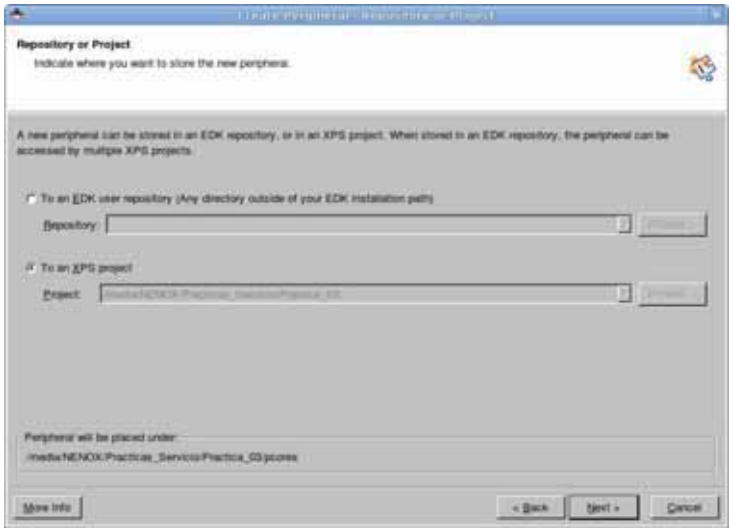


Figura I.4: Create Peripheral-Repository or Project.

8. En la sección Name and Version, como nombre de periférico poner my_led, con la versión por defecto y haz clic en Next.

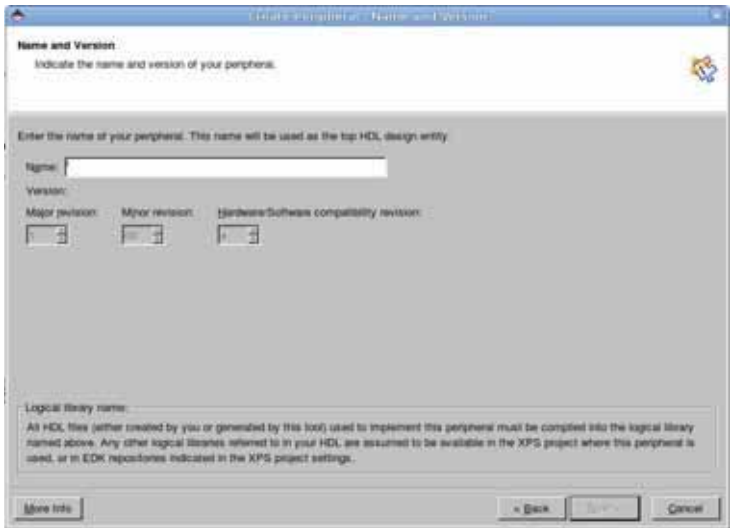


Figura I.5: Create Peripheral-Name and Version.

9. En la sección Bus Interface, Elegir On-chip Peripheral Bus (OPB), y hacer clic en Next.

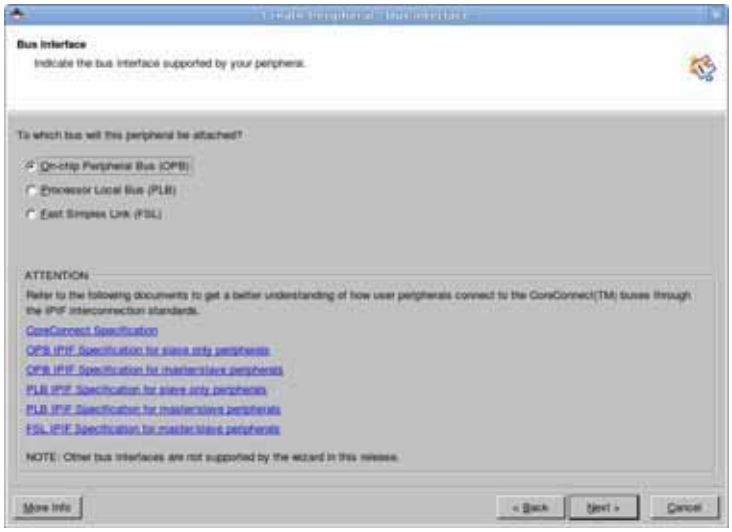


Figura I.6: Create Peripheral-Bus Interface.

10. En la seccón IPIF Services, Elige S/W Reset and MIR y User Logic S/W Register Support, Hacer clic en Next.

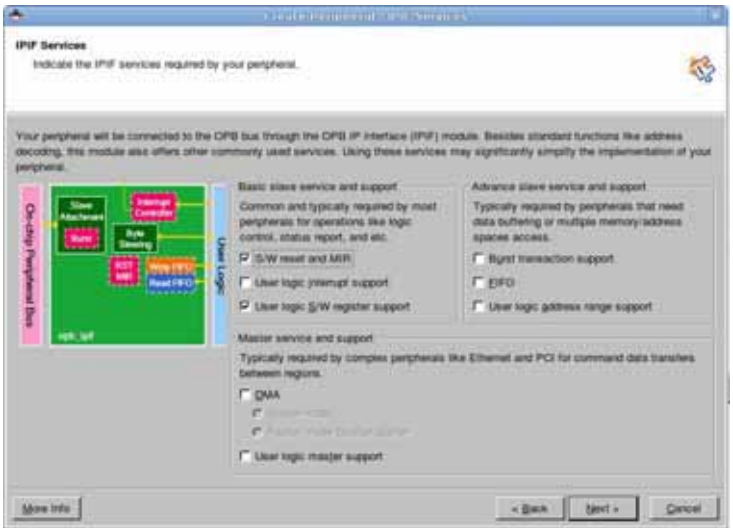


Figura I.7: Create Peripheral-IPIF Services.

11. En la sección User S/W Register, elige la segunda opción (Disable posted write behavior for normal acknowledged write behavior) y haz clic en Next dejendo las otras opciones con sus valores por defecto.

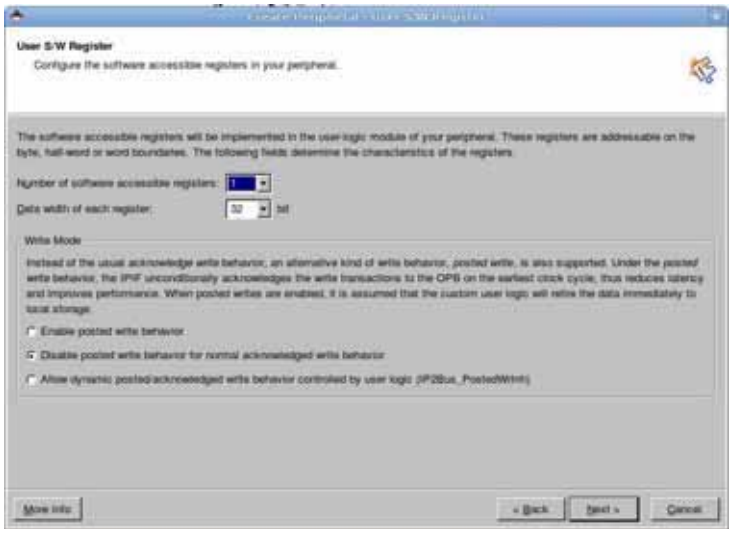


Figura I.8: Create Peripheral-User S/W Register.

12. Desplácese a través de la sección IP Interconect, que muestra las señales IPIC por defecto que estan disponibles para la lógica de usuario basadas en las secciones previas, hacer clic en Next.

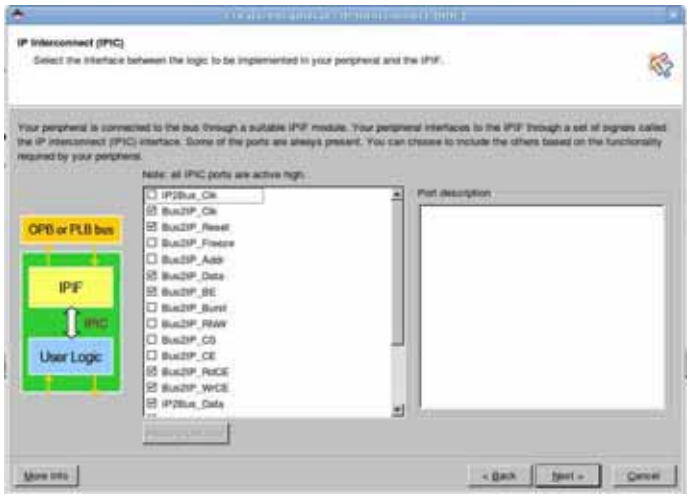


Figura I.9: Create Peripheral-IP Interconect (IPIC).

13. En la sección (OPTIONAL) Peripheral Simulation Support, desmarcar Generate BFM simulation platform, con el fin de no generar la simulación BFM asociada a los archivos y directorios, y hacer clic en Next.

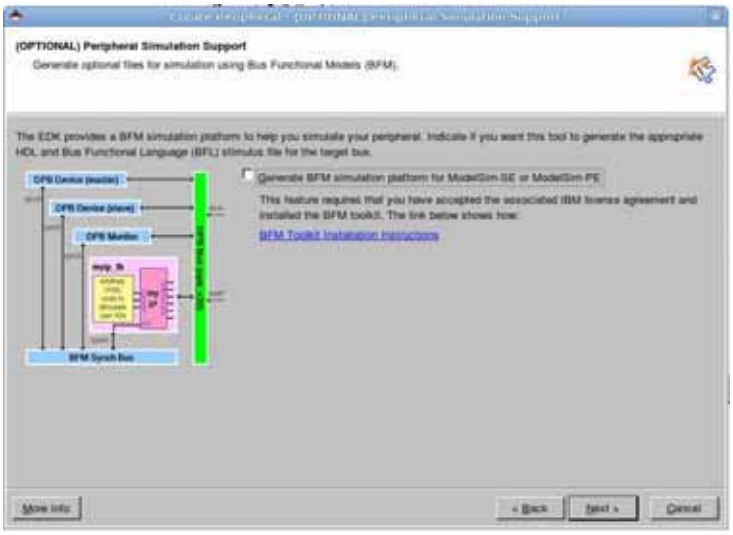


Figura I.10: Create Peripheral-(OPTIONAL) Peripheral Simulation Support.

14. En la sección (OPTIONAL) Peripheral Implementation Support, desmarcar Generate ISE and XTS project files to help you to implement the peripheral using XTS flow, marca Generate template driver files to help you to implement software interface y haz clic en Next.

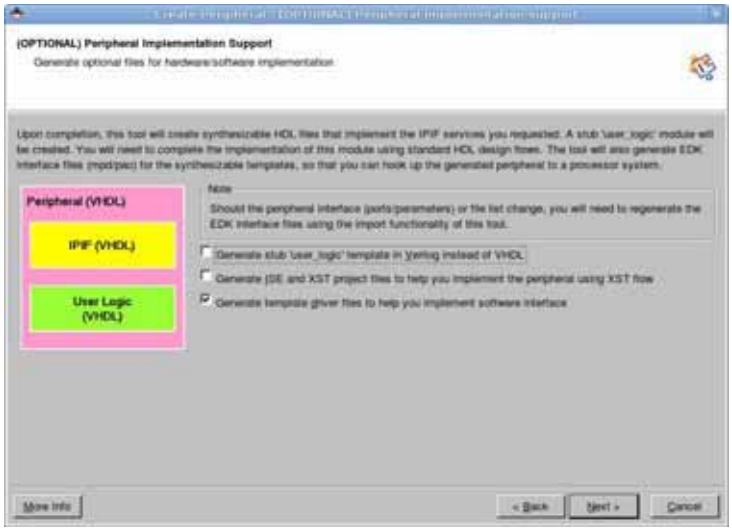


Figura I.11: Create Peripheral-(OPTIONAL) Peripheral Implementation Support.

15. Hacer clic en Finish para cerrar el resumen de información.

Agregar el puerto LED en el archivo my_led_v2_1_0.mpd generado por el asistente.

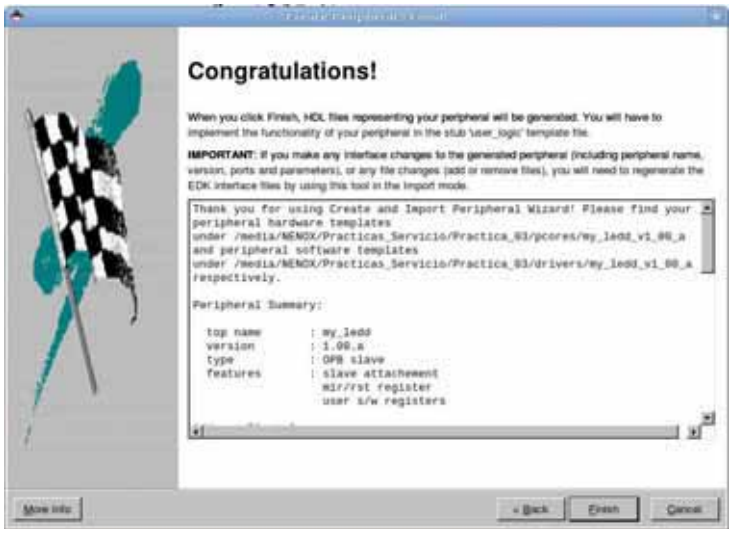


Figura I.12: Resumen de Información.

1. Usando un editor de texto abrir el archivo my_led_v2_1_0.mpd que se encuentra en el directorio Practica_03/pcores/my_led_v1_00_a/data.

2. Agrega la siguiente linea después de el puerto OPB_Clk debajo de la seccion de puertos.

   - PORT LED = "", DIR = O, VEC = [0:3]

   Esto es necesario para que el puerto aparezca en IP Catalog.

```
22 ## Generics for VHDL or Parameters for Verilog
23 PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector, MIN_SIZE = 0x200, BUS = 
   _HIGHADDR
24 PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector, BUS = SOPB, ADDRESS = HI
25 PARAMETER C_OPB_AWIDTH = 32, DT = INTEGER, BUS = SOPB
26 PARAMETER C_OPB_DWIDTH = 32, DT = INTEGER, BUS = SOPB
27 PARAMETER C_USER_ID_CODE = 3, DT = INTEGER
28 PARAMETER C_FAMILY = virtex2p, DT = STRING
29
30 ## Ports
31 PORT OPB_Clk = "", DIR = I, SIGIS = Clk, BUS = SOPB
32 PORT LED = "", DIR = O, VEC = [0:3]
33 PORT OPB_Rst = OPB_Rst, DIR = I, SIGIS = Rst, BUS = SOPB
34 PORT Sl_DBus = Sl_DBus, DIR = O, VEC = [0:(C_OPB_DWIDTH-1)], BUS = SOPB
35 PORT Sl_errAck = Sl_errAck, DIR = O, BUS = SOPB
36 PORT Sl_retry = Sl_retry, DIR = O, BUS = SOPB
37 PORT Sl_toutSup = Sl_toutSup, DIR = O, BUS = SOPB
38 PORT Sl_xferAck = Sl_xferAck, DIR = O, BUS = SOPB
```

3. Guarda el archivo y cierralo.

Agregar las declaraciones y lógica necesarias a los archivos my_led.vhd y user_logic.vhd.

1. Abrir con un editor de texto el archivo my_led.vhd que se encuentra en el directorio Practica_03/pcores/my_led_v1_00_a/hdl/vhdl.

2. Agrega el puerto de usuario LED debajo de la cadena USER PORTS ADDED HERE.

```
116     -- DO NOT EDIT ABOVE THIS LINE ------------------
117   );
118   port
119   (
120     -- ADD USER PORTS BELOW THIS LINE ------------------
121     --USER ports added here
122     LED : out std_logic_vector (0 to 3);
123     -- ADD USER PORTS ABOVE THIS LINE ------------------
124
125     -- DO NOT EDIT BELOW THIS LINE ------------------
126     -- Bus protocol ports, do not add to or delete
127     OPB_Clk                         : in  std_logic;
128     OPB_Rst                         : in  std_logic;
```

3. Busca el siguiente –USER y agrega la declaración de asignación de puertos.

```
396   )
397   port map
398   (
399     -- MAP USER PORTS BELOW THIS LINE ------------------
400     --USER ports mapped here
401     LED => LED,
402     -- MAP USER PORTS ABOVE THIS LINE ------------------
403
404     Bus2IP_Clk                    => iBus2IP_Clk,
```

4. Abre el archivo user_logic.vhd de el directorio /vhdl y agrega la definición de puerto LED en el área USER Ports.

```
84 entity user_logic is
85   generic
86   (
87     -- ADD USER GENERICS BELOW THIS LINE ---------------
88     --USER generics added here
89     -- ADD USER GENERICS ABOVE THIS LINE ---------------
90
91     -- DO NOT EDIT BELOW THIS LINE --------------------
92     -- Bus protocol parameters, do not add to or delete
93     C_DWIDTH                       : integer              := 32;
94     C_NUM_CE                       : integer              := 1
95     -- DO NOT EDIT ABOVE THIS LINE --------------------
96   );
97   port
98   (
99     -- ADD USER PORTS BELOW THIS LINE ------------------
100    --USER ports added here
101    LED : out std_logic_vector (0 to 3);
102    -- ADD USER PORTS ABOVE THIS LINE ------------------
103
104    -- DO NOT EDIT BELOW THIS LINE --------------------
```

5. Busca el próximo –USER y agrega la declaración de señal la interna para el user logic.

```
125 architecture IMP of user_logic is
126
127   --USER signal declarations added here, as needed for user logic
128   signal LED_i : std_logic_vector (0 to 3);
129   ---------------------------------------------
130   -- Signals for user logic slave model s/w accessible register example
131   ---------------------------------------------
```

6. Busca el próximo –USER logic implementation y agrega el siguiente código.

```
139 begin
140
141   --USER logic implementation added here
142
143   LED_PROC: process (Bus2IP_Clk) is
144   begin
145      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
146         if Bus2IP_Reset = '1' then
147            LED_i <= "0000";
148         else
149            if Bus2IP_WrCE(0) = '1' then
150               LED_i <= Bus2IP_Data (28 to 31);
151            else
152               LED_i <= LED_i;
153            end if;
154         end if;
155      end if;
156   end process LED_PROC;
157   LED <= LED_i;
158
```

7. Guarda los cambios y cierra el archivo.

I.3.2. Paso 2: Importar y agregar un IP personalizado al proyecto

Importing and Adding Custom IP to the Project

Usando el IP Catalog del XPS , agregar my_led al proyecto, hacer las conexiones de bus, generar las direcciones de la instancia my_led, agregar los puertos necesarios a la instancia, nombrarlos apropiadamente y darle un puerto de datos. Agregar lo siguiente al archivo UCF:

- Net led<0> LOC=AC4;
- Net led<1> LOC=AC3;
- Net led<2> LOC=AA6;
- Net led<3> LOC=AA5;

1. Abrir el proyecto de la Práctica_03 en XPS.
2. En el IP Catalog, expandir el Project Repository y agregar el periférico personalizado my_led al sistema de hardware.

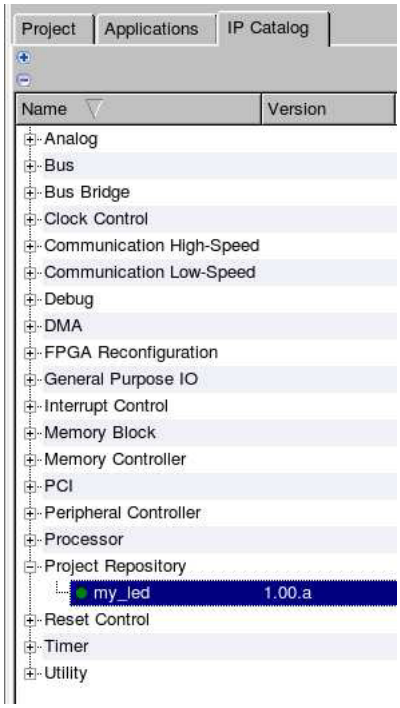


Figura I.13: User Created Peripheral in IP Catalog.

3. Cambia a la sección de conexiones de bus, conecta my_led a el bus OPB.

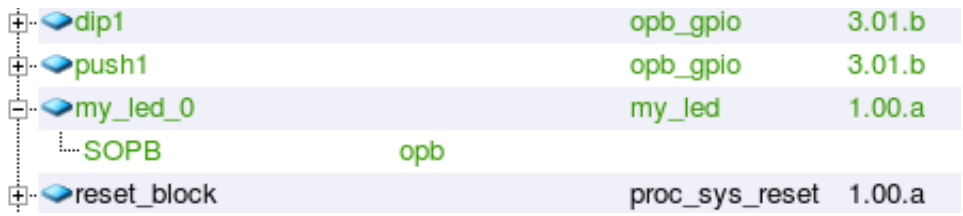
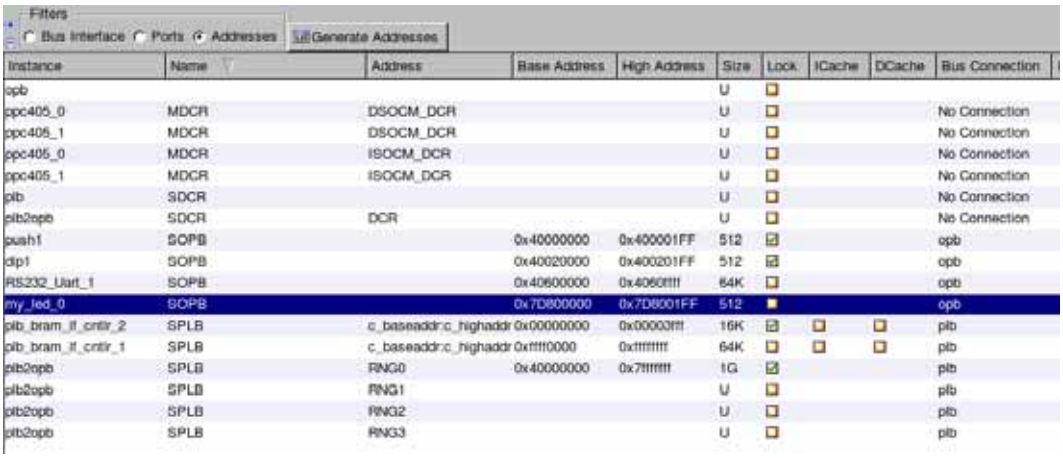


Figura I.14: Conexion de periférico creado al OPB bus.

4. Selecciona el filtro de direcciones y bloquea las direcciones del periférico my_led.

5. Selecciona como tamaño 512 para my_led y haz clic en Generate Addresses.

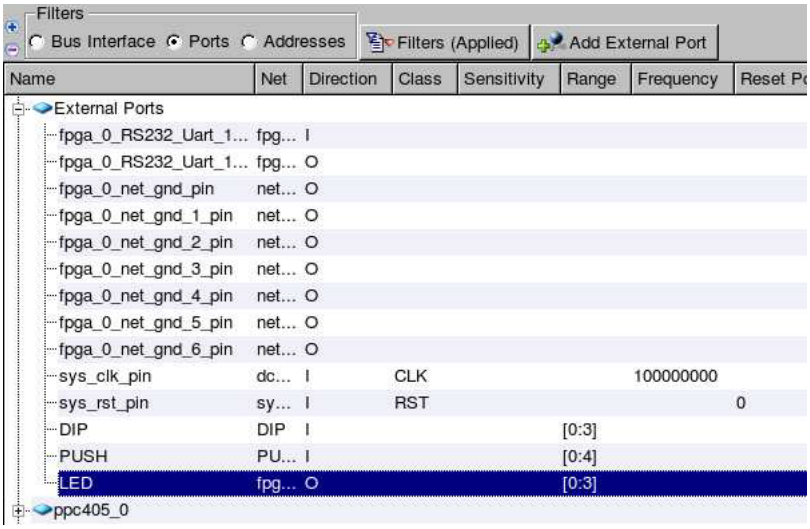


6. Cambia a la sección de puertos y aplica lo siguiente:

   - Localiza el puerto LED my_led_0's en la tabla, modifica el nombre de Net a fpga_0_LEDs_4Bit_GPIO_d_out.
   - Haz clic en Make External para hacer externo el paso de arriba y renombra el pin externos como led.

7. Abre el archivo system.mhs y verifica los siguientes fragmentos:

   - PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = I
   - PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = O
   - PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
   - PORT sys_rst_pin = sys_rst_s, DIR = I, RST_POLARITY = 0, SIGIS = RST
   - PORT dip = DIP, DIR = I, VEC = [0:3]



- PORT push = PUSH, DIR = I, VEC = [0:4]
- PORT led = fpga_0_LEDs_4Bit_GPIO_d_out, DIR = O, VEC = [0:3]

8. Agrega las siguientes asignaciones de pins al archivo UCF:

   - Net led<0> LOC=AC4;
   - Net led<1> LOC=AC3;
   - Net led<2> LOC=AA6;
   - Net led<3> LOC=AA5;

9. Guarda y cierra el archivo UCF.

I.3.3. Paso 3: Generar y Bajar el Bitstream

Generar y descargar el bitstream. Verificar que el diseño opera como en la práctica anterior.

1. Hacer clic en Configuration ⟶ Update Bitstream, para generar el archivo bit.
2. Bajar el archivo bit a la tarjeta.
3. Cambiar el estado del dip switch y los push buttons, verificar la salida en la hiperterminal.

Figura I.15: Salida en la hiperterminal.

Bibliografía

[1] McCarty, Bill, "Libro oficial de Red Hat Linux Firewalls",Anaya, 2003

[2] L. Ziegler, Robert, "Firewalls Linux: Guia Avanzada", Prentice Hall.

[3] Estándar RFC791 http://www.rfc-es.org/rfc/rfc0791-es.txt ,18 de Febrero del 2010

[4] Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual http://www.xilinx.com/univ/xupv2p.html

[5] Xilkernel se encuentra en http://www.xilinx.com o en la ruta de instalación de EDK

[6] Tesis de Maestría : Implementación en FPGAs de Algoritmos de Compresión-Descompresión para Dispositivos Móviles, M. en C. Oscar Alvarado Nava, 2007