UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN
SISTEMAS Y REDES DE TELECOMUNICACIÓN

PROYECTO FINAL DE CARRERA

SISTEMA DE GESTIÓN REMOTA DE CAMPAÑAS DE INYECCIÓN DE FALLOS EN CIRCUITOS DIGITALES MEDIANTE EMULACIÓN EN FPGA

AUTOR: ÁLVARO CRIADO BRAVO
TUTOR: MARIO GARCÍA VALDERAS

20 de Diciembre de 2012

Agradecimientos

A mis padres, que han hecho de mi lo que soy y a los que les debo todo.

A Marta, mi novia y amiga.

A mis abuelas. Sin sus preguntas: "Hijo, ¿cuándo vas a terminar?", no habría sido lo mismo.

A Mario, mi tutor.

A mis familiares y amigos.

A todos los compañeros que en mayor o menor medida han pasado por mi vida de estudiante. Dani, Xavi, Joe, Jaime, Rober, María, Jorge, Diana, Kike, Elena, Balta, Jony, Marta, Isma, Almu, Alvaro, Andres, Barco, Estefanía, Marian, Alfon, Gerar, Chuso, Nacho, Rubens, Fernando, Lentisco, Taoufik, MJ, Joseco, Neo, Kun, Wilian Walace, Gustavo, Bertos...

A todos los profesores, de los que he aprendido muchísimo no sólo del ámbito académico.

A todos y cada uno de los integrantes del Grupo de Diseño Microelectrónico y Aplicaciones. Durante los últimos años han sido mi segunda familia. En especial a Marta Portela por sus largas horas dedicadas a mi formación.

Resumen

Este proyecto nace de la necesidad de ampliación de las características de un emulador autónomo de fallos. Estos emuladores son capaces de evaluar la robustez frente a radiaciones de un circuito digital, y por tanto, pueden realizar estudios de tolerancia a fallos.

Cuando los circuitos digitales que se quieren estudiar crecen en tamaño y complejidad, la cantidad de datos y resultados que se generan al evaluar su tolerancia a fallos también crece. Además, hoy en día, la tecnología con la que se fabrican dispositivos lógicos programables permite generar circuitos de un tamaño creciente con el tiempo. Esta necesidad de ancho de banda es de donde nace el objetivo principal de este Proyecto Fin de Carrera.

Una característica a mejorar de estos emuladores es la posibilidad de obtener el diccionario de fallos completo, que contiene todas las clasificaciones realizadas. Uniendo esta necesidad de mejora al progreso de la tecnología, se ha necesitado dotar al sistema de una comunicación con gran ancho de banda, por lo que se ha elegido el protocolo TCP/IP sobre Ethernet.

Por otro lado, al usar una comunicación Ethernet, se ha implementado un programa cliente que accede a la plataforma de emulación y es capaz de descargar resultados, hacer seguimientos y configurar emulaciones desde cualquier lugar con conexión a internet.

Tras analizar las necesidades se ha utilizado como pieza fundamental del sistema la plataforma Microblaze de Xilinx, facilitando la comunicación TCP/IP entre el cliente y el servidor de emulaciones.

Abstract

This project improves the features of an autonomous emulation fault injection. This emulator is able to evaluate the robustness of a digital circuit regarding radiation effects, in order to study the fault tolerance level.

When the digital circuit under test grows in size and complexity, the amount of data generated by the emulator grows too. According to this, the main objective of this project is to increase the data transmission capabilities of the emulation system.

With a high data throughput, it is possible to manage the transmission of the complete fault dictionary, which contains the classification of each possible tested fault (up to billions). In order to reach this objective, the aim of this project is to use a TCP/IP protocol over an ethernet interface.

Furthermore, a client application had been implemented in order to access the emulation platform and several capabilities, such as downloading the results, emulation progress tracking and remote configuration and operation.

Xilinx Microblaze microprocessor has been used as the main part of the platform system, providing the communication interface between the client (personal computer) and the server (emulation system).

Índice general

Lista de Figuras

Lista de Tablas

Lista de Acrónimos

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
DHCP	Dynamic Host Configuration Protocol
DMA	Grupo de Diseño Microelectrónico y Aplicaciones
DNS	Domain Name System
DUT	Devide Under Test
EDK	Embebed Development Kit
EMAC	Ethernet Media Access Controller
FAT	File Allocation Table
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
IDE	Integrated Development Environment
IP	Intellectual Property
IP	Internet Protocol
MCU/MBU	Multiple-Cell Upset / Multiple-Bit Upset
JTAG	Joint Test Action Group
LWIP	LightWeight Internet Protocol
OSI	Open System Interconnection
PDU	Protocol Data Unit
POSIX	Portable Operating System Interface uniX
PLB	Processor Local Bus
SDK	Software Development Kit
SEFI	Single-Event Functional Interrupt
SEL	Single-Event Latch
SET	Single-Event Transient
SEU	Single Event Upset
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
VLSI	Very Large Scale Integration
XPS	Xilinx Platform Studio

Capítulo 1

INTRODUCCIÓN

1.1. Motivación

La microelectrónica es la ciencia que intenta aplicar la ingeniería electrónica a componentes y circuitos de dimensiones muy pequeñas. Esta naturaleza junto con la evolución de la tecnología ha provocado que cada vez los circuitos tengan mayor complejidad y menores dimensiones. Según los circuitos se hacen más pequeños, también se debilitan frente a factores ambientales, como son las variaciones de temperatura, de la alimentación, interferencias electromagnéticas o la radiación cósmica. En determinados entornos, como el aeroespacial, esta última tiene especial interés debido a los efectos que llega a provocar en los circuitos. La radiación cósmica está compuesta por un conjunto de partículas cargadas como protones, partículas alfa e iones pesados entre otras. Además, este fenómeno puede estar acompañado de radiaciones electromagnéticas de muy alta energía [5]. El tamaño actual de los circuitos electrónicos, y más concretamente la escala de integración, hace que una sola partícula pueda afectar a más de un elemento de memoria (Figura 1.1). Esta afección es conocida como fallo y se entenderá como tal a un cambio en el estado de un elemento de memoria, o más concretamente, un biestable.

Durante los primeros lustros de vida de la microelectrónica, los circuitos que se construían poseían unas dimensiones que los hacían, sin quererlo, muy robustos a radiaciones electromagnéticas. Sin embargo, los circuitos VLSI (very large scale integrity) que han venido apareciendo con el devenir del progreso, han traido consigo una predisposición a sufrir distintos tipos de afecciones. Los fallos suaves o Soft Errors son uno de los tipos de fallos mas importantes que puede padecer un circuito integrado y ahondando más, los fallos SEUs (single event upset). Consistente en un fallo producido por un único evento, es decir, se asume que los errores generados van a manifestarse de uno en uno como sucesos aislados.

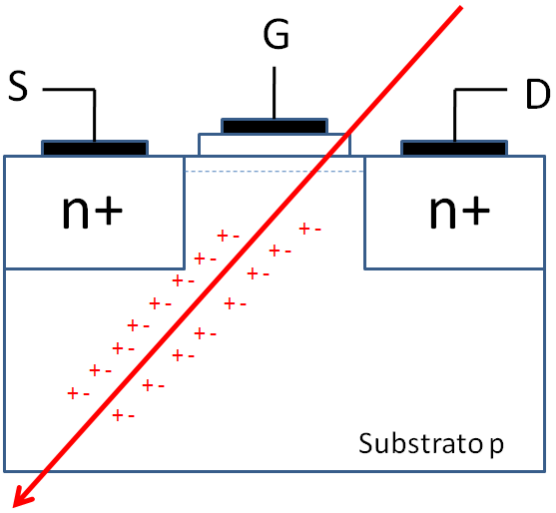


Figura 1.1: *Generación de pares electrón-hueco por la acción de una única partícula [5].*

Estos efectos pueden ser desde el cambio de estado de un elemento de memoria (bitflip) a la destrucción temporal o permanente del circuito. Debido a que el efecto más habitual en entornos moderadamente hostiles es el fallo SEU, los expertos han venido investigando multitud de técnicas de mitigación de estos efectos [5].

Por este motivo surge la necesidad de cuantificar cómo de robustos a radiaciones son los circuitos que se diseñan, para así evitar, en la medida de lo posible, los efectos que provocan en su funcionamiento. Para satisfacer esta necesidad, el DMA desarrolló una herramienta que evalúa la resistencia de un circuito digital a fallos SEU. Por tanto, este proyecto está enmarcado en el departamento de Tecnología Electrónica, dentro del grupo de Diseño Microelectrónico y aplicaciones.

El circuito emulador es capaz de clasificar distintos tipos de fallos que podrían producirse en un entorno real:

- Fallos latentes, se producen cuando el fallo continúa en el circuito, es decir, algún elemento de memoria contiene un estado distinto al que debería tener pero aún no ha producido un malfuncionamiento perceptible en las salidas. Esta clasificación denota que el banco de pruebas no es lo suficientemente largo o exhaustivo como para que el error sea perceptible en el funcionamiento.
- Fallo silencioso, este tipo de fallo se clasifica cuando en el interior del circuito no existe ningún elemento de memoria que tenga un valor incorrecto. El fallo ha desaparecido sin provocar un malfuncionamiento. Este tipo de clasificación tiene lugar, por ejemplo, cuando un elemento de memoria que contiene fallos se sobrescribe o resetea, quedando el dispositivo tal y como estaría si no hubiera sido objeto de fallo.
- Avería. Este fallo es clasificado cuando el circuito se comporta de forma errónea, y por

tanto, sus salidas poseen valores erróneos comparándolas con las que deberían tener. En este caso el fallo provocado por un evento SEU ha generado un funcionamiento erróneo del dispositivo, que es lo que los investigadores del DMA intentan evitar.

La arquitectura de este sistema es la que se muestra en la figura 1.2. Se aprecia un emulador con un bloque para comunicaciones serie y un PC en el que se leen los resultados de la campaña de emulación. La característica fundamental del emulador de fallos es que es capaz de inyectar millones de fallos en unas pocas horas a una tasa máxima de una clasificación por cada 5 ciclos de reloj. Esta alta velocidad de clasificación de fallos hace que sea un reto obtener la lista completa de todas las clasificaciones. A esta lista se la llama "diccionario completo de emulación". Para conseguirlo existen dos posibles soluciones. La primera consiste en almacenar el diccionario en memoria. Para ello puede hacerse uso de la memoria interna de la FPGA o la RAM externa con la que cuenta la tarjeta. Como se explicará más detalladamente, cada clasificación se codifica con 2 bits y haciendo un cálculo aproximado tomando 50.000.000 fallos:

$$50000000 fallos \times 2 \frac{bits}{fallo} = 100000000 bits \approx 120 MBytes \tag{1.1}$$

Es decir, para un caso típico, es necesario disponer de una cantidad de memoria nada despreciable y una velocidad de escritura alta.

La segunda solución consiste en transmitir las clasificaciones por un canal de comunicaciones mientras se están generando. Esta segunda opción es la más tecnológicamente asequible y es la que se emplea en este Proyecto Fin de Carrera. En este caso se evita tener una memoria de gran tamaño y alta velocidad, pero a cambio se requiere de una comunicación con un ancho de banda alto. En algunos casos, esta cantidad de datos se generarán a una tasa muy alta, llegando a velocidades de una clasificación cada 5 ciclos de reloj [5]. Realizando unos cálculos aproximados se pueden llegar a alcanzar tasas de 20Mbps.

$$50 MHz \times 2 \frac{bits}{clasificacion} \times \frac{clasificaciones}{5 ciclos} = 20 Mbps \tag{1.2}$$

El sistema de partida cuenta con una interfaz RS-232 a una velocidad de 115200 bps. Por este motivo surge la necesidad de mejorar las capacidades de comunicación del emulador autónomo de fallos.

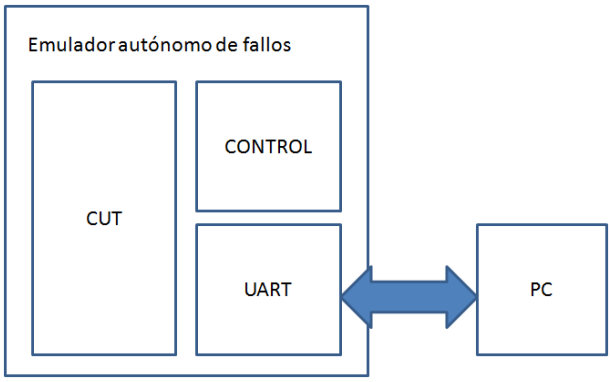


Figura 1.2: *Emulador Autónomo de fallos.*

1.2. Objetivos

Las comunicaciones son un cuello de botella en la emulación autónoma de fallos, en concreto a la hora de transmitir en tiempo real o cuasi tiempo real el diccionario de fallos completo al PC desde el que se lanza. Por este motivo el primer objetivo es integrar un procesador MicroBlaze junto al emulador cuyas funciones sean:

- Transmitir datos de configuración desde el ordenador hacia el emulador.
- Recoger los resultados del emulador.
- Implementar comunicaciones vía Ethernet.

El segundo objetivo es desarrollar un programa cliente capaz de conectarse con el emulador, configurarlo, lanzar emulaciones y obtener resultados.

1.3. Breve descripción de la memoria

Este Proyecto Fin de Carrera está dividido en 3 bloques fundamentales. El sistema de emulación autónoma, el microprocesador Microblaze empotrado en la FPGA y el programa cliente que conectará con el sistema de emulación.

Durante el segundo capítulo se describe el hardware empleado y las aplicaciones de desarrollo utilizadas a lo largo del proyecto.

El tercer capítulo ofrece una descripción del sistema completo, tanto del bloque Microblaze como de sus periféricos y comunicaciones. También se realizará una introducción al sistema de emulación autónoma y su configuración.

El cuarto capítulo está destinado a presentar el diseño hardware del servidor pormenorizando en su arquitectura, comunicaciones y lógica implementada.

Durante el quinto capítulo se exponen las especificaciones y requisitos que debe cumplir el servidor de emulaciones así como los pormenores de la creación de este bloque funcional. Se encontrarán todas las características del software implementado para su ejecución en el entorno Microblaze junto con el hardware necesario para funcionar.

En el siguiente capítulo se detalla cada componente del programa cliente, comunicaciones, tratamiento de los resultados, generación de informes e interfaz gráfica. También se detalla la arquitectura general de la aplicación, flujo de ejecución y algoritmos que transforman los datos recibidos del sistema emulador en clasificaciones organizadas de fallos.

En el séptimo capítulo se encuentra explicado el proceso de integración del sistema completo, las conexiones entre el ordenador y la tarjeta, la carga del bitstream en la FPGA y una breve guía con los pasos necesarios para poder realizar una emulación de un circuito digital.

Por último, el octavo capítulo expone las conclusiones y realiza una revisión de los requisitos cumplidos, así como de las partes a mejorar.

Capítulo 2

SISTEMA DE EMULACIÓN DE FALLOS

Durante este capítulo se describe el sistema de partida, qué se pretende conseguir, los elementos que lo componen, la plataforma empleada, así como las capacidades de comunicaciones necesarias. El sistema de partida consta de un emulador de fallos con una interfaz serie RS-232 que se describe en los siguientes puntos.

2.1. Sistema de emulación de fallos

Para comenzar, se realizará una descripción del emulador autónomo de fallos, de donde viene la necesidad de usarlo, para qué sirve y de qué partes está formado.

2.1.1. Introducción

Los sistemas de emulación autónoma surgen con la integración cada vez mayor de los circuitos electrónicos. Dadas las debilidades de estos circuitos se necesita evaluar su tolerancia a fallos para conocer si son fiables en determinados escenarios.

Durante los primeros compases de la historia de la inyección de fallos, se utilizaban diversas técnicas para evaluar la robustez a los distintos tipos de eventos producidos en los circuitos [5].

- Exponer el circuito a radiación con iones pesados.
- Inyección de fallos con láser.

- Inyección de fallos con interferencias electromagnéticas.

Estos métodos tienen la ventaja de que ofrecen resultados altamente fiables, ya que se trabaja con un ejemplar del circuito ya fabricado y se le somete a un entorno hostil similar al que se encontrará en el peor de los casos en su lugar de funcionamiento definitivo. Sin embargo requieren un gran desembolso para fabricar el dispositivo y de grandes y costosas instalaciones para generar radiaciones.

Por este motivo nace la inyección basada en simulación, que aporta una gran flexibilidad en las campañas, así como diversos modelos de fallo. Entre los modelos de fallo existentes, concretamente los eventos simples, se encuentran los SEU (Single-Event Upset), SET (Single-Event Transient), SEFI (Single-Event Functional Interrupt), MCU/MBU (Multiple-Cell Upset / Multiple-Bit Upset) y los SEL (Single-Event Latch). La inyección de fallos se realiza a través de simuladores HDL (Hardware Description Language) mediante comandos del simulador, ó modificando el modelo HDL para incluir la capacidad de inyección de fallos. El inconveniente de esta técnica es el elevadísimo coste computacional, que necesitaría de semanas de cómputo para obtener los mismo resultados que ofrece la inyección física de fallos en unos minutos.

Para evitar todas estas desventajas surge la emulación de fallos, que supone una situación intermedia a la simulación y la inyección física.

Para utilizar esta técnica es necesario disponer de una descripción hardware del circuito y un banco de pruebas lo más exhaustivo posible. Se trata de modificar la descripción hardware del circuito introduciendo la posibilidad de generar fallos en los propios elementos de memoria, para después realizar una ejecución del banco de pruebas comparando alternativamente ciclos con fallo inyectado y ciclo sin fallo.

Los pasos a seguir para crear una campaña de emulación autónoma son los siguientes:

- Obtener una descripción hardware del circuito a probar (DUT).
- Generar una netlist con jerarquía plana del circuito de test. Tras este paso se obtiene un archivo que contiene toda la descripción hardware hasta el nivel de puerta.
- Sustituir los elementos de memoria por “Instrumentos”. Estos instrumentos contienen una copia del elemento para la ejecución sin fallo y otra para la ejecución con fallo, además incluyen un biestable de estado para llevar al circuito a una situación controlada conocida y un biestable de máscara que indica si se debe generar bitflip o no.
- Construir el emulador como una carcasa que se coloca sobre el circuito para controlar sus entradas y comprobar sus salidas. De esta forma clasifica los fallos posibles. Averías, latentes y silenciosos.

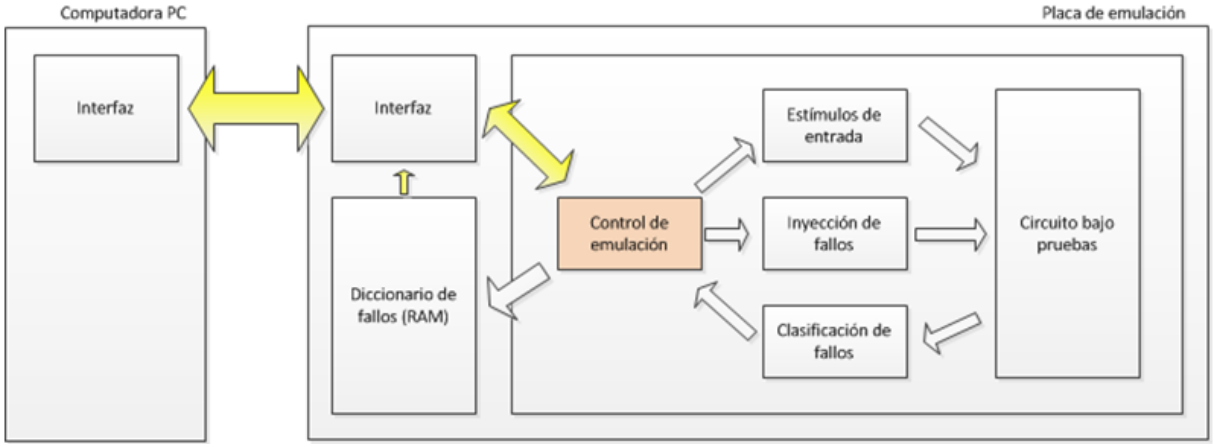


Figura 2.1: *Arquitectura del servidor de emulación [5].*

2.1.2. Arquitectura

La arquitectura del emulador autónomo de fallos propuesta por el grupo de Diseño Microelectrónico y Aplicaciones se puede observar en la figura 2.1.

El DMA es pionero en adoptar esta arquitectura con algunas mejoras, consiguiendo así un incremento sustancial en el rendimiento [2]. La tendencia anterior consistía en realizar todo el proceso de emulación en un PC y dejar en la FPGA tan sólo el circuito a probar. Lo que producía un cuello de botella en las comunicaciones entre tarjeta y PC.

De esta forma, los estímulos, el control de la emulación, la inyección y clasificación de fallos pasan a estar en la propia FPGA.

El proceso de emulación consiste en forzar un cambio de estado en un biestable del circuito en un instante determinado y observar su ejecución futura. Una vez clasificado se procede a inyectar en el ciclo siguiente del banco de pruebas. Cuando se ha terminado con el rango de ciclos de inyección definido al inicio, se pasa a realizar la misma tarea con el siguiente biestable. Este proceso se repite hasta terminar con todos los biestables del circuito.

El módulo de control se encarga de orquestar todo el proceso de emulación. Genera los estímulos de entrada del circuito bajo pruebas y verifica el valor de sus salidas. Para conocer si el circuito se está comportando erróneamente se suceden ejecuciones con fallo y sin el ciclo a ciclo. Estas ejecuciones se denominan “Golden” y “Faulty”. En el momento en el que estas dejen de ser idénticas se clasifica el fallo como avería.

La posibilidad de ejecutar ciclos Golden y faulty alternativamente viene otorgada por el “instrumento” (Figura 2.2), un circuito diseñado en el DMA que viene a sustituir a cada elemento del memoria del circuito bajo pruebas durante el proceso de “instrumentación”

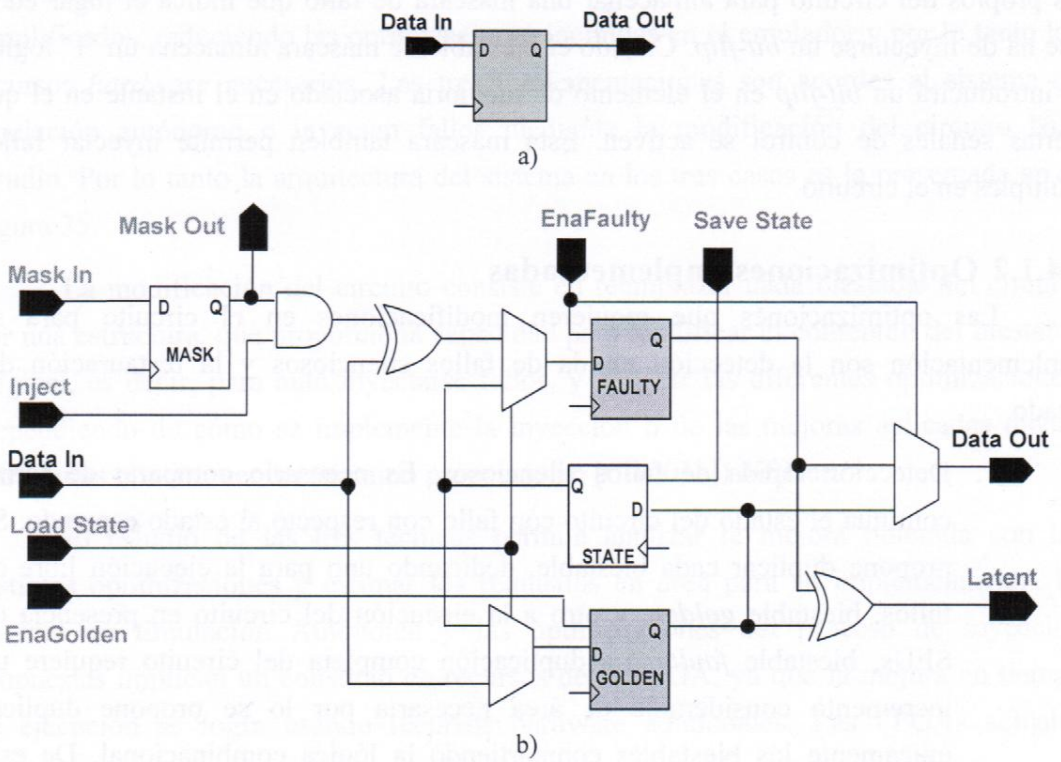


Figura 2.2: *a) biestable original del circuito a modificar. b) instrumento que sustituye al biestable original [5].*

que se detallará posteriormente. El instrumento contiene un biestable de máscara, para saber si inyectar fallo o no. Un biestable para la ejecución Golden y otro para la Faulty. Por último, incluye un biestable de estado, que proporciona la capacidad de restauración del estado inmediatamente anterior a la inyección. Esto es muy útil para optimizar la velocidad de emulación, ya que no es necesario ejecutar el banco de pruebas hasta el instante de inyección, sino que se restaura al estado al momento anterior y se inyecta en el ciclo siguiente.

En todo momento el módulo de control conoce el estado del circuito bajo pruebas gracias a la señal de estado que incorpora el instrumento. De este modo, el control clasifica los fallos y restaura su valor en el momento óptimo, asegurando velocidades de inyección realmente altas.

En la versión de la que parte este Proyecto Fin de Carrera, las órdenes que recibe el módulo de control desde el exterior llegan a través del puerto serie en forma de comandos. Dada la naturaleza automática de este módulo, los comandos son muy reducidos y simplemente permiten cargar datos, leerlos y lanzar la emulación. En la práctica, los comandos se envían mediante el cliente de conexiones serie Hyperterminal. Una muestra de los comandos que emplea se muestran en la tabla 2.1 [3].

Tabla 2.1: *Comandos del sistema emulador de partida*

Caracter	*Acción realizada*
#	*Pasa al registro siguiente*
.	*Orden de comienzo de emulación*
-	*Mostrar registros*

2.1.3. Circuito Bajo Test

Se conoce como circuito bajo test al módulo del que se desea conocer su tolerancia a fallos. Este circuito se insertará en el emulador y este a su vez en el periférico del MicroBlaze.

Para realizar las pruebas y diseños se ha elegido un circuito llamado “ClockManager”. Este módulo es uno de los que operan el satélite artificial Optos, desarrollado en el INTA. En un satélite es crítico que todos los elementos se encuentren sincronizados y funcionando de forma correcta. Para aportar esta sincronización se utiliza el módulo ClockManager, que se encarga de sincronizar la temporización del resto de módulos operativos.

El funcionamiento concreto es confidencial, pero se puede resumir en que contiene contadores, realiza comprobaciones y distribuye el tiempo real entre el resto de módulos.

Este circuito ya ha sido evaluado en el grupo DMA en cuanto a tolerancia a fallos se refiere. Por tanto, es un ejemplo del circuito típico que puede llegar a evaluarse con esta herramienta.

2.2. Modificaciones en el emulador de partida

Las modificaciones que se han realizado en el emulador de partida consisten en eliminar el bloque de comunicación serie y colocar una comunicación Ethernet. Para ello se utiliza un procesador MicroBlaze con el circuito emulador conectado a el como periférico. En la figura 2.3 se muestra un diagrama de bloques con la arquitectura del sistema implementado a diferencia del de partida, en la figura 2.1.Para la realización de esta arquitectura ha sido necesario desarrollar lógica de unión entre el emulador y el MicroBlaze, así como las aplicaciones software que ejecutará el procesador y el PC cliente.

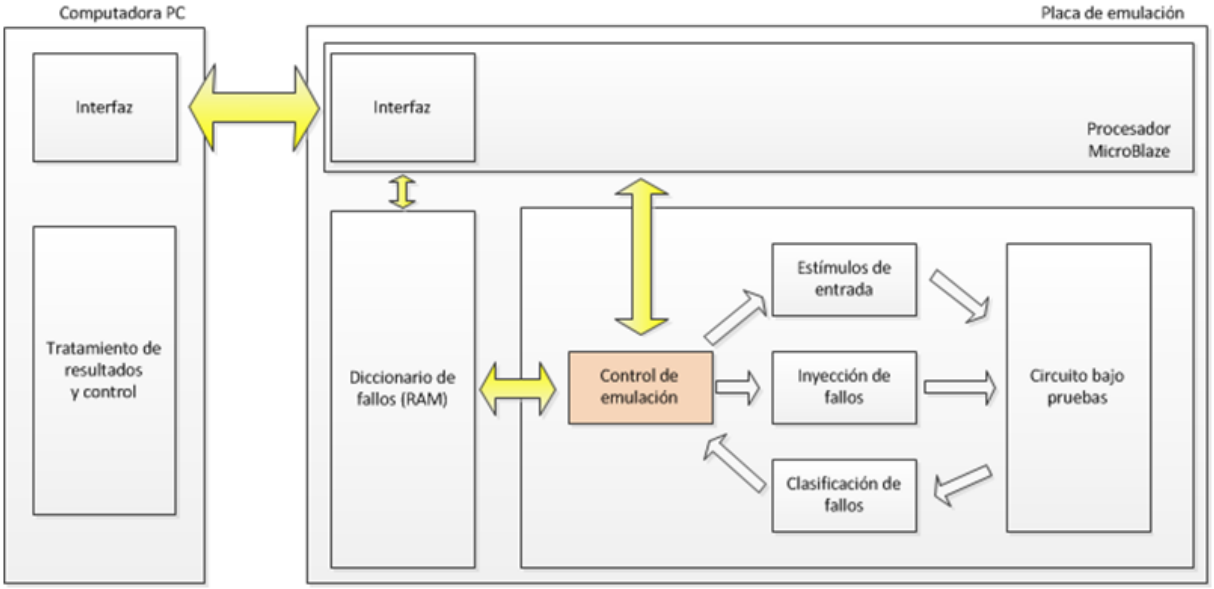


Figura 2.3: *Arquitectura del servidor de emulación con una interfaz Ethernet.*

2.3. Aplicación específica para el Servidor de Emulaciones

Como ya es sabido, este proyecto parte de la necesidad de incorporar una interfaz Ethernet al emulador descrito en el punto anterior. Para ello se ha optado por utilizar una solución con microprocesador embebido en la FPGA, en concreto el procesador MicroBlaze. MicroBlaze es un procesador de 32 bits de propósito general que ofrece Xilinx para soluciones en las que sea necesario integrar procesado software con hardware específico. Una de las ventajas más importantes de este procesador es que es totalmente configurable y por tanto ocupa sólo el área necesaria.

La razón de utilizar un procesador embebido en la nueva arquitectura es la facilidad y transparencia con la que incluye funciones y bibliotecas de transferencia de datos. Sin ellas sería necesario implementar la torre de protocolos OSI hasta el nivel de transporte (físico, enlace, red y transporte), es decir, los protocolos Ethernet, IP y TCP. No es necesario comentar lo tedioso e inabordable de tal empresa.

Para facilitar aún más la tarea, Xilinx ofrece un sistema operativo que actúa como una biblioteca de funciones para obtener varios hilos de ejecución simultáneos, facilitar el uso y configuración de timers e interrupciones y otras características como las siguientes [21].

- API POSIX, que incluye hilos, sincronización entre hilos, comunicación entre procesos, timers e interrupciones.
- Amplia capacidad de configuración.

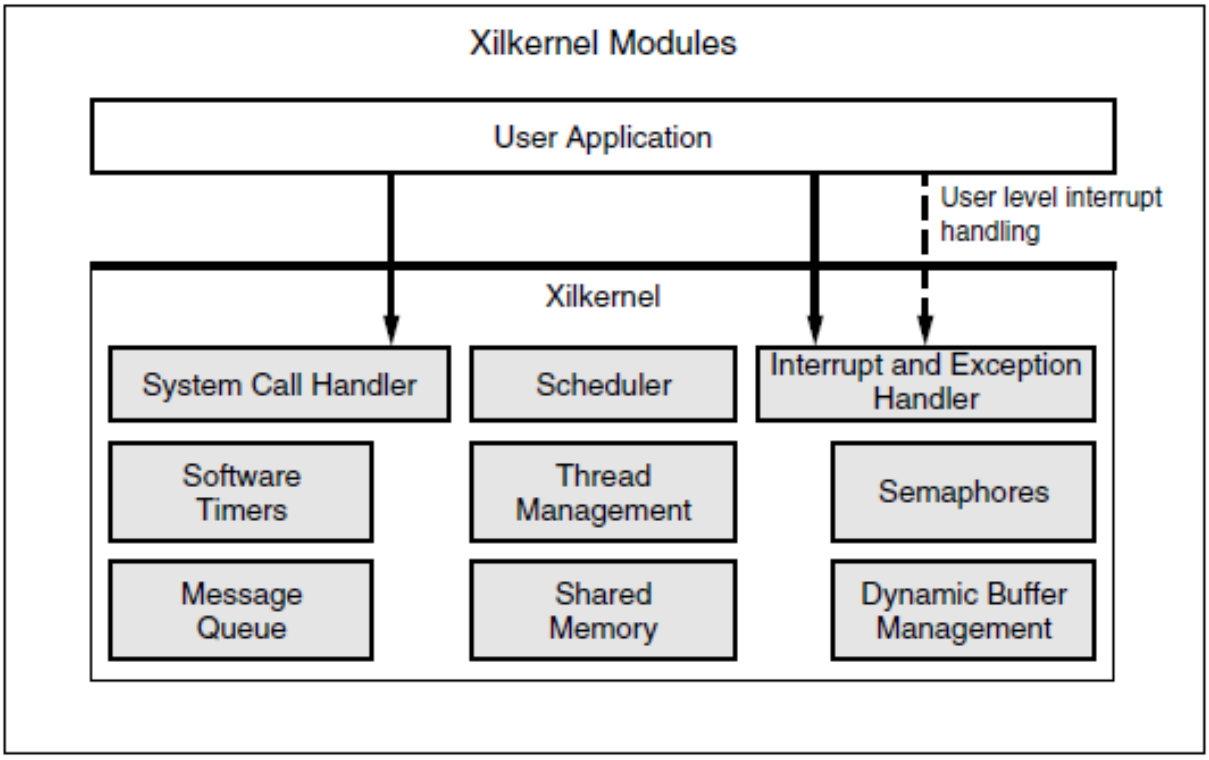


Figura 2.4: *Arquitectura y alcance de Xilkernel*

En la figura 2.4 se muestra un diagrama del alcance de Xilkernel, en el que las aplicaciones del usuario se sitúan sobre la capa de Xilkernel y hacen uso de las funciones que este provee. Las funciones permiten el acceso de forma simplificada a los registros del sistema, así, como a las funcionalidades propias que incorpora (Timers, gestión de hilos, comunicación entre hilos, etc). La arquitectura elegida para la aplicación a ejecutar en el MicroBlaze sigue el paradigma "Cliente-Servidor" que deberá cumplir los siguientes objetivos básicos.

- Permitir establecer conexiones a través de una red IP con un sistema cliente.
- Ofrecer un conjunto de comandos de control y seguimiento.
- Servir los resultados de las emulaciones en tiempo real.

Para conseguir estos requisitos básicos, el sistema microprocesado poseerá un periférico emulador y deberá ser capaz de tutelar el proceso de emulación, así como configurar el arranque y parada del mismo.

2.4. Aplicación específica para PC

La aplicación específica para el PC se corresponde con un programa cliente multihilo con las capacidades descritas a continuación.

La aplicación multihilo proporciona el desempeño adecuado en las tareas que realiza, además de permitir desglosar tareas independientes que se ejecutan concurrentemente. Estas tareas son las encargadas de gestionar la interfaz gráfica de usuario y la de obtener resultados en tiempo real. Ambas tareas serán explicadas detalladamente en el capítulo 6.

Los requisitos de este sistema obligan a que sea posible realizar conexiones con un servidor de emulaciones, por tanto debe permanecer con conexión a la red permanentemente. Además se pretende ofrecer una comunicación robusta a caidas o saturación en la red mediante el uso de un stack TCP/IP que crea un entorno orientado a conexión. JAVA ofrece una biblioteca de funciones que abstrae al programador de todos los aspectos relacionados con la conexión física entre los dispositivos y proporciona una “caja negra”.

Como añadido, esta aplicación almacena y procesa los resultados de las emulaciones permitiendo un estudio posterior del diccionario completo de fallos. Facilitando, así, la obtención de conclusiones en los experimentos.

Todo ello bajo una interfaz gráfica amigable que facilite el uso de estas funcionalidades.

2.5. Periféricos Empleados

Los periféricos son circuitos externos al procesador embebido y que junto a él conforman el sistema que se programará en la FPGA. Los periféricos pueden encargarse de diferentes tareas en un diseño, por ejemplo incrementar la potencia de cálculo del procesador añadiendo módulos que calculen en hardware determinados algoritmos que en software serían cientos de veces menos eficientes. Otra tarea para la que están ideados los periféricos es la de servir de interfaz de control de circuitos externos a la FPGA, por ejemplo controladores de RAM, de comunicaciones o de gráficos. En resumen, las posibilidades que ofrecen los periféricos son casi ilimitadas. Los empleados en la realización del proyecto se describen a continuación.

- RS-232. Este periférico se usa en el proyecto para labores de depuración, a pesar de no tener una velocidad elevada permite conocer el estado interno del sistema en todo momento.
- LEDs indicadores en la placa. Permiten conocer de forma rápida el estado del sistema sin necesidad de conectar un PC para depurar.

- Registros de comunicación entre el Microblaze y el emulador.
- Memoria compartida. Se utiliza para almacenar temporalmente los resultados de la emulación hasta que son enviados al programa cliente.
- Controlador Ethernet para las comunicaciones por internet mediante sockets.
- Emulador autónomo de fallos.

Posteriormente en este documento se detallarán estos periféricos, así como sus configuraciones y prestaciones finales en los casos que proceda.

La razón de utilizar memoria compartida y registros simultáneamente responde a la problemática de implementar dos tipos de comunicación entre el Microblaze y el emulador.

- Por un lado, la comunicación bidireccional implementada con registros para el control del emulador y seguimiento del estado del mismo. Puede darse el caso en el que un registro es leído por el microblaze mientras el emulador escribe su estado en otro. Esta simultaneidad de acciones solo puede implementarse mediante registros. En un bloque de memoria compartida el acceso es único y exclusivo.
- Por otro lado, la comunicación unidireccional se emplea cuando el emulador escribe las clasificaciones de los fallos y posteriormente el Microblaze los lee para enviarlos por Ethernet. Este paradigma de comunicación también puede resolverse mediante registros, pero el alto coste computacional y de memoria necesario a la hora de la síntesis hace necesaria su implementación en forma de memoria compartida.

2.6. Comunicaciones empleadas

Una parte fundamental de este Proyecto Fin de Carrera son las comunicaciones, por la naturaleza y necesidades del mismo. En él se hace uso de una comunicación serie y una comunicación Ethernet. Además es necesaria una conexión por medio de JTAG para la programación de la FPGA, pero al ser transparente al usuario final no se profundizará en ella.

2.6.1. Comunicación serie RS-232

Se ha utilizado la comunicación RS-232 entre la tarjeta y el PC con fines de depuración a lo largo de todas las fases de desarrollo del proyecto. Para recibir los datos del puerto serie en el PC se ha empleado el software HyperTerminal con los parámetros de configuración visibles en la tabla 2.2

Tabla 2.2: *Parámetros de comunicaciones RS-232*

Puerto	*COM1*
Tasa de Transmisión	*115200*
Datos	*8 bits*
Paridad	*No*
Bit de parada	*1 bit*
Control de flujo	*No*

Estas comunicaciones son una forma muy fiable y rápida de implementar y depurar sistemas embebidos como el de este proyecto. Sin embargo no proveen la velocidad suficiente como para transmitir el diccionario completo de fallos, que es el caso que nos ocupa. Por tanto, quedará relegada a transferir mensajes de depuración y de información de estado al PC cliente.

2.6.2. Comunicación Ethernet

Las comunicaciones entre la tarjeta e internet se realizan a través de una interfaz Ethernet de velocidad 10/100Mbps. Esta velocidad viene limitada a 1Mbps por el API de sockets ofrecido por Xilinx como se explica posteriormente en este documento [16].

Por el lado del cliente, al usar el lenguaje Java, queda oculta toda la configuración de las conexiones y es extremadamente sencillo establecer este tipo de comunicación. Tan sólo es necesaria una dirección IP y un puerto al que enviar las peticiones de establecimiento de conexión.

Aunque la velocidad final es menor de lo que se hubiera deseado, es 10 veces mayor que la tasa nominal de la comunicación serie.

Capítulo 3

ENTORNO DE TRABAJO

Durante este tercer capítulo se detallan las herramientas empleadas para la realización del proyecto así como el hardware utilizado. Para llevar a cabo los propósitos descritos en el primer capítulo se ha utilizado una tarjeta de evaluación de propósito general XUPV5 LX110T que monta una FPGA Virtex5 [18] y diversos periféricos. Esta placa está comercializada por AVNET [1].

El diseño y desarrollo del Hardware se ha realizado utilizando la herramienta de síntesis Symplify, El programa Xilinx ISE y el software Xilinx Platform Studio (XPS). Estas herramientas son algunas de las que utiliza el grupo DMA en sus investigaciones.

Para implementar el programa cliente se ha empleado el entorno de desarrollo NetBeans [9] de Java y el programa servidor que corre en la tarjeta se ha realizado en el propio XPS.

En los siguientes epígrafes se detalla cada una de estas herramientas utilizadas ofreciendo un resumen de sus características y las razones de su utilización.

3.1. Hardware de desarrollo

Para el desarrollo de este Proyecto Fin de Carrera se ha utilizado la tarjeta “Xilinx XUPV5-LX110T Evaluation Platform” [18] propiedad del grupo de Diseño Microelectrónico y Aplicaciones. Esta tarjeta normalmente se emplea para desarrollos de todo tipo de proyectos de investigación, para prototipos y variados experimentos. Se ha empleado esta tarjeta por ser barata y por disponer de todos los elementos necesarios en la nueva arquitectura. Está formada por los siguientes componentes:

- FPGA: Virtex 5 XC5VLX110T.
- Periféricos de Entrada/Salida:
  - Entrada de Audio - Línea y Micrófono.
  - Salida de Audio - Línea, Amplificador, SPDIF y Altavoz piezoeléctrico.
  - Entrada de Video.
  - Salida de Video en DVI y VGA.
  - DIP Switch de 8 interruptores.
  - 8 LEDs.
  - 5 Botones pulsadores.
  - Interfaz Ethernet 10/100/1000 Mbps.
  - PCI Express®.
- Memoria:
  - DDR2 SODIMM (256 MB).
  - ZBT SRAM (1 MB).
  - Linear Flash (32 MB).
- Comunicaciones:
  - Interfaz de programación por JTAG.
  - 2 Puertos USB 2.0 host.
  - 2 PS/2 para ratón y teclado.
  - RJ-45 - 10/100/1000 Mbps.
  - RS-232.

La tarjeta XUPV5-LX110T proporciona un hardware capaz de implementar prácticamente cualquier aplicación. Entre los componentes incorporados destaca el módulo de memoria SODIMM de 256 MB compatible con el procesador embebido MicroBlaze, esta unión permite crear aplicaciones mono y multi-procesador de uso intensivo de grandes cantidades de memoria. Otro componente que resulta de mucha utilidad es el lector de tarjetas de memoria “Compact Flash”, con la que es posible formatear tarjetas, utilizando diferentes sistemas de ficheros, y almacenar en ellas archivos necesarios para la aplicación.

La FPGA que incorpora la Tarjeta posee unas prestaciones que la colocan por encima de muchas otras soluciones hardware. Un extracto de sus características se detallan en la tabla 3.1 comparándola con las mismas características de una Virtex4 también utilizada en el grupo DMA.

Nótese cómo el numero de Slices de la FPGA de la familia Virtex4 es mayor que el de la Virtex5. Esto es debido al cambio generacional producido entre estas familias de

Tabla 3.1: *Comparación de características de FPGAs*

Device	*Row x Col*	*Logic Cells*	*Slices*	*RAM Dist*	*18Kb Blocks*	*Block RAM*
XC4VLX60	*128 x 52*	*59.904*	*26.624*	*416Kb*	*160*	*2.880Kb*
XC5VLX110T	*160 x 54*	*110.592*	*17.280*	*1.120Kb*	*296*	*5.328Kb*

dispositivos. Los Slices que posee la familia Virtex4 contienen 2 LUTs y 2 Biestables, en cambio, en la siguiente generación poseen 4 LUTs y 4 biestables. Por ello aunque contenga menos Slices, la cantidad de lógica que se puede generar con ellos es mucho mayor.

Se puede obtener más información en la web de la familia de FPGAs Virtex5 de Xilinx [17]

3.2. Herramientas Hardware

En esta sección se detallan los programas utilizados para el desarrollo del hardware de este proyecto. Entre ellos se encuentran el software XPS (Xilinx Platform Studio), el sintetizador Synplify, el gestor de proyectos hardware ISE y el simulador ModelSim. Todas estas herramientas, salvo el sintetizador Synplify, son productos software del fabricante de dispositivos electrónicos Xilinx. La elección de este software viene condicionada al fabricante, y por tanto, existe poca posibilidad de usar otros productos. Para otros fabricantes, como Altera o Atmel, existen programas propietarios similares.

3.2.1. Xilinx Platform Studio

El software XPS o Xilinx Platform Studio (Figura 3.1) es parte del EDK o Embebed Development kit. Con este software se genera un sistema Microblaze y se programa el código C que ejecutará. Es capaz, además, de generar aplicaciones en entornos multiprocesador de forma visual.

XPS da la posibilidad a los desarrolladores hardware de crear sistemas con procesador embebido altamente adaptables a la aplicación concreta sin más que pulsar unos botones.

El sistema Microblaze no es más que un microprocesador empotrado en un dispositivo lógico programable. Puede estar programado en la memoria de configuración o también puede ser un módulo disponible físicamente en silicio dentro de la FPGA. Algunas versiones de las familias más avanzadas llegan a contar con varios de estos módulos dentro del dispositivo, quedando su uso a disposición del usuario.

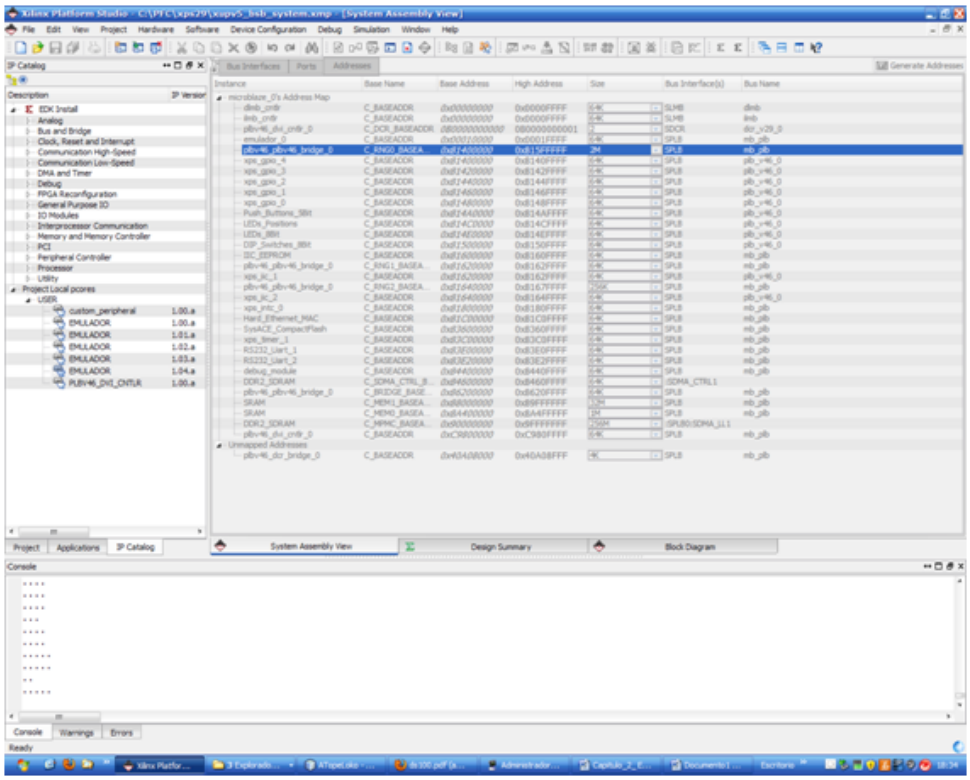


Figura 3.1: *Captura de pantalla de la aplicación XPS.*

Con el XPS, además, se pueden crear periféricos para el Microblaze. Estos periféricos son circuitos diseñados por el usuario a los que se les coloca una interfaz específica y son conectados al microprocesador. De esta manera desde el Microblaze es posible controlar, gestionar y verificar el funcionamiento de circuitos en la propia FPGA. XPS cuenta con un sistema de gestión de periféricos con el que se puede asignar rangos de direcciones, conectar puertos específicos a interrupciones y conectar a los diferentes buses con los que puede contar el sistema.

Las características fundamentales de este software se listan a continuación.

- Catálogo de IP´s (Intellectual Property).
- Altamente integrado con ISE.
- Creación e importación de IP´s propios y de terceros de forma simple.
- Documentación del diseño automática.
- Organización eficaz de los archivos de código fuente.

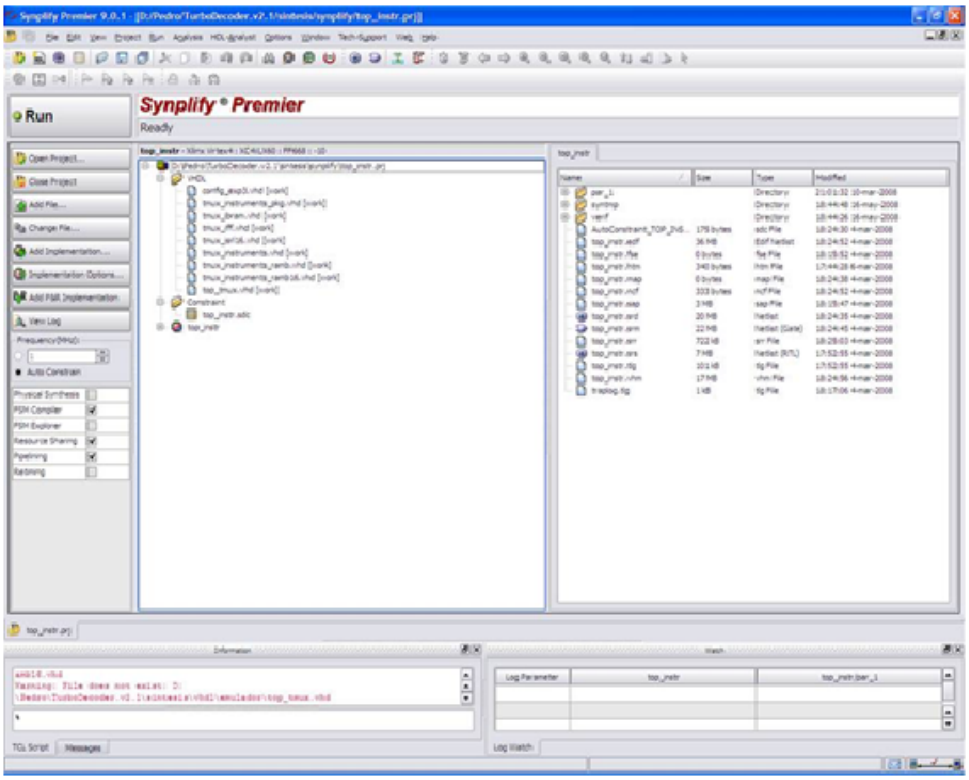


Figura 3.2: *Captura de pantalla de la aplicación Synplify.*

3.2.2. Synplify

Synplify es un software (Propiedad de Synopsys) de síntesis independiente de la plataforma. Este software se ha utilizado en la fase de generación de la netlist del circuito bajo pruebas. Para poder inyectar fallos en un circuito necesitamos cambiar sus elementos de memoria por unos componentes creados en el DMA que contienen la lógica necesaria para provocar bitflips y hacer un seguimiento del estado del fallo inyectado. Estos componentes son llamados “instrumentos” y la tarea del cambio de unos por otros se denomina “instrumentación”. Para facilitar la modificación se genera una netlist en la que queda todo el circuito reducido a celdas lógicas y biestables en un solo archivo de texto. Para generar esta netlist usamos el programa Synplyfy (Figura 3.2), que no es más que un sintetizador externo e independiente del fabricante. Transforma código de descripción hardware (en este caso VHDL) de alto nivel y jerarquía vertical en código de bajo nivel (nivel de puerta) y jerarquía plana. Las características más importantes se listan a continuación [10].

- Integración con herramientas de place and route de diferentes fabricantes de FPGAs (EDK, SoPC Builder).

- Mapeado aplicado al dispositivo concreto para asegurar una implementación optima e independiente de la tecnología.

- Capacidades para sintetizar circuitos optimizando area, consumo o velocidad de reloj máxima.
- integración con módulos Altera megafunctions y Xilinx COREGen que permiten optimizaciones a nivel de sistema.
- VHDL, Verilog, SystemVerilog, VHDL 2008.
- Soporte para dispositivos de todos los fabricantes y sus diferentes familias de FPGA: Altera, Lattice, Microsemi (formerly Actel), SiliconBlue y Xilinx.

3.2.3. Xilinx ISE

El software Xilinx ISE permite organizar jerárquicamente los diseños VHDL y generar la descripción del hardware que se desea. Además enlaza directamente con el simulador ModelSim para verificar el comportamiento del circuito diseñado. ISE incluye diversas herramientas y asistentes para desarrollar diferentes tipos de harware, entre ellos destaca el Core Generator, que genera módulos parametrizables optimizados para satisfacer multitud de necesidades [20]. De esta forma, Xilinx garantiza que el desarrollo de proyectos sea lo más rápido posible.

Este programa, entre otras bondades, permite realizar simulaciones comportamentales, postlayout y postmap según el grado de fidelidad requerido. El resto de características se detallan a continuación [19].

- Project Navigator.
- Generador de módulos CORE Generator.
- Herramienta PlanAhead, cuya metodología eleva la calidad y el desempeño del diseño.
- Reconfiguración parcial, que permite resintetizar y programar una parte del diseño, dejando el resto sin cambios. Esta técnica es util para circuitos en los que una parte es fija y otra cambia, ya sea por pruebas o por la naturaleza de la aplicación. Permite reducir enormemente los tiempos de síntesis.
- Optimización de consumo y velocidad de forma configurable. Generalmente en los diseños hay que decantarse por una u otra opción, ya que una alta velocidad conlleva un consumo elevado en la mayoría de los casos y viceversa.
- ISE Simulator (ISim).
- Síntesis XST.

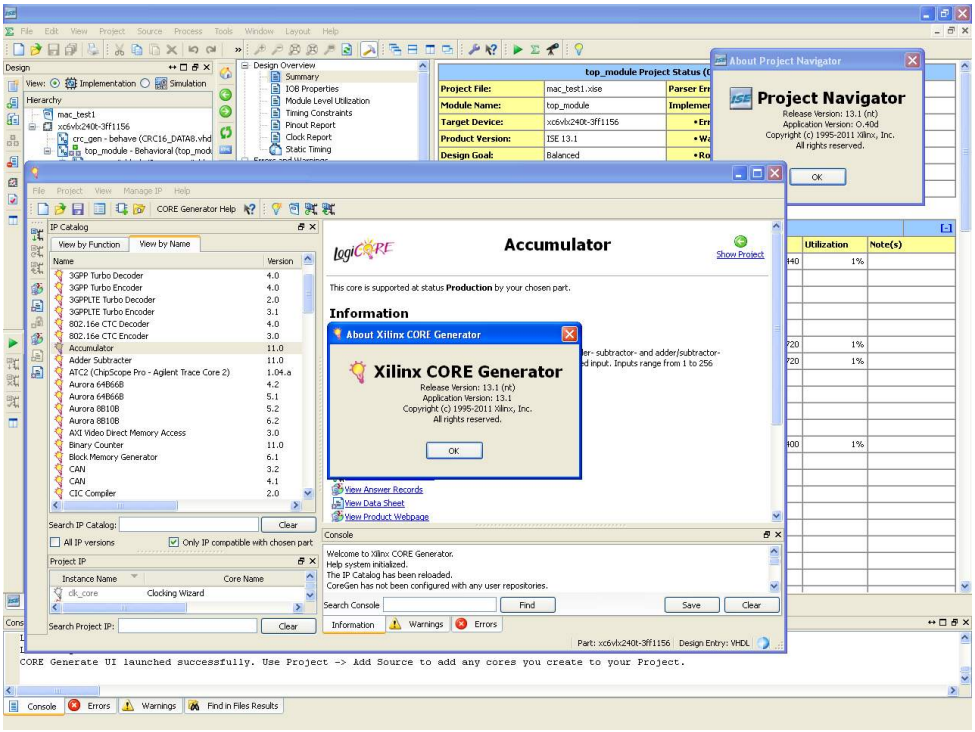


Figura 3.3: *Captura de pantalla de la aplicación Xilinx ISE.*

3.2.4. ModelSim

ModelSim es un programa para desarrollar circuitos digitales que cuenta con un simulador y depurador de código. Tanto en Verilog, en VHDL como en ambos, mezclando, en un mismo diseño, bloques realizados en ambos lenguajes. Este software permite simular el comportamiento que tendrá el circuito una vez programado en la FPGA. Esta herramienta puede manejarse también desde la línea de comandos, que aporta versatilidad a la hora de generar scripts que automaticen el proceso además de ser muy potente y eficaz para grandes proyectos.

ModelSim está desarrollado por MentorGraphics y cuenta con una versión gratuita para estudiantes. Las características fundamentales son las siguientes [6].

- Soporte para diseños en lenguaje VHDL y Verilog.
- Interfaz grafica de usuario inteligente y fácil de usar e integrar con TCL.
- Gestor de proyectos y código fuente.
- Incluye diferentes plantillas y asistentes para generar módulos de forma sencilla.
- Depurador y simulador del hardware muy potente, permitiendo colocar cursores, agrupar señales, realizar búsquedas, control de la base en la que se representan las señales, puntos de parada de la ejecución (breakpoints), etc.

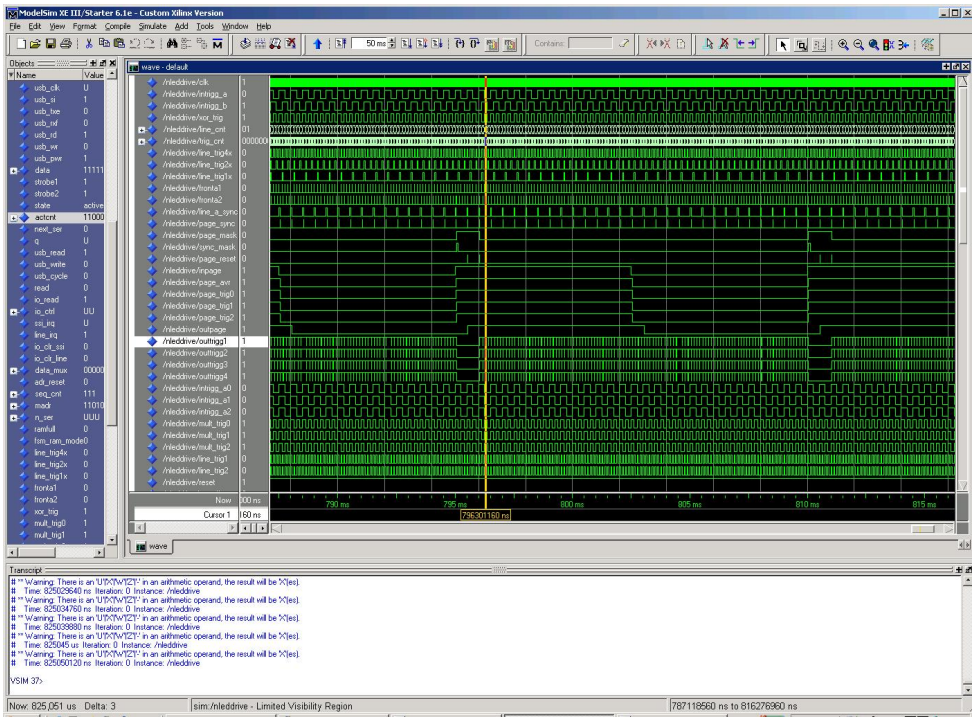


Figura 3.4: *Captura de pantalla de la aplicación ModelSim.*

3.3. Herramientas software

Para dar funcionalidad a un hardware como el diseñado en este Proyecto Fin de Carrera es necesario desarrollar un software que sea capaz de controlar dicho hardware. Para crear estas aplicaciones se han usado diferentes herramientas, entre las que se ha optado por entornos de desarrollo. Esta elección se basa en que estos entornos condensan todas las necesidades de un desarrollador a la hora de implementar una aplicación y permiten una velocidad de diseño elevada.

En contraposición, abordar el diseño software de una aplicación actual sin contar con un buen entorno de desarrollo eleva varios órdenes de magnitud la complejidad, siendo necesarios varios programas como un compilador, un gestor de archivos, un editor de texto, un programa de diseño UML, un depurador, etc. A continuación se describen los entornos de desarrollo usados en este Proyecto Fin de Carrera y sus principales características.

3.3.1. NetBeans

NetBeans (Figura 3.5) constituye un entorno de desarrollo para aplicaciones de escritorio, móviles y web, con soporte para múltiples lenguajes como Java, PHP, C/C++, Delphy, Ajax, etc. Es un proyecto de código abierto con una gran base de usuarios. Sun Microsis-

tems fundó el proyecto NetBeans en Junio de 2000 para promocionar Java en el universo de los desarrolladores, ya que en ese momento no existían IDE´s serios para este lenguaje. A día de hoy, Oracle, propietaria de Sun Microsistems, continúa siendo patrocinador principal de los proyectos NetBeans [11].

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados "módulos". Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Este IDE se ha usado para la construcción del programa cliente en lenguaje Java. Existen multitud de entornos de desarrollo, se ha elegido este por la experiencia del autor con la construcción de aplicaciones en dicho IDE. Las características más destacadas son las siguientes:

- Facilidad para crear interfaces graficas de usuario de gran complejidad. Tiene dos modos de construcción de interfaces gráficas, por un lado es posible generarlas arrastrando y soltando los componentes en el lugar deseado, para después adaptar sus dimensiones a las necesidades. Por otro lado es posible generar una GUI utilizando lo que en Java se conoce como "layout". Esta técnica adapta los controles en función del sitio disponible, del tamaño preferible del control y del tamaño de la ventana. Así los controles se redistribuyen y adaptan por la ventana según esta cambia de tamaño. En el presente Proyecto Fin de Carrera se ha empleado esta segunda opción.

- Generación de código automático, con lo que es posible escribir grandes cantidades de código repetitivo con unas pocas pulsaciones de ratón. Entre otras cosas, es capaz de generar constructores, métodos get y set, sobrecarga de métodos heredados como "toString()", "'hasCode()", "equals()" y otros definidos por el usuario.

- Desarrollo del software mediante UML para posteriormente generar el esqueleto de la aplicación. Esta característica es de gran utilidad en proyectos de cierta envergadura, debido a que permite diseñar la aplicación a nivel de bloques funcionales y relaciones entre ellos para posteriormente generar código Java compilable a falta de rellenar la funcionalidad específica. Todo esto permite acelerar enormemente el diseño e implementación de aplicaciones grandes, dejando sólo al programador los ladrillos fundamentales con los que se construirá el código completo.

- Depurador y compilador integrado. Durante la escritura de código Java en el entorno NetBeans el IDE va ofreciendo ayuda al programador según programa su aplicación. Existen diferentes niveles de ayuda, el más útil consiste en completar las sentencias que el usuario va escribiendo. No sólo completa las funciones y las bibliotecas nativas de Java, sino también las del código escrito en el propio programa en desarrollo.

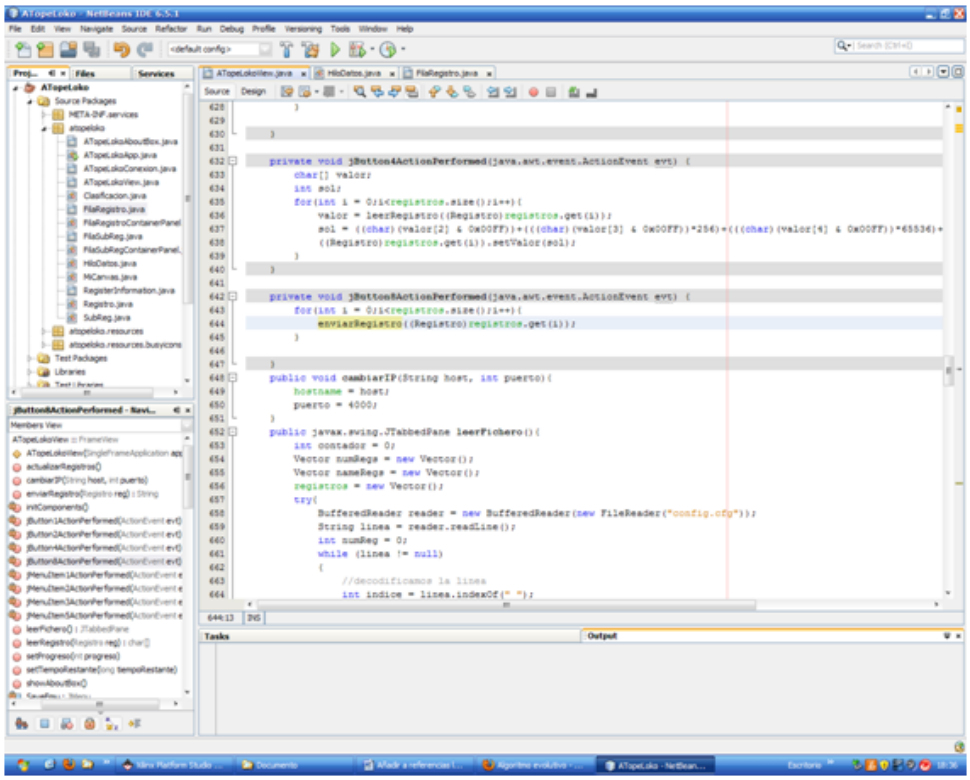


Figura 3.5: *Captura de pantalla de la aplicación Netbeans.*

Otro nivel de ayuda ofrece un compilado en tiempo real de la aplicación, indicando constantemente posibles errores de sintaxis y sus posibles soluciones.

- Organización eficaz de los archivos de código fuente.

3.3.2. XPS

El XPS, ya comentado en el apartado de herramientas hardware, también se ha usado para el desarrollo de la aplicación que ejecuta el Microblaze. Se ha elegido este software en vez de otros IDE's como Eclipse (exportando el diseño al SDK en formato XML) porque el programa creado es bastante compacto y no requiere de complicadas herramientas para hacer viable su finalización. El tiempo invertido en configurar el proyecto y aprender la plataforma es mucho mayor que desarrollar la aplicación directamente sobre XPS.

A pesar de esto, el editor de código software del Xilinx Platform Studio ofrece ayudas al programador alejándolo de un simple editor de texto plano. Entre las capacidades del editor están:

- Coloreado y resaltado de tokens reservados del compilador.

- Compilador y depurador integrado.
- Gestión de proyectos.

Capítulo 4

DESARROLLO HARDWARE DEL SERVIDOR DE EMULACIÓN

4.1. Arquitectura

La tarjeta utilizada durante la realización de este Proyecto Fin de Carrera posee múltiples circuitos y conexiones. Por ello responde al nombre de “Tarjeta de evaluación”, para que sea posible realizar casi cualquier proyecto con ella y sirva de muestra de lo que es posible realizar con un sistema similar. Estas tarjetas tienen esa finalidad, en el caso de proyectos grandes con presupuestos elevados se diseñarían tarjetas con únicamente los recursos necesarios y se descartaría el resto de componentes. El resultado es una tarjeta específica para la aplicación concreta con un tamaño y costos reducidos. Como el propósito de este proyecto es usar una tarjeta genérica que posee el departamento de Tecnología electrónica se ha pasado por alto este aspecto.

La arquitectura física de la tarjeta de evaluación viene descrita en la figura 4.1. En ella se aprecian los circuitos integrados más importantes que la componen con sus respectivos fabricantes y numeros de referencia. Concretamente, para el caso de la UART (Universal Asynchronous Receiver-Transmitter) la tarjeta cuenta con un dispositivo (Analog Devices 3202A) que transforma las señales lógicas de salida de la FPGA a los voltajes y corrientes del estandar RS-232. Por otro lado, el controlador Ethernet funciona de manera similar, cuenta con un circuito integrado (Marvell 88E1111) cuyas entradas son las salidas que genera el periférico EMAC y obtiene las señales eléctricas que viajarán por el cable de red. La tarjeta posee otros muchos módulos con los que está conectada la FPGA que pueden deducirse de la lista de características expuesta en el punto 2.1.

La arquitectura del sistema hardware queda descrita en la figura 4.2. Sigue el esquema genérico de diseño ofrecido por Xilinx en su asistente. Consta de un procesador mononú-

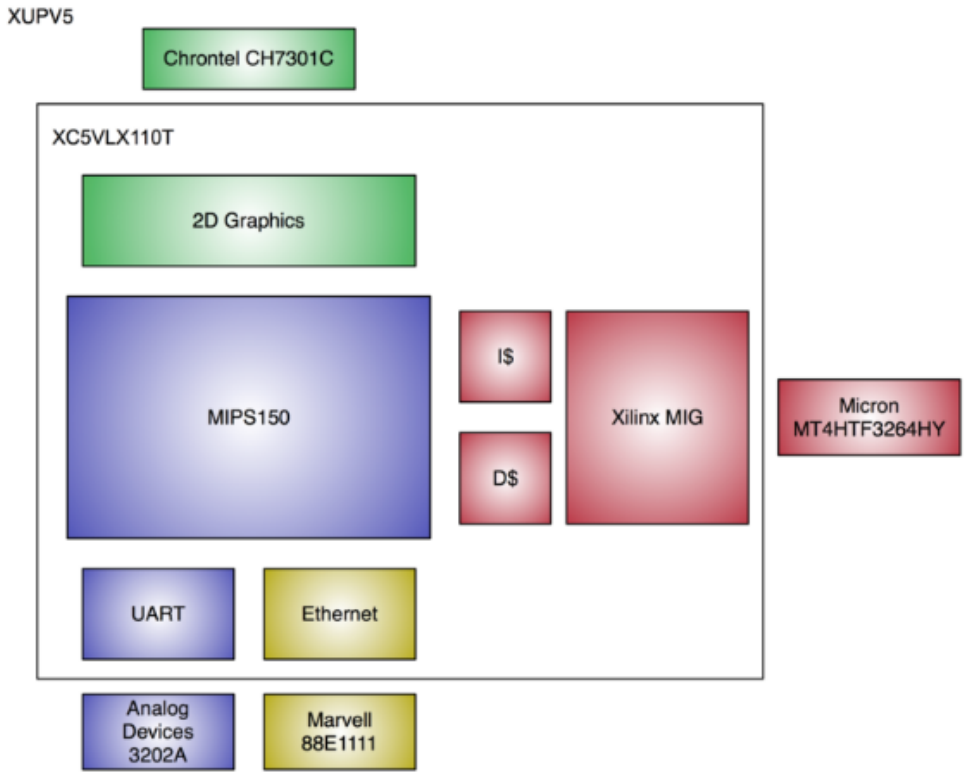


Figura 4.1: *Descripción de los módulos y componentes de la tarjeta XUPV5.*

cleo conectado a diferentes periféricos por medio de un bus PLB. Este método de conexión permite al microprocesador embebido leer y escribir datos en los periféricos de forma secuencial, es decir, no permite accesos simultáneos al ser un bus físico compartido. Es una limitación de la arquitectura. El bus PLB usado en este Proyecto Fin de Carrera se corresponde con la versión 4.6, que consiste en una unidad de control, un timer watchdog así como un DRC (Device Control Register) opcional para tener acceso a los registros de estado del bus. El arbitro encargado del control del bus y la lógica necesaria, es la parte más importante del bus [13].

Los periféricos, a su vez, pueden contar con conexiones a puertos externos de la FPGA para controlar otros circuitos o hardware. El emulador, que constituye un periférico más del procesador MicroBlaze, no incorpora conexiones a puertos externos de la FPGA porque no hay necesidad funcional. Sin embargo, como mejora futura se propone incorporar al emulador control sobre los leds de propósito general con los que cuenta la tarjeta. De esta manera sí que contaría con conexiones externas. Merece la pena resaltar que la arquitectura de esta solución admite gran cantidad de ampliaciones respondiendo a un sistema altamente escalable, tal y como pretendía Xilinx. Es posible incorporar decenas de periféricos al bus PLB, incluso en el momento de saturarlo puede añadirse un segundo y tercer bus PLB con puentes entre ellos, garantizando la conectividad de todos ellos. Concretamente, el diseño cuenta con un controlador Ethernet, RAM y RS-232 además de otros circuitos integrados que no se les da uso en el contexto de este proyecto.

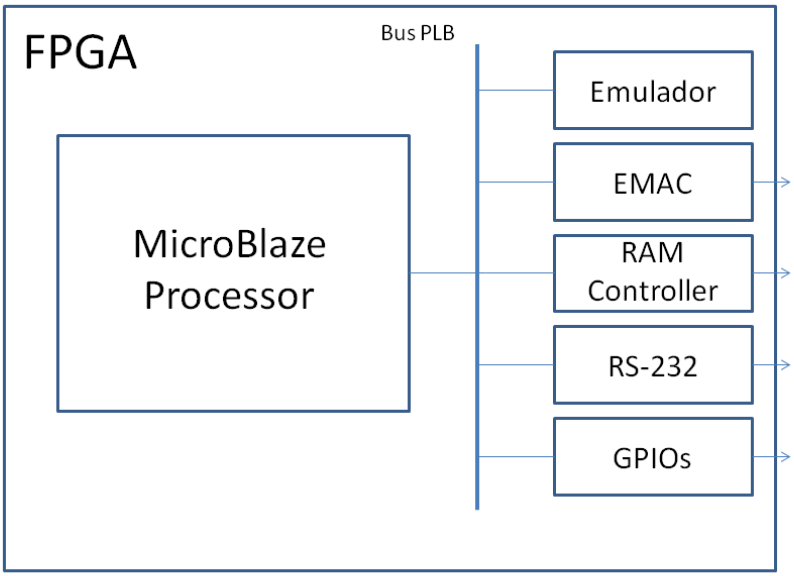


Figura 4.2: *Arquitectura y organización interna de los módulos de la FPGA.*

A continuación quedan explicados los periféricos más importantes usados por el sistema generado.

4.1.1. Periférico EMAC

El periférico Ethernet Media Access Controller (EMAC) ofrece la posibilidad de enviar y recibir datos por el interfaz físico Ethernet. Junto con las bibliotecas lwIP, adaptadas por Xilinx, se genera una potente plataforma de transmisión y recepción de datos TCP/IP con la especificación 802.3 del año 2008. Algunas de las características más importantes de este periférico son las siguientes [14].

- Velocidad configurable automática 10/100/1000 Mbps.
- Comunicación Duplex
- Soporte para MII (Media Independent Interface), GMII (Gigabit Media Independent Interface) y RGMII (Reduced Gigabit Media Independent Interface).
- Soporte para redes VLAN.
- Control de flujo configurable que evita pérdida de paquetes IP por congestión. El periférico es capaz de detectar el estado de sobrecarga del otro extremo de la conexión

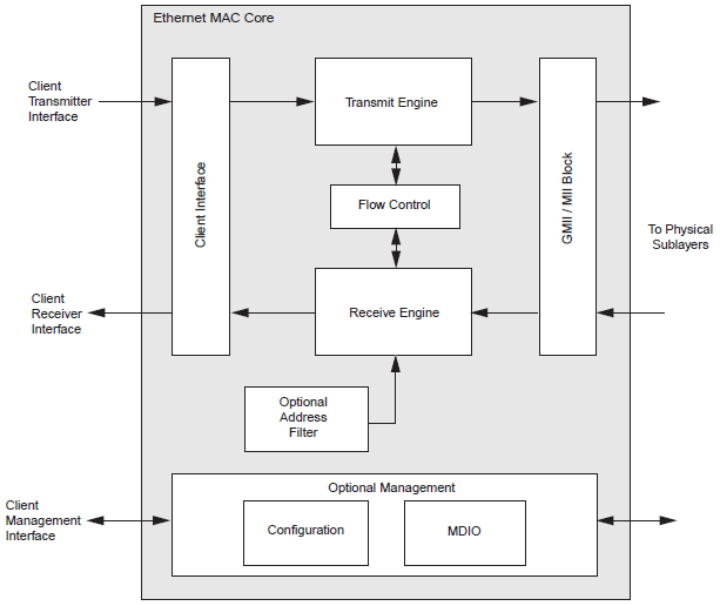


Figura 4.3: *Arquitectura del periférico EMAC.*

y en función de ello configura el tamaño de ventana de transmisión según el mecanismo de “Ventana deslizante”. Ampliándolo para aumentar la velocidad de transmisión y disminuyéndolo para limitarla. Para más información se recomienda revisar la RFC 1323 [7]

4.1.2. Periférico de RAM

La memoria principal que usa el procesador MicroBlaze puede estar ubicada en la tarjeta de desarrollo como memoria externa o puede formarse con bloques de RAM internos de la FPGA. En este proyecto se ha optado por la segunda opción, ya que el asistente de Xilinx genera automáticamente la lógica necesaria para el correcto funcionamiento sin necesidad de configurar el controlador de RAM externo (en un módulo SODIMM de 256MB). El tamaño de esta memoria RAM es de 64KB, suficiente para la aplicación que nos ocupa. Además, la velocidad de la RAM interna es sensiblemente mayor que la externa al estar confinada dentro del propio chip.

Los periféricos utilizados cuentan con sus propios bloques de memoria que en el momento de la síntesis se mapean en posiciones accesibles en el código. Por este motivo no es necesaria una RAM mayor.

4.1.3. Periférico controlador RS232

El asistente de Xilinx a la hora de configurar un nuevo proyecto con MicroBlaze ofrece la posibilidad de insertar un controlador RS-232 en la solución final. Es la forma que se ha elegido para instalarlo, automáticamente genera drivers y queda listo para usarlo de forma muy sencilla. Desde el código C se accede rápidamente al periférico mediante las llamadas típicas “read” y “write”.

4.2. Transferencia de Datos

Los datos se transmiten entre el procesador MicroBlaze y los periféricos a través de un bus PLB compartido. Cuando el MicroBlaze pretende leer o escribir una dirección de memoria los datos se leen o escriben del bus.

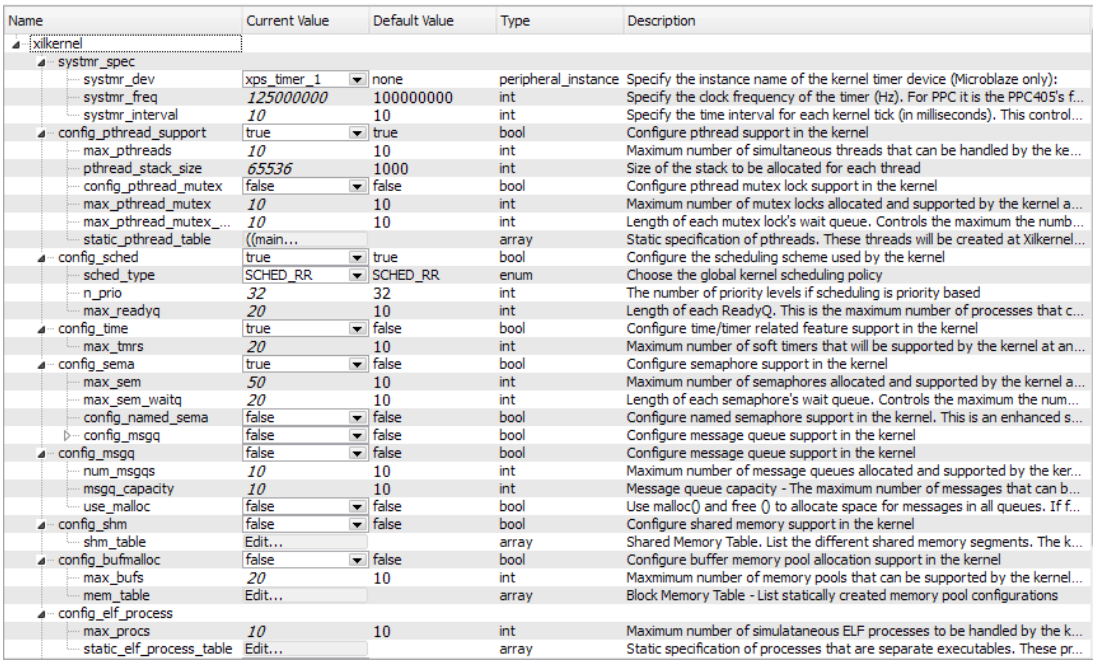
La forma de comunicación entre el MicroBlaze y sus periféricos es a través de registros, memoria compartida y/o FIFO´s. Estos mecanismos se encuentran implementados en la lógica de los periféricos, de hecho el software EDK ofrece un asistente con el que añadir cualquiera de ellos. Estos sistemas de comunicación se encuentran mapeados en memoria, de forma que los programas que ejecute el MicroBlaze tendrán acceso a escribir y leer el estado o configuración de los periféricos a través de escribir y leer datos en direcciones de memoria.

El mapeo de estas direcciones de memoria se realiza en un paso previo a la construcción de las aplicaciones, generando un archivo con sentencias “define” para que el compilador de C sustituya nombres fáciles de recordar por las direcciones de memoria.

4.3. Configuración de Microblaze y Xilkernel

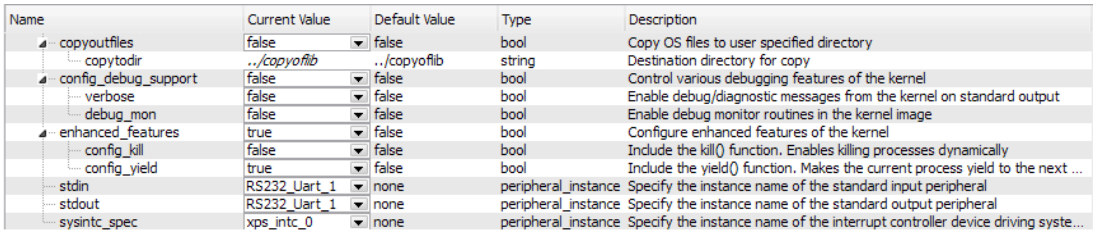
Xilkernel, en su versión 4.0, ofrece servicios POSIX como un planificador, gestión de hilos de procesamiento en paralelo, sincronización, paso de mensajes y timers. Además se incluyen bibliotecas que facilitarán diversas tareas y que se integran perfectamente con Xilkernel. Las bibliotecas activadas para este Proyecto Fin de Carrera son:

- Xilfatfs (versión 1.00.a): provee rutinas de lectura y escritura para acceder a un sistema de archivos FAT16 y FAT32.
- Xilmfs (versión 1.00.a): Xilinx memory file system
- Lwip (versión 3.00.a): Biblioteca ya comentada en secciones anteriores que integra un stack tcp/ip en la solución.



Name	Current Value	Default Value	Type	Description
xilkernel				
systmr_spec				
systmr_dev	xps_timer_1	none	peripheral_instance	Specify the instance name of the kernel timer device (Microblaze only):
systmr_freq	125000000	100000000	int	Specify the clock frequency of the timer (Hz). For PPC it is the PPC405's f...
systmr_interval	10	10	int	Specify the time interval for each kernel tick (in milliseconds). This control...
config_pthread_support	true	true	bool	Configure pthread support in the kernel
max_pthreads	10	10	int	Maximum number of simultaneous threads that can be handled by the ke...
pthread_stack_size	65536	1000	int	Size of the stack to be allocated for each thread
config_pthread_mutex	false	false	bool	Configure pthread mutex lock support in the kernel
max_pthread_mutex	10	10	int	Maximum number of mutex locks allocated and supported by the kernel a...
max_pthread_mutex_...	10	10	int	Length of each mutex lock's wait queue. Controls the maximum the numb...
static_pthread_table	((main...		array	Static specification of pthreads. These threads will be created at Xilkernel...
config_sched	true	true	bool	Configure the scheduling scheme used by the kernel
sched_type	SCHED_RR	SCHED_RR	enum	Choose the global kernel scheduling policy
n_prio	32	32	int	The number of priority levels if scheduling is priority based
max_readyq	20	10	int	Length of each ReadyQ. This is the maximum number of processes that c...
config_time	true	false	bool	Configure time/timer related feature support in the kernel
max_tmrs	20	10	int	Maximum number of soft timers that will be supported by the kernel at an...
config_sema	true	false	bool	Configure semaphore support in the kernel
max_sem	50	10	int	Maximum number of semaphores allocated and supported by the kernel a...
max_sem_waitq	20	10	int	Length of each semaphore's wait queue. Controls the maximum the num...
config_named_sema	false	false	bool	Configure named semaphore support in the kernel. This is an enhanced s...
config_msgq	false	false	bool	Configure message queue support in the kernel
config_msgq	false	false	bool	Configure message queue support in the kernel
num_msgqs	10	10	int	Maximum number of message queues allocated and supported by the ker...
msgq_capacity	10	10	int	Message queue capacity - The maximum number of messages that can b...
use_malloc	false	false	bool	Use malloc() and free () to allocate space for messages in all queues. If f...
config_shm	false	false	bool	Configure shared memory support in the kernel
shm_table	Edit...		array	Shared Memory Table. List the different shared memory segments. The k...
config_bufmalloc	false	false	bool	Configure buffer memory pool allocation support in the kernel
max_bufs	20	10	int	Maxmimum number of memory pools that can be supported by the kernel...
mem_table	Edit...		array	Block Memory Table - List statically created memory pool configurations
config_elf_process				
max_procs	10	10	int	Maximum number of simulataneous ELF processes to be handled by the k...
static_elf_process_table	Edit...		array	Static specification of processes that are separate executables. These pr...

Figura 4.4: *Configuración de Xilkernel.*



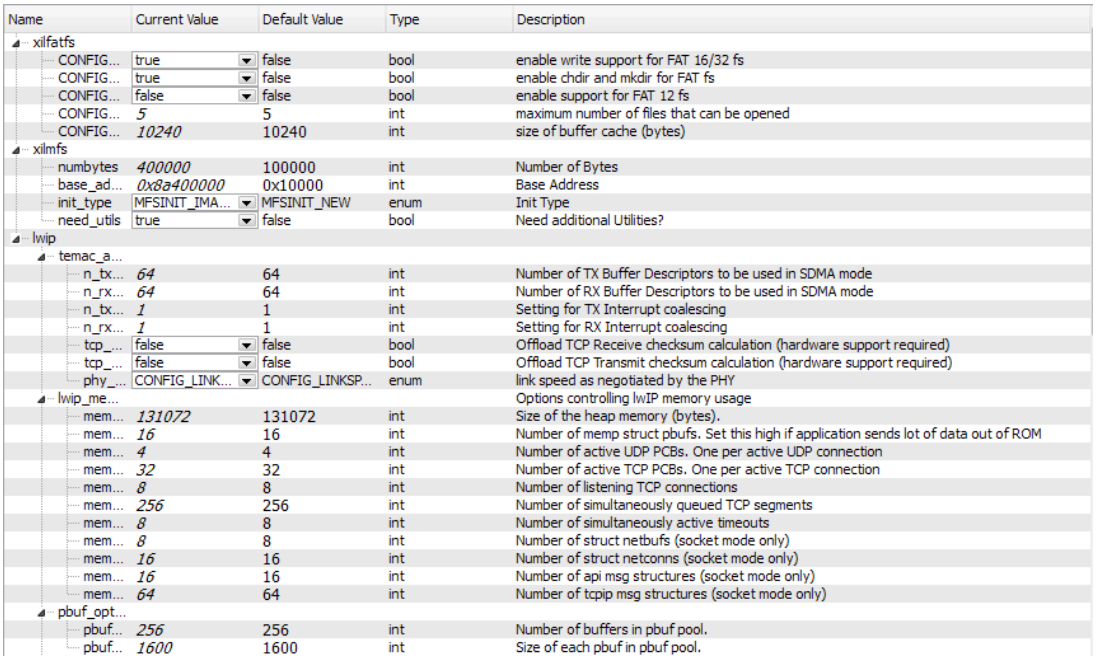
Name	Current Value	Default Value	Type	Description
copyoutfiles	false	false	bool	Copy OS files to user specified directory
copytodir	.../copyoflib	.../copyoflib	string	Destination directory for copy
config_debug_support	false	false	bool	Control various debugging features of the kernel
verbose	false	false	bool	Enable debug/diagnostic messages from the kernel on standard output
debug_mon	false	false	bool	Enable debug monitor routines in the kernel image
enhanced_features	true	false	bool	Configure enhanced features of the kernel
config_kill	false	false	bool	Include the kill() function. Enables killing processes dynamically
config_yield	true	false	bool	Include the yield() function. Makes the current process yield to the next ...
stdin	RS232_Uart_1	none	peripheral_instance	Specify the instance name of the standard input peripheral
stdout	RS232_Uart_1	none	peripheral_instance	Specify the instance name of the standard output peripheral
sysintc_spec	xps_intc_0	none	peripheral_instance	Specify the instance name of the interrupt controller device driving syste...

Figura 4.5: *Configuración de Xilkernel.*

La configuración del propio Xilkernel es muy extensa, ya que es posible configurar detalles muy concretos en cada una de sus características. Un extracto de ella se muestra en las figuras 4.4 y 4.5. En ellas se pueden encontrar datos acerca de las capacidades multihilo, comunicaciones entre hilos (semáforos, mutex, memoria compartida y paso de mensajes), así como propiedades del planificador y timers.

A modo de resumen, se permiten como máximo 10 hilos simultáneos con 32 niveles de prioridad y una pila de 65536 posiciones cada uno. Se configuran también 20 timers para propósito general.

Por otro lado, otro aspecto importante de la configuración del sistema consiste en el ajuste de los parámetros de las librerías antes comentadas (lwip, xilfatfs y xilmfs). Al igual que en el caso de la configuración relativa a Xilkernel, las posibilidades de personalización son muy grandes. En las figuras 4.6 y 4.7 se aprecian los valores concretos utilizados.



Name	Current Value	Default Value	Type	Description
xilfatfs				
CONFIG...	true	false	bool	enable write support for FAT 16/32 fs
CONFIG...	true	false	bool	enable chdir and mkdir for FAT fs
CONFIG...	false	false	bool	enable support for FAT 12 fs
CONFIG...	*5*	5	int	maximum number of files that can be opened
CONFIG...	*10240*	10240	int	size of buffer cache (bytes)
xilmfs				
numbytes	*400000*	100000	int	Number of Bytes
base_ad...	*0x8a400000*	0x10000	int	Base Address
init_type	MFSINIT_IMA...	MFSINIT_NEW	enum	Init Type
need_utils	true	false	bool	Need additional Utilities?
lwip				
temac_a...				
n_tx...	*64*	64	int	Number of TX Buffer Descriptors to be used in SDMA mode
n_rx...	*64*	64	int	Number of RX Buffer Descriptors to be used in SDMA mode
n_tx...	*1*	1	int	Setting for TX Interrupt coalescing
n_rx...	*1*	1	int	Setting for RX Interrupt coalescing
tcp_...	false	false	bool	Offload TCP Receive checksum calculation (hardware support required)
tcp_...	false	false	bool	Offload TCP Transmit checksum calculation (hardware support required)
phy_...	CONFIG_LINK...	CONFIG_LINKSP...	enum	link speed as negotiated by the PHY
lwip_me...				Options controlling lwIP memory usage
mem...	*131072*	131072	int	Size of the heap memory (bytes).
mem...	*16*	16	int	Number of memp struct pbufs. Set this high if application sends lot of data out of ROM
mem...	*4*	4	int	Number of active UDP PCBs. One per active UDP connection
mem...	*32*	32	int	Number of active TCP PCBs. One per active TCP connection
mem...	*8*	8	int	Number of listening TCP connections
mem...	*256*	256	int	Number of simultaneously queued TCP segments
mem...	*8*	8	int	Number of simultaneously active timeouts
mem...	*8*	8	int	Number of struct netbufs (socket mode only)
mem...	*16*	16	int	Number of struct netconns (socket mode only)
mem...	*16*	16	int	Number of api msg structures (socket mode only)
mem...	*64*	64	int	Number of tcpip msg structures (socket mode only)
pbuf_opt...				
pbuf...	*256*	256	int	Number of buffers in pbuf pool.
pbuf...	*1600*	1600	int	Size of each pbuf in pbuf pool.

Figura 4.6: *Configuración de Xilkernel.*

La biblioteca más importante y con más opciones configurables es lwip, ya que es imprescindible para enviar datos a través de la interfaz Ethernet. También se han empleado las configuraciones por defecto.

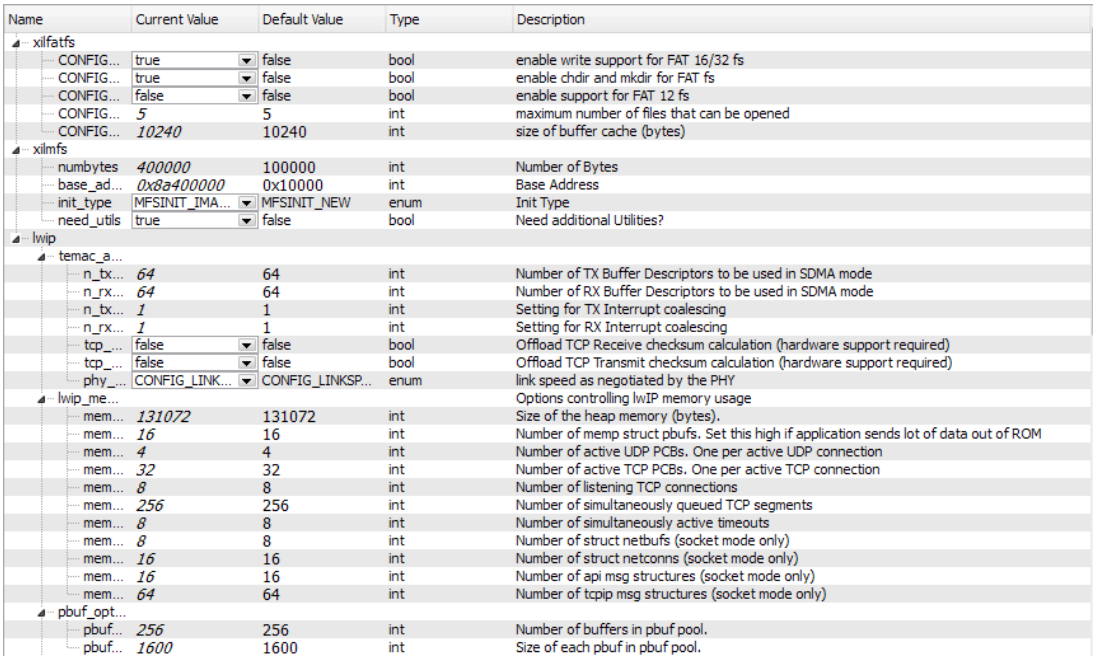
4.4. Implementación del emulador como periférico

La lógica implementada en este proyecto trata de comunicar el bus PLB con el circuito emulador ya descrito. En este circuito de adaptación se pueden diferenciar 2 bloques funcionales, la lógica generada por el asistente de Xilinx y la diseñada por el autor de este documento que se encarga de reordenar los resultados del circuito emulador a la forma de comunicación con el procesador embebido.

4.4.1. Lógica generada por el asistente

Como se indica en la sección anterior, al crear un periférico, el software EDK lanza un asistente que genera una plantilla con la lógica de comunicaciones MicroBlaze-Periférico. En el caso particular que nos ocupa, la lógica generada posee 40 registros y un bloque de memoria compartida de 256 posiciones de 4 bytes (32 bits).

Esta lógica se encarga de proteger de escrituras simultáneas y sincronización del bus



Name	Current Value	Default Value	Type	Description
xilfatfs				
CONFIG...	true	false	bool	enable write support for FAT 16/32 fs
CONFIG...	true	false	bool	enable chdir and mkdir for FAT fs
CONFIG...	false	false	bool	enable support for FAT 12 fs
CONFIG...	*5*	5	int	maximum number of files that can be opened
CONFIG...	*10240*	10240	int	size of buffer cache (bytes)
xilmfs				
numbytes	*400000*	100000	int	Number of Bytes
base_ad...	*0x8a400000*	0x10000	int	Base Address
init_type	MFSINIT_IMA...	MFSINIT_NEW	enum	Init Type
need_utils	true	false	bool	Need additional Utilities?
lwip				
temac_a...				
n_tx...	*64*	64	int	Number of TX Buffer Descriptors to be used in SDMA mode
n_rx...	*64*	64	int	Number of RX Buffer Descriptors to be used in SDMA mode
n_tx...	*1*	1	int	Setting for TX Interrupt coalescing
n_rx...	*1*	1	int	Setting for RX Interrupt coalescing
tcp_...	false	false	bool	Offload TCP Receive checksum calculation (hardware support required)
tcp_...	false	false	bool	Offload TCP Transmit checksum calculation (hardware support required)
phy_...	CONFIG_LINK...	CONFIG_LINKSP...	enum	link speed as negotiated by the PHY
lwip_me...				Options controlling lwIP memory usage
mem...	*131072*	131072	int	Size of the heap memory (bytes).
mem...	*16*	16	int	Number of memp struct pbufs. Set this high if application sends lot of data out of ROM
mem...	*4*	4	int	Number of active UDP PCBs. One per active UDP connection
mem...	*32*	32	int	Number of active TCP PCBs. One per active TCP connection
mem...	*8*	8	int	Number of listening TCP connections
mem...	*256*	256	int	Number of simultaneously queued TCP segments
mem...	*8*	8	int	Number of simultaneously active timeouts
mem...	*8*	8	int	Number of struct netbufs (socket mode only)
mem...	*16*	16	int	Number of struct netconns (socket mode only)
mem...	*16*	16	int	Number of api msg structures (socket mode only)
mem...	*64*	64	int	Number of tcpip msg structures (socket mode only)
pbuf_opt...				
pbuf...	*256*	256	int	Number of buffers in pbuf pool.
pbuf...	*1600*	1600	int	Size of each pbuf in pbuf pool.

Figura 4.7: *Configuración de Xilkernel.*

PLB, así como de ofrecer una interfaz de acceso al bus a una entidad de nivel superior. Esta entidad es la lógica de usuario.

4.4.2. Lógica de adaptación del emulador al MicroBlaze

La lógica implementada contiene los circuitos que ordenan y almacenan las clasificaciones en las posiciones de memoria correspondientes llevando un control de las posiciones insertadas. La lógica diseñada consta de 5 simples módulos fácilmente identificables en la figura 4.8, que muestra el mecanismo usado para colocar las clasificaciones en posiciones de memoria compartida con el MicroBlaze.

El emulador contiene un conjunto de modificaciones que colocan las clasificaciones en una palabra de 32 bits, es decir, 16 clasificaciones. Además, una vez llenado dicho registro se habilita una señal de “write enable”. Fuera del emulador se encuentra un detector de flancos de la señal “write enable” y un contador que incrementa un registro cada vez que se detecta un flanco. A este registro se le llama “indice” y señala la posición de memoria en la que se debe escribir la palabra que contiene el registro “registro out”.

El módulo de control de los registros de estado niega el bit correspondiente al índice actual. De esta forma se puede saber qué registros han sido modificados desde la última lectura.

Con este circuito se consigue que las clasificaciones se vayan ubicando en posiciones

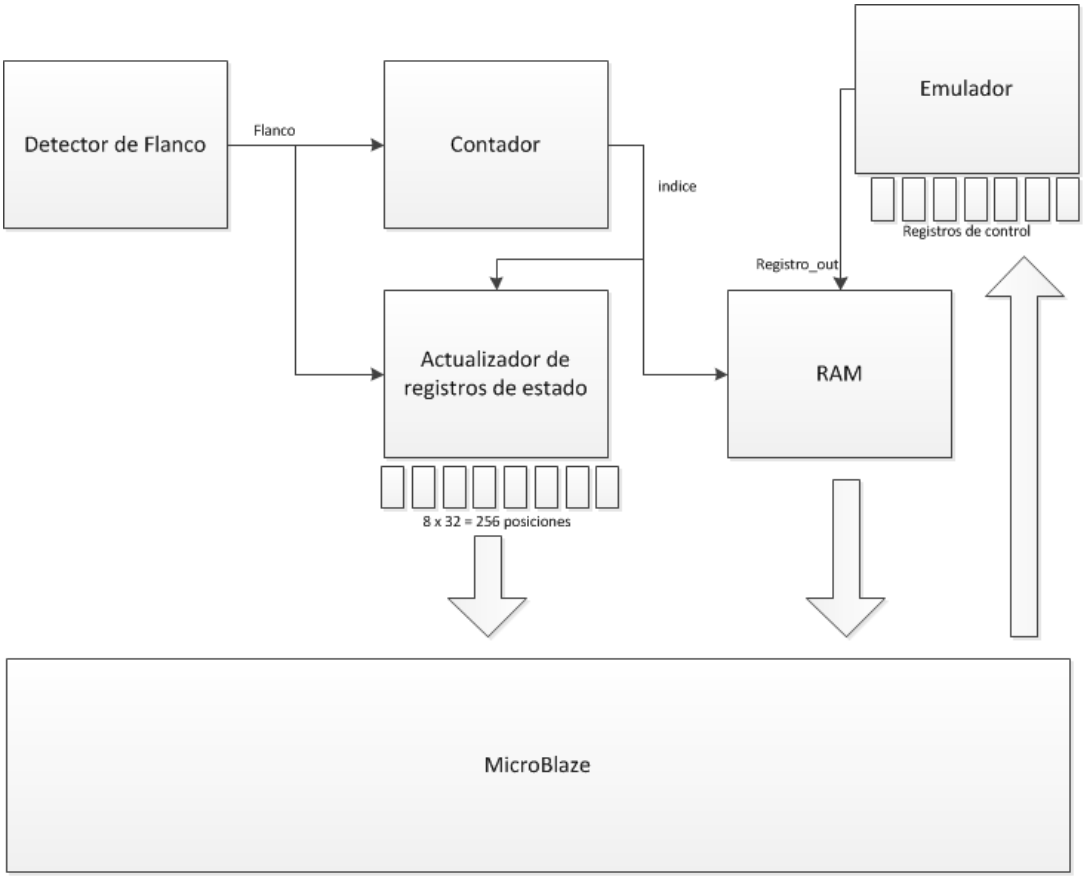


Figura 4.8: *Arquitectura de la lógica de usuario.*

sucesivas de la memoria compartida con el MicroBlaze.

En esta entidad se genera una instancia del propio emulador y se conecta con la lógica de usuario, que a su vez ofrece una interfaz de control y gestión al circuito emulador que se detalla en capítulos anteriores.

4.5. Resultados de síntesis

La adición de este hardware a la FPGA para aumentar la velocidad de las comunicaciones del emulador provoca un aumento en las exigencias de área al sintetizar. El precio que hay que pagar por ese aumento de velocidad se desglosa en las tablas 4.1 y 4.2. Pasa de ocupar un 2-3 % a un 21-23 % del dispositivo utilizado, que como ya ha quedado descrito, se trata de una FPGA Virtex5 5vlx110tff1136-1.

Se trata de un aumento considerable, que hará que la capacidad de sintetizar circuitos grandes sea menor. El emulador contiene una réplica modificada del circuito a probar mas

Tabla 4.1: *Resultados de síntesis de la solución propuesta*

Slice logic utilization	*Cantidad utilizada*	*Utilización*
Slice registers	*15.965 de 69.120*	*23 %*
Slice LUTs	*15.344 de 69.120*	*22 %*

Tabla 4.2: *Resultados de síntesis del emulador sin la solución propuesta*

Slice logic utilization	*Cantidad utilizada*	*Utilización*
Slice registers	*1.779 de 69.120*	*2 %*
Slice LUTs	*2.317 de 69.120*	*3 %*

un conjunto de lógica de control. Con el trabajo realizado en este Proyecto Fin de Carrera se añade más lógica a la que ya generaba el emulador por sí solo. Por tanto, las necesidades de área crecen.

En este caso que nos ocupa el incremento es debido en gran medida a que el circuito a probar tiene unas dimensiones reducidas al ser comparadas con las del sistema MicroBlaze. Por este motivo pierde importancia la diferencia apreciada entre la tabla 4.1 y la 4.2.

Por otro lado, hoy en día, la familia Virtex5 se ha visto superada ampliamente, tanto en tamaño como en velocidad, por generaciones posteriores de FPGAs. Con ello, en dispositivos más modernos y grandes, el tamaño del MicroBlaze será despreciable.

Capítulo 5

SOFTWARE DEL SERVIDOR DE EMULACIÓN

En este capítulo se describen las especificaciones y requisitos del servidor de emulaciones, así como los resultados de los test de velocidad realizados. A continuación se detalla el software desarrollado.

5.1. Especificaciones y requisitos

A continuación se explican los requisitos necesarios para llevar a cabo la mejora del sistema de emulación de fallos desde el punto de vista de la programación del microprocesador embebido MicroBlaze. Planteando los requisitos que debe cumplir el software ejecutado y las medidas que se han tomado para llegar a la solución propuesta.

Tal y como se introdujo a grandes rasgos en el capítulo anterior, las especificaciones del software del servidor son las siguientes:

- Realizar una comunicación Ethernet.
- Recibir los parámetros de la emulación y transmitírselos al Emulador.
- Enviar la orden de inicio de emulación al emulador.
- Lectura del estado del emulador.
- Enviar resultados de emulación al PC en tiempo real a una tasa lo suficientemente elevada como para evitar pérdidas de datos.

Más concretamente, se trata de una pasarela que comunica el software cliente con el emulador. Como se ha comentado en capítulos anteriores, el propósito del presente proyecto es el de mejorar el emulador autónomo de fallos diseñado por el DMA añadiendo una interfaz Ethernet al mismo. Para incorporar esta interfaz se hace uso de un sistema MicroBlaze, ya que ofrece un completo set de bibliotecas C para el envío y recepción de datos vía Ethernet.

En la versión inicial del emulador, desde la que parte este proyecto, la carga de parámetros, inicio de emulación y lectura de resultados se realiza mediante una comunicación serie utilizando el programa TeraTerm o Hyperterminal. Tras este Proyecto Fin de Carrera se pretende que el usuario sea capaz de realizar estas acciones a través de una comunicación Ethernet.

El componente que se encarga de la transmisión de los parámetros y de de dar la orden de arranque al Emulador es el MicroBlaze. Para ello debe ser capaz de recoger los comandos que le son enviados desde el programa cliente y también debe ser capaz de enviar los resultados de la emulación de manera que dicho cliente pueda gestionar estos datos, para posteriormente hacer un tratamiento de los mismos y poder presentarlos correctamente.

A lo largo de la realización del Proyecto Fin de Carrera se han usado las comunicaciones con el puerto serie para la depuración. Estas comunicaciones han sido en todo momento unilaterales, desde el MB hacia el ordenador, es decir, siempre se han transmitido datos desde el MicroBlaze y nunca hacia el. Los datos que se transmiten son diversas trazas para verificar el correcto funcionamiento del hardware y software. Asimismo, también se envía texto informativo de la versión y modo de proceder con el equipo. Entre otros parámetros se muestra la dirección IP y el puerto sobre el que realizar las peticiones de conexión. A diferencia del sistema de partida, no ha sido necesario utilizar el otro sentido de la comunicación serie (PC hacia MicroBlaze) que se empleaba para el envío de comandos al emulador. Este Proyecto Fin de Carrera trata de sustituir ese medio de comunicación por el usado en internet.

Las comunicaciones con el protocolo de comunicación TCP/IP sobre Ethernet se basan en el envío y la recepción de paquetes de longitud variable, pero en este caso concreto se transmitirán y recibirán paquetes de tamaños idénticos y preestablecidos. El tamaño viene indicado en la figura 5.6, de forma que se transmiten 1000 bytes de resultados de la emulación en curso seguidos de 32 bytes de estado. Todo ello forma el bloque de datos de cada paquete TCP/IP transmitido por el servidor de emulación. El funcionamiento por defecto del protocolo TCP/IP abstrae al usuario del tamaño de paquete final, troceando y agrupando datos de forma transparente tanto al receptor como al emisor.

Por último, para concluir este apartado de especificaciones y requisitos, el tercer tipo de comunicaciones que se deben implementar son las internas intrínsecas a la plataforma utilizada, es decir, el bus PLB, que conecta los periféricos con el propio Microblaze. Gracias a la plataforma ofrecida pro Xilinx, el uso de este BUS no es algo de lo que el usuario deba preocuparse, quedando así integrado al procesador MicroBalze de forma monolítica.

Tabla 5.1: *Velocidad de interfaz ethernet en tarjetas de gama alta (Datos de fabricante)*

Tarjeta	*Tamaño Cache*	*Modo RAW rx*	*Modo RAW tx*	*Modo Socket rx*	*Modo Socket tx*
ML605	*8k*	*78.1Mbps*	*65.2Mbps*	*21.5Mbps*	*32Mbps*
ML605	*16k*	*101Mbps*	*80.7Mbps*	*24.4Mbps*	*38Mbps*
ML605	*32k*	*128Mbps*	*104Mbps*	*27.5Mbps*	*46Mbps*

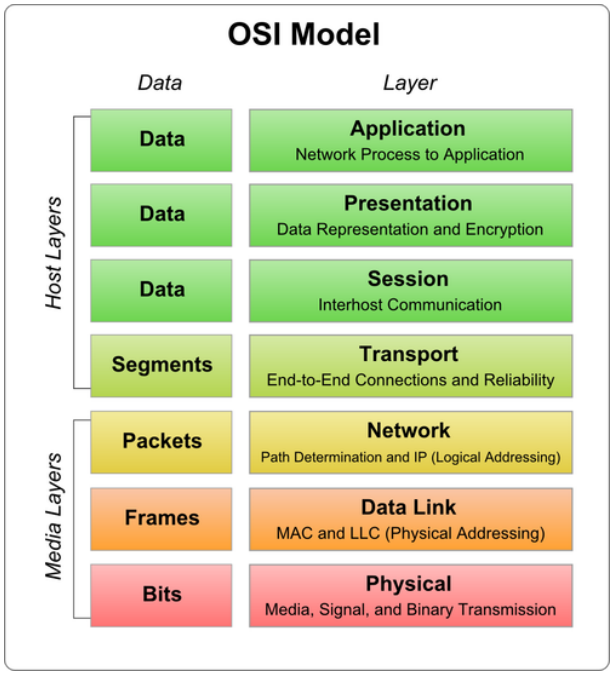


Figura 5.1: *Torre de protocolos OSI.*

5.2. Pruebas de comunicación Ethernet

La velocidad que puede alcanzar una comunicación ethernet en una plataforma Xilinx llega a superar los 120Mbps, tal y como se puede ver en la tabla 5.1 obtenida de la web de Xilinx [15]. Esta velocidad se obtiene con la plataforma ML605, que monta una FPGA Virtex6 (superior a la disponible para este proyecto).

Para establecer comunicaciones por medio de una red TCP/IP en una plataforma MicroBlaze es necesario disponer de un conjunto de funciones que se encarguen de implementarlo. A nivel de periféricos, Xilinx ofrece un módulo EMAC que provee del nivel físico y de enlace de la torre de una comunicación Ethernet. El resto de niveles de la torre de protocolos OSI (Figura 5.1) corre a cuenta del usuario, teniendo que implementar de forma hardware o software dichos protocolos. Afortunadamente, existen bibliotecas software de este tipo ya programadas totalmente disponibles para entornos MicroBlaze.

En este proyecto se ha utilizado la biblioteca lwIP [8], que consta de un conjunto de funciones para enviar y recibir datos con protocolo TCP/IP. Concretamente, lwIP (lightweight IP) es un stack TCP/IP de codigo abierto ampliamente usado para sistemas embebidos. Las características fundamentales que ofrece son:

- IP (Internet Protocol) incluido el reenvío de paquetes por múltiples interfaces.
- ICMP (Internet Control Message Protocol) para el mantenimiento y depuración de la red.
- IGMP (Internet Group Management Protocol) para gestión del tráfico multicast.
- UDP (User Datagram Protocol), comunicación no orientada a conexión.
- TCP (Transmission Control Protocol) con control de congestión, estimación de RTT y mecanismos FastRecovery y Fast Retransmit.
- API RAW para altos desempeños.
- API Socket.
- DNS (Domain names resolver).
- SNMP (Simple Network Management Protocol).
- DHCP (Dynamic Host Configuration Protocol).
- PPP (Point-to-Point Protocol).
- ARP (Address Resolution Protocol) para Ethernet.

La adaptación de Xilinx de este API permite el uso de 2 modos diferentes de funcionamiento, el modo RAW y el modo Socket.

El modo RAW soporta una velocidad de transferencia más alta, pero a cambio introduce una complejidad mucho mayor en el diseño del software a ejecutar en el MicroBlaze. En la tabla 5.2 se exponen las velocidades alcanzadas por este modo [16].

El modo Socket simplifica enormemente la utilización de este API, sin embargo, la adaptación de este modo por parte de Xilinx no está optimizada y provee tan solo alrededor de 1Mbps para cualquiera que sea la FPGA, la CPU, el modulo EMAC o la frecuencia del sistema [16]. Tabla 5.3.

Por tanto, a pesar de las grandes posibilidades que ofrece la tecnología, sólo podremos obtener velocidades máximas de 1Mbps.

Para Probar el rendimiento sobre la plataforma real se ha realizado una batería de pruebas y un test de velocidad de transferencia sostenida de datos. El resultado varía en

Tabla 5.2: *Velocidad de interfaz ethernet con lwIP en modo RAW (Datos de fabricante)*

FPGA	*CPU*	*EMAC*	*Frecuencia*	*Max TCP Throughput*
Virtex®-4	*PPC405*	*xps-ll-temac*	*100MHz*	*100Mbps*
Virtex®-5	*MicroBlaze*	*xps-ll-temac*	*125MHz*	*100Mbps*
Spartan3	*MicroBlaze*	*xps-ll-temac*	*66MHz*	*40Mbps*
Spartan3	*MicroBlaze*	*xps-ethernetlite*	*66MHz*	*20Mbps*

Tabla 5.3: *Velocidad de interfaz ethernet con lwIP en modo Socket (Datos de fabricante)*

FPGA	*CPU*	*EMAC*	*Frecuencia*	*Max TCP Throughput*
any	*any*	*any*	*any*	*1Mbps*

función del tamaño del buffer de datos transmitidos, es decir, del tamaño de los paquetes IP. En la tabla 5.4 se muestran las tasas obtenidas. Se aprecia cómo según aumenta el tamaño de los datos a enviar se obtienen velocidades mayores hasta llegar a un máximo en 870 registros de 4 bytes (3480 bytes). Analizar este resultado es complicado dado que se desconoce la implementación a bajo nivel de los sockets de Xilinx, pero la velocidad máxima es coherente con las especificaciones del fabricante.

El resultado es previsible, cuanto más se acerca el tamaño del buffer al tamaño máximo de la PDU (campo de datos) de la trama TCP/IP mayor rendimiento se obtiene. esto es debido a que la sobrecarga por cabecera (también conocido como overhead) es mayor si se utilizan tamaños pequeños de PDU. Cuando se supera el tamaño máximo de los paquetes a transmitir, 1500 bytes, la mejora en velocidad comienza a ser cada vez menor. La trama TCP/IP se muestra en la figura 5.2 haciéndose notar que el campo de datos de la trama Ethernet se corresponde con la trama completa del protocolo IP. A su vez, el campo de datos de la trama IP es la trama completa del protocolo TCP. La forma en la que funciona la torre de protocolos es similar a la de las muñecas rusas, se van incluyendo los protocolos de alto nivel en los de nivel inmediatamente inferior [12].

Tabla 5.4: *Test de velocidad via Ethernet*

Nº de Registros	*Velocidad [Bps]*	*Velocidad [Mbps]*
20	*2.680*	*0,02*
40	*5.330*	*0,04*
80	*16.030*	*0,12*
200	*32.150*	*0,24*
400	*89.170*	*0,71*
800	*122.032*	*0,97*
870	*138.375*	*1,107*

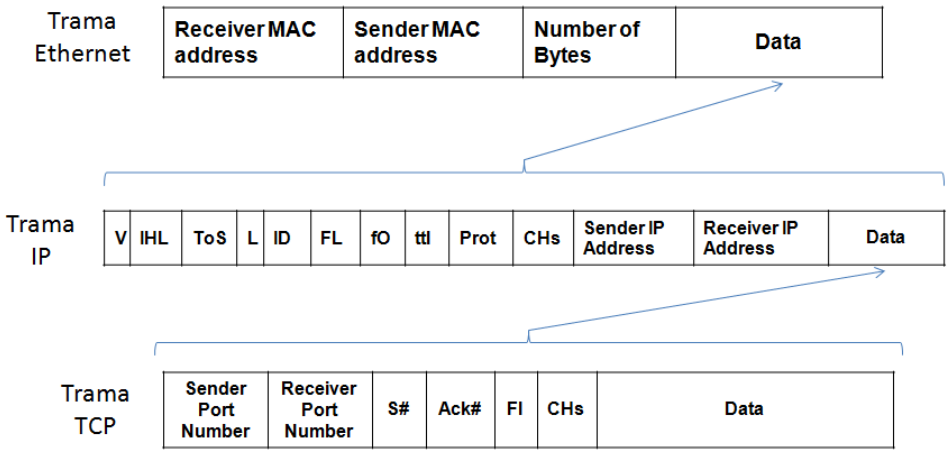


Figura 5.2: *Desglose de las tramas Ethernet, IP y TCP.*

5.3. Arquitectura Software

La arquitectura que describe el software del servidor de emulaciones se resume en el diagrama de flujo de la figura 5.3. Para cumplir con las especificaciones el software realiza unas configuraciones iniciales, entre ellas crear variables a usar posteriormente e inicializarlas, inicialización del periférico emulador activando su reset, así como envío de mensajes a través del puerto serie que permite al usuario comprobar el correcto arranque del sistema.

Además se inicializa Xilkernel junto con las configuraciones que designan el código a ejecutar por los hilos con los que contará el sistema. Al arrancar, tras las inicializaciones la ejecución queda a la espera de una conexión con un cliente. La naturaleza multihilo de la aplicación permite la conexión con múltiples clientes simultáneamente con plenas capacidades de control y gestión del servidor de emulaciones.

Tras el establecimiento de la conexión, el hilo queda a la espera de recibir comandos. Existen dos tipos de comandos, los de gestión y los de inicio y parada de la emulación. Al arrancar el sistema cliente e iniciar una conexión, se envían al servidor dos peticiones de inicialización de un hilo nuevo. De esta forma, antes de comenzar el proceso de emulación ya están listos los hilos de atención a comandos de gestión y de envío de resultados.

En el caso de comandos de gestión, como se esboza en la figura 5.4, al iniciar el hilo correspondiente lo primero que se hace es un reset del periférico emulador. Asi se evita que queden datos corruptos en los registros del circuito y se asegura que arranca en un estado conocido. Tras el reset, la ejecución queda a la espera de comandos por parte del programa cliente.

Con la llegada de un comando comienza el ciclo de ejecución típico de este hilo. Prime-

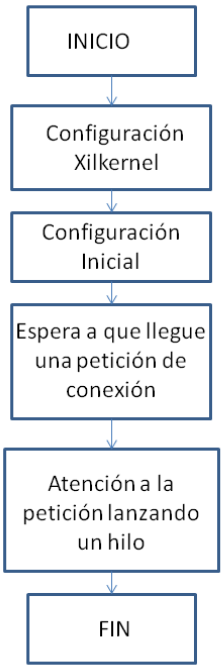


Figura 5.3: *Diagrama de flujo del arranque del servidor de emulaciones.*

ramente se obtiene el tipo de comando a partir de la información contenida en la propia petición (figura 6.4), para después ejecutarlo si el comando no es el de Desconexión. Una vez ejecutado el comando se construye la respuesta que se enviará al cliente, formada por la misma trama que la petición con la diferencia de incluir el resultado de la operación si procede. Por ejemplo, la respuesta del comando de escritura de un registro será una copia del comando enviado por el cliente, pero la respuesta de un comando de lectura incluirá el registro leído en el campo de datos de la trama.

Los comandos de arranque y parada de la emulación llevan asociado el mismo comportamiento inicial que los comandos de gestión figura 5.5. Fundamentalmente siguen el mismo esquema general de petición-respuesta, pero las diferencias recaen en la funcionalidad que realiza tras recibir una petición. Una vez recibido el comando de inicio de emulación, se espera hasta que el emulador esté arrancado, tras lo cual comienza la ejecución de un bucle infinito que sólo termina cuando la emulación ha finalizado. Durante el tiempo en el que el hilo de ejecución está en este bucle se envían a la máxima velocidad posible paquetes TCP/IP con los resultados de las emulaciones. Al finalizar una ejecución el hilo termina.

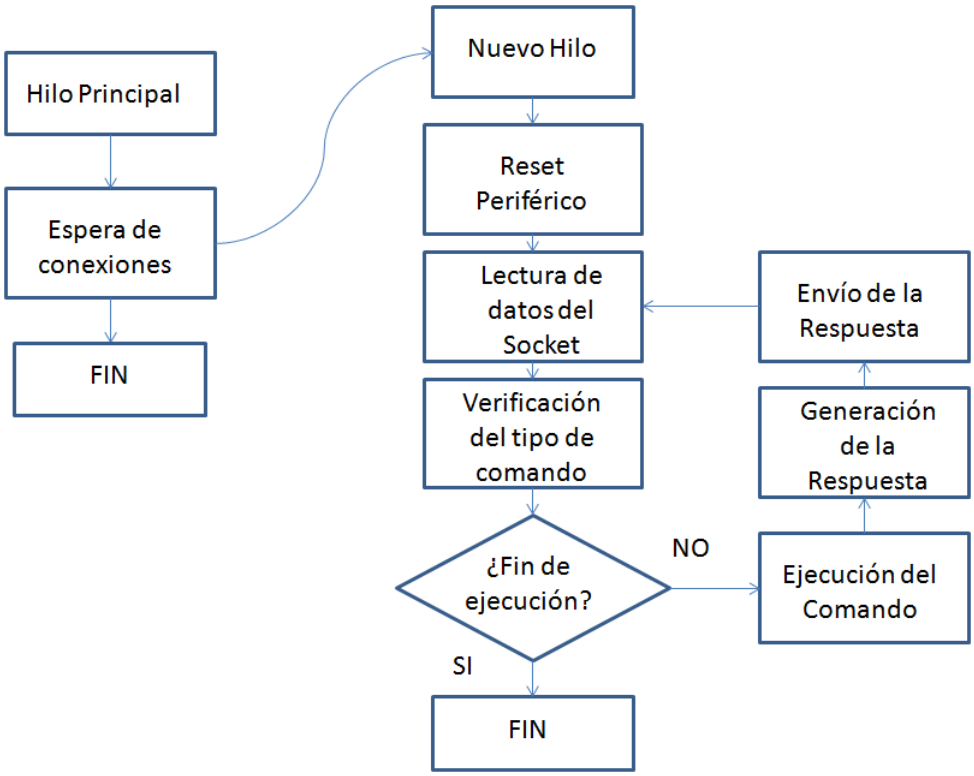


Figura 5.4: *Diagrama de flujo del código ejecutado por los comandos de gestión.*

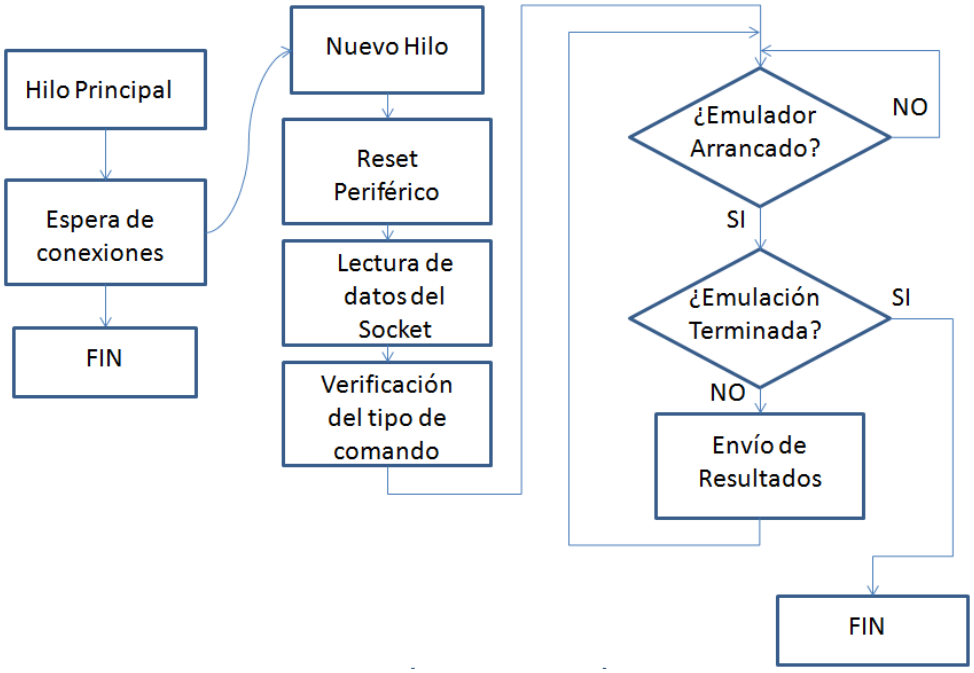


Figura 5.5: *Diagrama de flujo del código ejecutado por los comandos de arranque y parada del emulador.*

Tabla 5.5: *Comandos del protocolo de comunicaciones*

Nombre	*Código*	*Descripción*
Conectar	*0x01*	*Establece la conexión con el servidor de emulaciones*
Desconectar	*0x02*	*Desconecta el programa cliente y libera recursos*
Ping	*0x03*	*Efectúa una petición de ping*
Escribir Registro	*0x04*	*Escribe un registro del periférico*
Leer Registro	*0x05*	*Lee un registro del periférico*
Start	*0x06*	*Lanza la emulación*
Stop	*0x07*	*Interrumpe la emulación*
Socket Datos	*0x08*	*Lanza el hilo de envío de datos*
Reset	*0x09*	*Genera un reset en el periférico*

5.4. Gestión de parámetros

Como ya se ha apuntado anteriormente en este documento, la forma en la que se leen y escriben los datos en el emulador es a través de registros. Estos registros están mapeados en memoria, de forma que desde el Microblaze leer o escribir en un registro es tan sencillo como leer o escribir en una dirección de memoria.

De este mapeo se encarga el software EDK de Xilinx, que ubica todos los periféricos con sus respectivos métodos de comunicación en la memoria (registros, memoria compartida o FIFO´s). Este paso se realiza al pulsar el botón ”Generate Libraries” dentro del entorno de desarrollo en el menú “Software”. Como resultado se genera un archivo de cabecera con el nombre “xparameters.h”, que se incluye en la carpeta “include” del proyecto y que contiene todas las sentencias “#define” que facilitan el desarrollo de la aplicación. El origen de los datos plasmados en este archivo es la pestaña “addresses” del entorno visual del XPS. Un segmento del archivo xparameters.h puede verse a continuación.

```
[...]
/* Definitions for driver EMULADOR */
#define XPAR_EMULADOR_NUM_INSTANCES 1

/* Definitions for peripheral EMULADOR_0 */
#define XPAR_EMULADOR_0_DEVICE_ID 0
#define XPAR_EMULADOR_0_BASEADDR 0x00010000
#define XPAR_EMULADOR_0_HIGHADDR 0x0001FFFF
#define XPAR_EMULADOR_0_MEM0_BASEADDR 0x00020000
#define XPAR_EMULADOR_0_MEM0_HIGHADDR 0x00021FFF
[...]
```

Tabla 5.6: *Codificación de las posibles clasificaciones de fallos.*

Codificación	*Clasificación*
00	*Silencioso*
01	*Avería*
10	*Latente*
11	*Latente en RAM*

Se aprecia cómo se define el número de instancias del periférico emulador, así como el identificador de dispositivo. Posteriormente se define el inicio y fin de los segmentos de memoria correspondientes a los registros y a la memoria compartida respectivamente.

5.5. Envío de resultados en tiempo real

La forma en la que se transmiten los resultados de la emulación en tiempo real consta de dos partes y sigue el esquema mostrado en la figura 5.6. Cada paquete de datos enviado contiene 2 bloques , el primero de 256 palabras de 4 bytes y el segundo de 8 palabras también de 4 bytes.

El primer bloque contiene los resultados de las emulaciones codificados de forma que cada 2 bits se corresponden con la clasificación de un fallo inyectado (tabla 5.6). Cada palabra de 32 bits contiene 16 clasificaciones y el bloque completo 4096.

El segundo segmento de datos está formado por 8 palabras de 4 bytes cada una y contiene el estado del buffer de transmisión. Cada bit de este bloque representa el estado de una palabra del buffer de clasificaciones.

El emulador, según va generando clasificaciones, las inserta en un registro auxiliar que cuando se llena se guarda en un bloque de memoria compartida con el procesador MicroBlaze. Para conocer la posición de memoria en la que insertar resultados se almacenan estos 8 registros, que posteriormente en recepción servirán para señalar las direcciones de memoria que contienen nuevas clasificaciones (Figura 5.7).

De aquí se desprende que la tasa máxima admisible de generación de resultados es de 4096 clasificaciones por cada paquete de datos enviado. Si se supera esta velocidad de emulación se perderán clasificaciones y el diccionario de fallos no estará completo.

Se ha utilizado esta arquitectura para liberar la máxima carga posible al procesador MicroBlaze, ya que el cliente PC tendrá potencia de cómputo más que suficiente para ejecutar el algoritmo de recuperación de clasificaciones.

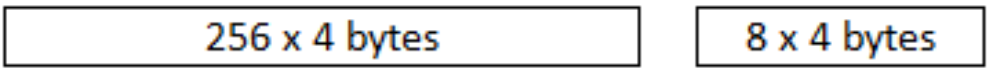


Figura 5.6: *Desglose de datos transmitidos por el servidor de emulaciones.*

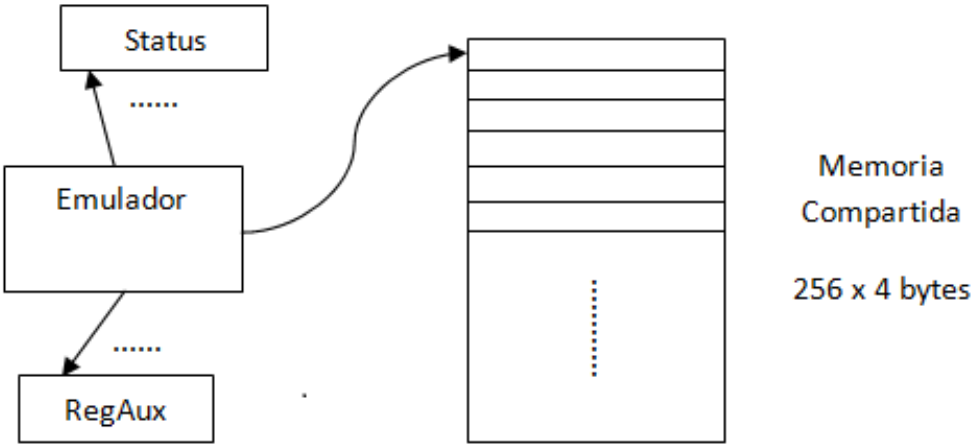


Figura 5.7: *Esquema de tratamiento de los resultados de emulación.*

5.6. Comunicación con programa cliente mediante Ethernet

Primeramente es necesario asignar una dirección IP estática al servidor de emulaciones, ya que una IP dinámica no permitiría establecer conexiones al no conocer dicho dirección. Por este motivo no se ha activado el protocolo DHCP (Dynamic Host Configuration Protocol), que permite al servidor obtener por de forma autónoma una dirección IP válida dentro de la red. Este protocolo hace que el servidor al arrancar, pregunte al router qué dirección IP puede tomar, en contestación el router responde una dirección dentro de un rango concreto que evita cualquier posibilidad de colisión con otro dispositivo conectado a dicho router.

Por tanto, la solución elegida finalmente ha sido la de establecer la dirección mediante software, para que así no colisione con otros dispositivos. Esto ha sido relativamente sencillo de realizar en el ambiente en el que se ha desarrollado el sistema, un laboratorio que dispone de una red Ethernet privada y dimensionada para este tipo de proyectos.

Capítulo 6

SOFTWARE DEL PROGRAMA CLIENTE

6.1. Especificaciones y requisitos

El requisito fundamental del software cliente es realizar una comunicación orientada a conexión con el servidor de emulaciones. Mediante esta comunicación se debe poder sincronizar datos, leer el estado del emulador y recibir los resultados en tiempo real. Todo ello bajo una interfaz gráfica de usuario que permita realizar estas acciones fácilmente.

La comunicación se realizará mediante protocolo TCP/IP para que el servidor quede operativo y accesible a través de internet a cualquier lugar del mundo. El protocolo TCP/IP (Transmission Control Protocol / Internet Protocol) se ubica por encima del protocolo IP y se enmarca como nivel de transporte (Figura 5.1). Las capacidades que aporta esta capa es la de crear una conexión, a ojos del usuario, por la que fluirán datos en ambos sentidos. TCP gestiona el control de flujo, control de congestión, ordenación de paquetes, etc. A la forma en la que se usa esta conexión de datos también se la puede llamar socket. La utilización del protocolo TCP/IP no surge como una necesidad, pero ofrece multitud de ventajas que eleva las posibilidades y usabilidad de la aplicación.

6.2. Arquitectura

Para llevar a cabo este proyecto fin de carrera se ha optado por el lenguaje Java para el desarrollo del software cliente. Sun Microsystems desarrolló el lenguaje Java a principios de los años 90 añadiendo funcionalidad, optimizando y adaptando Java a los nuevos tiempos que fueron apareciendo con ordenadores personales cada vez más potentes. Finalmente, en 2009 fue comprada por Oracle, que ha pasado a gestionar el desarrollo y mantenimiento

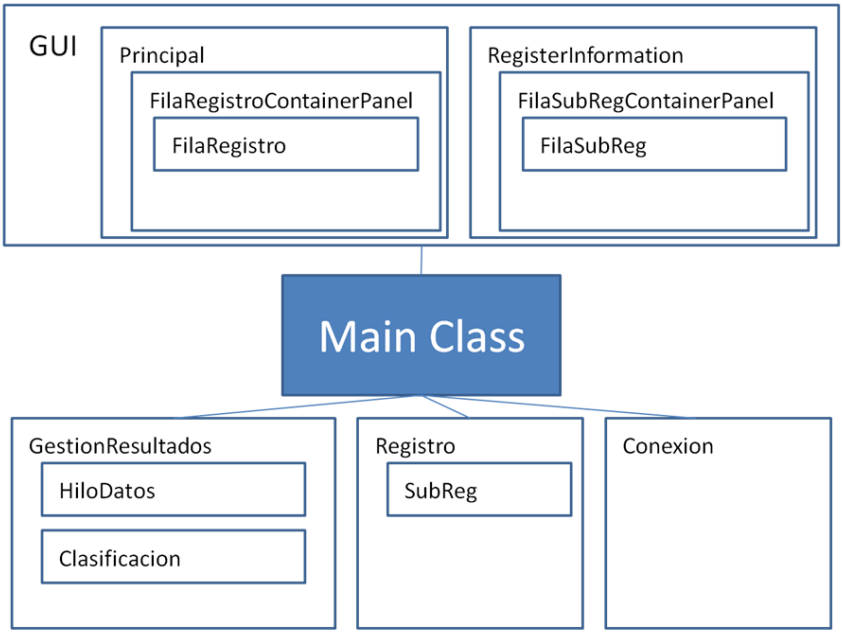


Figura 6.1: *Arquitectura Software del programa cliente.*

de la tecnología Java. Los desarrolladores de este lenguaje lo definen como “compilado en tiempo de ejecución”, lo que permite grandes optimizaciones en gran variedad de escenarios.

Java, además, corre sobre una máquina virtual que abstrae al programador de la gestión de memoria, compatibilidad entre equipos de muy distinto hardware, así como de ofrecer un entorno de depuración de errores muy avanzado. Entre otras particularidades, Java es orientado a objetos, lo que ofrece un conjunto de ventajas respecto a otros lenguajes de programación estructurada. Este tipo de programación fomenta la reutilización de código, permite crear sistemas más complejos más fácilmente, relaciona el sistema con el mundo real, agiliza el desarrollo y facilita el trabajo en equipo [4].

La arquitectura implementada en el programa cliente es la que se muestra en la figura 6.1. Consta de varios módulos independientes encargados de diferentes tareas, estos módulos en java son llamados paquetes. El módulo principal, también llamado Kernel, se encarga de encapsular todas las operaciones y funcionamientos, así como de comunicar los módulos entre sí. Sobre el módulo principal se ubica la interfaz gráfica de usuario (GUI), independiente del Kernel, en la que se realizan las funcionalidades de representación y manejo del programa. Bajo el Kernel se encuentran los otros 3 módulos, Gestión de resultados, Datos y Conexión. Estos últimos se encargan de dar soporte y facilitar las operaciones solicitadas al Kernel.

Tabla 6.1: *Atributos de la clase Principal*

Tipo	*Nombre*	*Descripción*
PrintWriter	*datosOut*	*Objeto usado para enviar comandos de control*
PrintWriter	*datosIn*	*Objeto usado para recibir respuestas de control*
BufferedWriter	*resIn*	*Objeto usado para recibir resultados*
String	*hostname*	*Nombre del servidor en la red*
int	*port*	*Puerto sobre el que realizar la conexión*
long	*ping*	*Retardo de la conexión*
boolean	*connected*	*Variable que indica el estado de la conexión*
Calendar	*dateConnected*	*Fecha en la que se estableció la conexión*
Vector	*registers*	*Lista de registros*

6.2.1. Módulo Principal

Este módulo recibe las peticiones de la interfaz grafica y además inicializa las conexiones y el resto de paquetes. Este paquete está formado únicamente por una clase Java, que tras inicializar el sistema y leer el archivo de configuración queda permanentemente a la escucha de peticiones de la interfaz.

Para implementar la recepción y envío de datos a través de una red se utilizan “sockets”. No es más que un concepto abstracto de programación que básicamente funciona como una tubería por la que enviar y recibir datos de forma transparente y ordenada. Todo ello entre dos programas informáticos (o entidades dentro de un mismo programa) ubicados en diferentes lugares conectados entre sí.

Esta clase principal es el núcleo de la aplicación y es la encargada de que el resto de módulos se entiendan entre sí correctamente y todo el flujo de información siga el camino correcto. En la tabla 6.1 se muestran los atributos de esta clase, entre ellos destacan los objetos “datosIn”, “datosOut” y “resIn”, que se utilizan para enviar y recibir datos de cada socket. Uno de ellos se encarga de los comandos de control y gestión y el otro de recibir los resultados de la emulación en tiempo real.

6.2.2. Módulo GUI

El módulo de interfaz es uno de los más complejos de la aplicación, consta de 6 clases Java además de hacer uso del módulo de Datos para facilitar el trabajo con registros. En los siguientes puntos se hace una reseña detallada de cada una de estas clases y sus

funciones.A modo de resumen, la clases son las siguientes:

- Clase FilaRegistro.
- Clase FilaRegistroContainerPanel.
- Clase Principal.
- Clase FilaSubReg.
- Clase FilaSubRegContainerPanel.
- Clase CustomCanvas.

En este módulo se encuentra la gestión de eventos producidos por la pulsación de botones en la interfaz. Cada ventana responde a un hilo de ejecución y cada pulsación en un botón o un control genera un evento que a su vez lanza un nuevo hilo que se encarga de ejecutar el código correspondiente.

Este módulo se beneficia particularmente de seguir el paradigma de la programación orientada a objetos, debido a que encapsula funcionalidad de cada elemento del panel a representar.

Clase FilaRegistro

Esta clase extiende la funcionalidad de un JPanel estándar de Java, que no es más que un panel para representar controles con algún añadido. El añadido es un conjunto de botones, campos de texto y etiquetas preconfigurados y colocados en su posición definitiva. Además incluye un gestor de eventos que atiende a las pulsaciones de botones y demás elementos. Existirán tantos objetos de esta clase como registros se hayan definido en el archivo de configuración. En la figura 6.2 se presenta la interfaz generada por esta clase, que posteriormente quedará incluida en la ventana principal de la aplicación.

Al generar cada objeto de esta clase, se pasan como parámetros de su constructor el número de registro, su nombre, el valor por defecto y el radix por defecto con el que se representará. Durante esta operación se representa visualmente en la pantalla junto con el resto de objetos FilaRegistro.

Figura 6.2: *Interfaz generada por la clase FilaRegistro.*

Clase FilaRegistroContainerPanel

Para poder colocar múltiples paneles FilaRegistro en un panel "padre" es necesario utilizar esta clase que, utilizando las funcionalidades de los GroupLayout, permite introducir un conjunto de paneles en otro panel generando barras de desplazamiento si el tamaño se excede del contenedor principal. De esta forma se genera un nuevo panel con un conjunto de subpaneles que representan toda la información requerida.

El resultado que busca esta clase es incluir en la ventana principal el conjunto de todos los registros de los que está formado el sistema leído de configuración. Más adelante se mostrará el contenido de este archivo de configuración y la forma en la que se gestiona.

Clase Principal

Esta clase se encarga de representar la ventana principal con toda su funcionalidad. Para cargar en tiempo de ejecución la configuración de pestañas y representarlo en consecuencia se vale de unos vectores de objetos de FilaRegistroContainerPanel y los incorpora a un panel de mayor jerarquía. Desde esta clase se realizan las llamadas al Kernel, que se encargará de proveer la funcionalidad de la aplicación.

Clase FilaSubReg

Esta clase es la equivalente a FilaRegistro para la ventana de configuración de registros. En este caso los controles que incorpora son un campo de texto, una lista desplegable y varios botones para modificar el valor de los subrregistros que contenga un registro dado. Habrá tantos objetos de esta clase instanciados como número de subrregistros contenga el registro seleccionado. una muestra de la interfaz generada por esta clase es la mostrada en la figura 6.3. Los datos que se representan en este panel son cargados al arrancar el sistema, por tanto no pueden ser modificados en tiempo de ejecución. En caso de querer modificar la distribución de subrregistros es necesario modificar el archivo de configuración y volver a arrancar la aplicación para que cargue los nuevos datos.

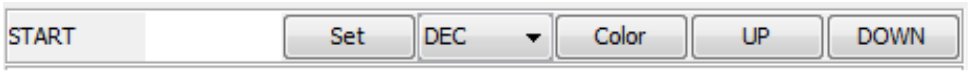


Figura 6.3: *Interfaz generada por la clase FilaSubReg.*

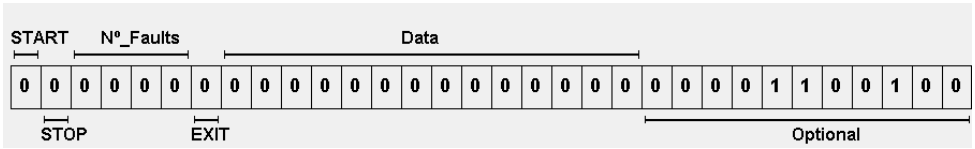


Figura 6.4: *Lienzo generado por la clase CustomCanvas.*

Clase FilaSubRegContainerPanel

Al igual que la clase FilaSubReg es la equivalente a FilaRegistro, esta clase es la equivalente a FilaRegistroContainerPanel. Por lo tanto aplica la misma explicación, ya que fundamentalmente tienen la misma función, aglutinar múltiples objetos FilaSubReg y que el usuario tenga la capacidad de modificar el contenido de los registros configurados.

CustomCanvas

La clase CustomCanvas extiende la funcionalidad de un Canvas de Java, que consiste en un lienzo en el que poder representar dibujos, imágenes y texto con la tecnología Java2D. Esta clase se ha creado para generar un gráfico informativo de la distribución de bits y subrregistros en un registro. Un ejemplo del lienzo generado es el mostrado en la figura 6.4. Esta clase soporta cualquier tamaño de subrregistro en cualquier posición a la hora de representarlo, así como ausencia de ellos en grupos de bits sin uso. Además permite seleccionar un color con el que resaltar cada uno de ellos de forma opcional.

6.2.3. Módulo Gestión de Resultados

El módulo de gestión de resultados tiene como objetivo principal obtener y dar formato a los datos recibidos, así como almacenarlos de la forma que proceda. Este módulo es de vital importancia, ya que sin él los resultados nunca llegarían al usuario.

El módulo de gestión de resultados contiene dos clases, HiloDatos y Clasificación, que

son las encargadas de construir el diccionario de fallos en tiempo real. En los siguientes puntos se desglosan ambas clases.

Clase Clasificacion

Esta clase encapsula el objeto que representa una clasificación junto a su funcionalidad. Permite generar una clasificación a partir de los bits que la codifican, así como listar y almacenar conjuntos de ellas. Esta clase hereda la funcionalidad de "Serializable", lo que quiere decir que en una futura ampliación de funcionalidad será sencillo almacenar un archivo binario en disco duro con las clasificaciones de forma que se puedan parar y relanzar emulaciones y estudios estadísticos de fallos.

Clase HiloDatos

En HiloDatos se encuentra el segundo hilo al que se hace referencia cuando se ha indicado que se construía una aplicación multihilo. Se encarga de recoger los resultados de la emulación recibidos por el socket y generar el diccionario de fallos a través de un algoritmo diseñado al efecto.

La forma en la que se reciben y gestionan los datos se ve de forma pormenorizada en el epígrafe 6.5.

6.2.4. Módulo de Datos

En este paquete se encuentran 2 clases encargadas de la gestión de los datos y la organización de ellos en registros y subrregistros. Las clases contenidas en este módulo no poseen entidad propia dentro del proyecto, es decir, no están asociados a ningún elemento visual de la ventana. Sin embargo son fundamentales para generar las estructuras de datos que facilitarán todos los cálculos que se realicen sobre registros.

Clase SubReg

La clase SubReg está encargada de encapsular el funcionamiento de un grupo de bits en el interior de un registro. Posee los siguientes atributos:

- Nombre, del tipo String.
- Posicion1, identifica en bit en el que comienza dentro del registro al que pertenece.

- Posicion2, identifica el bit en el que termina el subrregistro dentro del registro al que pertenece. Con estos dos atributos es posible conocer su tamaño sin necesidad de emplear otra variable para ello.
- FilaSubReg, objeto de esa clase que se encarga de representar en pantalla los controles.
- Color, indica el color con el que resaltar la posición del subrregistro en el lienzo en caso de activar la opción.

Clase Registro

Para almacenar y gestionar en memoria los registros a los que se tiene acceso de forma remota se ha creado la clase Registro. Los atributos que posee son:

- Numero, indica el número de registro mediante un entero.
- WR, identifica si el registro es de lectura solamente o de lectura y escritura.
- Radix, señala el tipo de representación a la hora de generar la interfaz gráfica, obviamente el registro no se ve modificado, sólo la forma en la que se representa.
- Pestaña a la que pertenece el registro obtenida del archivo de configuración leído al iniciar la aplicación.
- Valor, contiene el valor del registro.
- Subregs, almacena una lista de subrregistros, objetos de la clase SubReg.
- FilaRegistro, objeto con la interfaz necesaria para representar los controles del registro.

Esta clase además de poseer los atributos mencionados, implementa un conjunto de métodos, que son las acciones capaces de realizar un objeto de la clase Registro. En estas acciones se encuentran los métodos de obtener y modificar el contenido de los atributos antes mencionados y otro que actualiza los campos de la interfaz gráfica en función del contenido.

6.3. Configuración

Uno de los requisitos iniciales es que el sistema de emulación sirva para cualquier tipo de circuito. Por este motivo la interfaz de la aplicación cliente es configurable en función de un archivo de texto que debe estar alojado junto al ejecutable.

Como ya se ha expuesto en capítulos anteriores, la comunicación de control y estado entre el microprocesador MicroBlaze y el emulador se realiza a través de registros. Estos registros son accesibles en la interfaz gráfica de forma totalmente configurable. El archivo de configuración contiene información acerca de la colocación de los registros en pestañas, así como del nombre de los posibles subrregistros si los hubiere.

Un ejemplo de archivo de configuración es el mostrado a continuación:

```
0 STATUS R BIN Estado_Emulacion (0:Running 1:ScanError 2-3:NA 4:start 5:scanTest 6:clea
1 FFBEGIN W DEC Parametros_Emulacion (0-31:FFBEGIN)
2 FFEND W DEC Parametros_Emulacion (0-31:FFEND)
3 CYCLEBEGIN W DEC Parametros_Emulacion (0-31:CYCLEBEGIN)
4 CYCLEEND W DEC Parametros_Emulacion (0-31:CYCLEEND)
5 TBCYCLES W DEC Parametros_Emulacion (0-31:TBCYCLES)
6 INJFF R DEC Estado_Emulacion (sin subrregistros)
7 INJCYCLE R DEC Estado_Emulacion (sin subrregistros)
8 SILENTS R DEC Resultados (sin subrregistros)
9 FAILURES R DEC Resultados (0-31:registro)
10 LATENTS R DEC Resultados (0-31:registro)
11 RAMLATENTS R DEC Resultados (0-31:registro)
12 DETECTED R DEC Resultados (sin subrregistros)
13 RUNCOUNT0 R HEX Estado_Emulacion (sin subrregistros)
14 RUNCOUNT1 R HEX Estado_Emulacion (0-31:registro)
15 CONTROL W BIN Estado_Emulacion (0-31:CONTROL)
16 NO_USADO R HEX Estado_Emulacion (0-31:por determinar)
20 NO_USADO R HEX pruebas (0-31:por determinar)
21 NO_USADO R HEX pruebas (0-31:por determinar)
22 NO_USADO R HEX pruebas (0-31:por determinar)
23 NO_USADO R HEX pruebas (0-31:por determinar)
24 NO_USADO R HEX pruebas (0-31:por determinar)
```

Este archivo produce la interfaz gráfica de usuario de la figura 6.5. En ella se ve cómo la primera palabra de cada línea del archivo se corresponde con el nombre del registro junto al número del mismo. En caso de no necesitar algún registro, puede obviarse en esta lista, el sistema lo ignorará y no se representará en la interfaz gráfica.

El siguiente identificador indica si el registro es de lectura y escritura o de sólo lectura. Este aspecto es fundamental, ya que evita que el usuario pueda escribir datos en un mismo recurso simultáneamente. A bajo nivel no es posible escribir en el mismo registro desde dos fuentes distintas de datos gracias al diseño propuesto. El emulador nunca escribirá datos en los registros de estado, sólo lo hará en las posiciones de memoria compartida utilizando las interfaces ofrecidas por Xilinx. Estas interfaces proveen una temporización para la escritura de posiciones de memoria, por tanto, tampoco será posible escribir simultáneamente en una misma posición de memoria. A pesar de todo ello, la aplicación cliente permite especificar

registros de lectura y escritura para hacer más sencillo el uso y la comprensión de los datos representados.

Las siguientes 3 letras indican la base en la que se representarán los datos preferiblemente. Esta base puede cambiarse en cualquier momento, pero desde el archivo puede configurarse para que la aplicación muestre los datos al gusto del usuario desde el primer momento.

Para ordenar los datos en pestañas se utiliza la palabra posterior. Esta utilidad permite ordenar los diferentes registros indicando uno a uno a qué pestaña se desea que pertenezcan, indicando además el nombre de la pestaña.

Al final de cada línea de este archivo se encuentra un conjunto de identificadores entre paréntesis, con los que es posible crear subrregistros y asignarles un rango de bits dentro del registro. Los subrregistros irán precedidos de dos números separados por un guión para indicar el inicio y el fin del mismo. Si un subrregistro estuviera formado por un único bit sería suficiente indicar dicho bit antes de los dos puntos sin indicar el rango. Además, En el caso en el que un registro no contenga subrregistros no es necesario especificar nada entre paréntesis, de hecho ese contenido se ignorará.

Para obtener más información acerca de un registro concreto, la aplicación permite acceder a un panel específico. En él se ofrece al usuario el desglose de los datos que lo componen, así como la posibilidad de realizar cambios y lecturas de esto. Esta ventana se conoce como “Panel de Registro” y aporta toda la información que el usuario pueda necesitar en relación a un registro concreto. Se aprecia este panel en la figura 6.6, donde también se pueden cambiar los colores de la representación, cambiar su posicion en la lista (botones up y down), así como modificar y poner a cero todos o algunos subrregistros.

6.4. Protocolo de comunicaciones

En Java existe un completo API de clases que implementan multitud de funciones, entre ellas la comunicación TCP/IP mediante sockets. El programa cliente desarrollado para este proyecto Fin de Carrera hace uso de este API de sockets.

Las comunicaciones entre el cliente y servidor se realizan mediante comandos, que son construidos de la siguiente manera (figura 6.4):

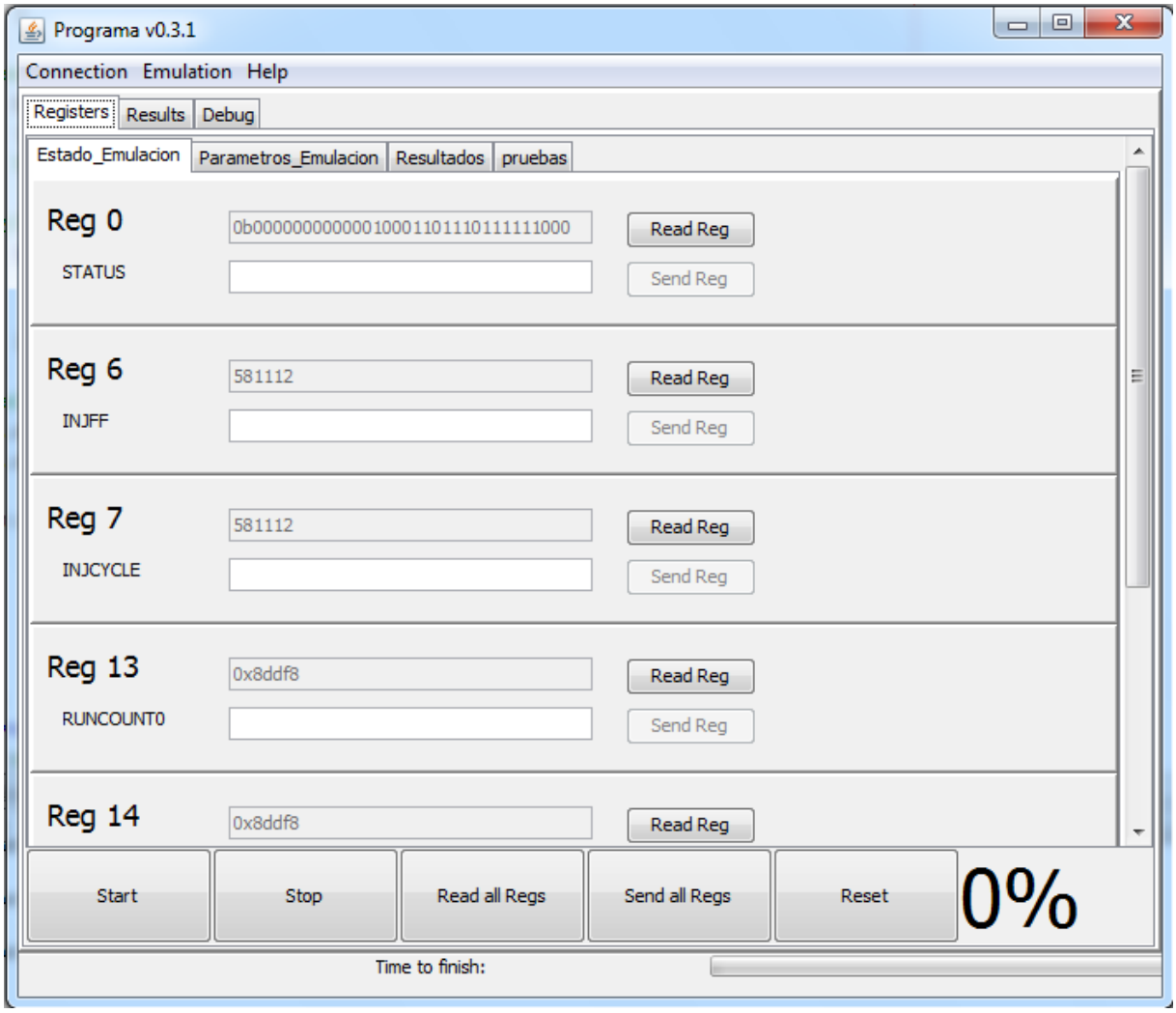


Figura 6.5: *Captura de pantalla de la interfaz gráfica de usuario (Panel principal).*

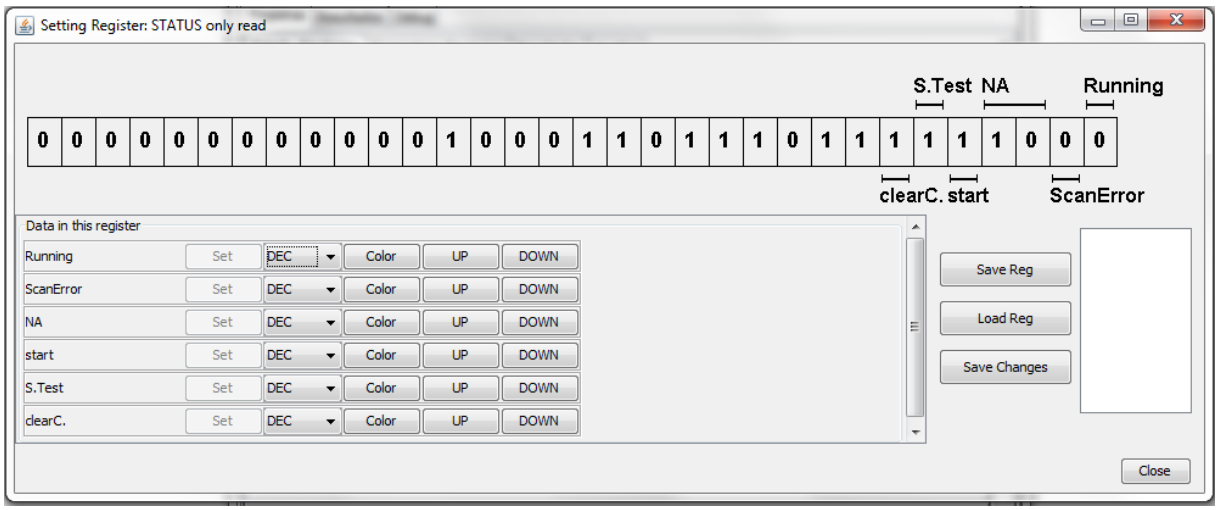


Figura 6.6: *Captura de pantalla de la interfaz gráfica de usuario (Panel de registros).*

Código Comando (1 byte)	Registro (1 byte)	Valor (4 bytes)

Figura 6.7: *Trama del protocolo implementado*

El comando consta de 6 bytes desglosados en 3 campos:

- Código de comando, se encarga de identificar el tipo de acción a realizar y debe ser uno de los definidos en la tabla 5.5. Se ha reservado espacio para 256 códigos de comando, dando la posibilidad de ampliación de funcionalidad en el futuro.

- Registro asociado al comando. Si el codigo de comando es relativo a lectura o escritura se tiene en cuenta este byte, en caso contrario se obvia. hace referencia al registro que ha de ser leído o escrito. En la solución propuesta se trabaja con 32 registros, este valor indica a cuál de ellos se hace referencia.

- Campo de Valor, consta de una longitud de 4 bytes y contiene el valor a escribir o leer del registro indicado en el campo “Registro” correspondiente.

En el protocolo implementado todo comando recibe una respuesta para informar del correcto funcionamiento, en función del código de comando el campo “valor” podrá contener la respuesta al mismo.

6.5. Tratamiento de resultados

Los datos de la emulación siguen el camino que muestra la figura 6.8. Primeramente las clasificaciones de los fallos se agrupan en palabras de 32 bits, es decir, 16 clasificaciones por palabra, y se almacenan temporalmente en un segmento de memoria compartida. Mientras tanto, el procesador Microblaze lee la memoria compartida y va enviando secuencialmente las palabras anteriormente almacenadas. Este envío no se realiza uno a uno, sino que también se agrupan en tramas. Finalmente estas tramas son enviadas por internet, terminando las responsabilidades del servidor de emulaciones.

Mas tarde, el PC cliente recibe las tramas de datos enviadas previamente por el servidor y mediante un algoritmo que se detalla en este mismo capítulo se obtienen las clasificaciones individuales y se procede al estudio estadístico de los resultados.

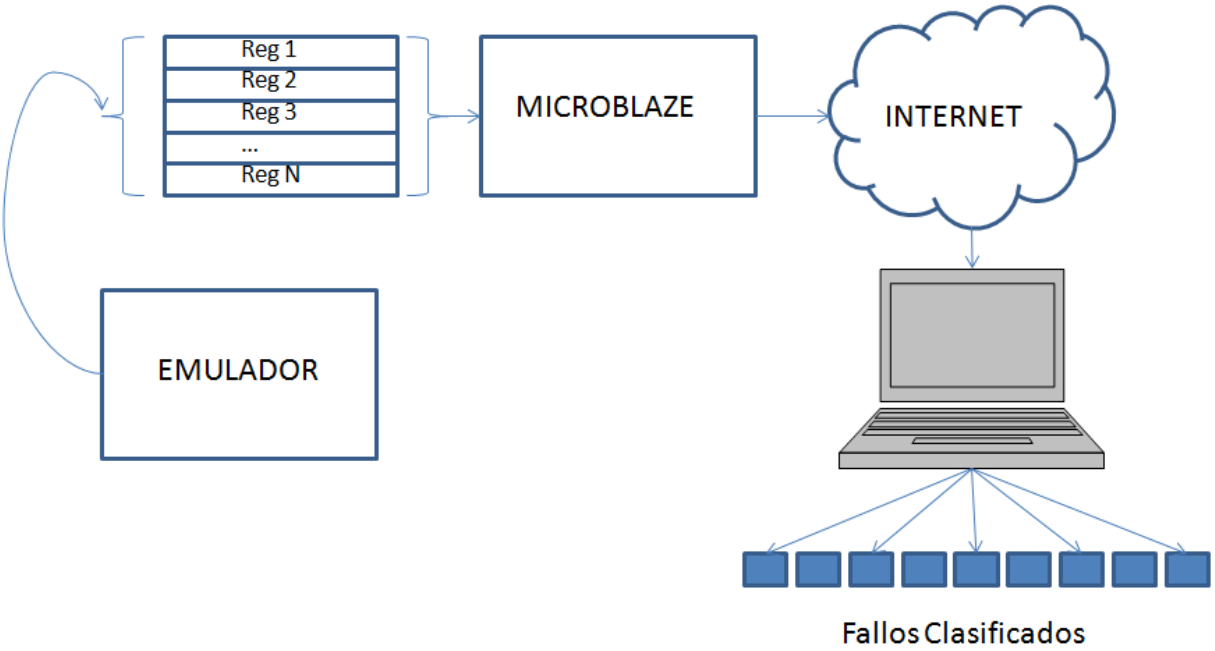


Figura 6.8: *Camino que siguen los datos desde que se generan en el emulador hasta que llegan al archivo de resultados.*

6.6. Interfaz gráfica

La interfaz gráfica de usuario mediante la cual se interactúa con el sistema consta principalmente de dos ventanas, la ventana principal y la ventana de gestión de registros.

6.6.1. Ventana Principal

Tras arrancar la aplicación se muestra la ventana principal (Figura 6.5), en ella se puede ver por orden de arriba a abajo una barra de menús, un panel de pestañas de 2 niveles, una barra de botones con acciones más frecuentes y finalmente una barra de progreso junto a un indicador de estado.

En la barra superior, se encuentran los menús de Conexión, Emulación y Ayuda. En esta primera versión de la aplicación sólo está disponible el menú conexión, desde el se establece la conexión con el servidor de emulaciones, se configura los parámetros de conexión y se desconecta la misma (Figura 6.9).

La opción “Propiedades” despliega otro panel en el que se configuran los parámetros de la

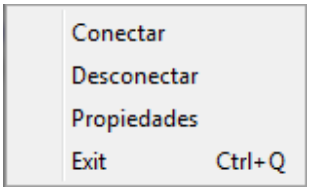


Figura 6.9: *Captura de pantalla de la interfaz gráfica de usuario (Menú "Connection").*

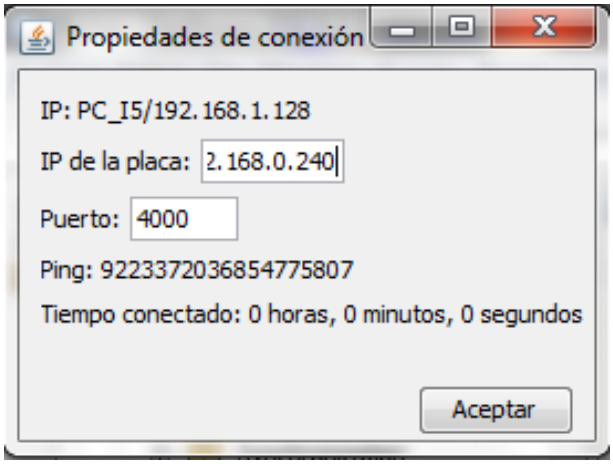


Figura 6.10: *Captura de pantalla de la interfaz gráfica de usuario (Panel "Propiedades").*

conexión con el servidor de emulaciones. Estos parámetros son la dirección IP y el puerto. Además, este panel, ofrece información útil acerca del estado de la conexión, como el ping (retardo en las comunicaciones en milisegundos), el nombre del equipo y la dirección IP propia. (Figura 6.10). En esta ventana es posible también modificar los parámetros fundamentales de la conexión, como son la dirección IP de la tarjeta y el puerto al que irá asociada dicha conexión. Una vez se presione el botón "start" los parámetros quedarán guardados en el sistema.

El panel de doble jerarquía de la ventana principal aporta la capacidad de organización que puede llegar a necesitar el usuario. En el primer nivel se encuentran las pestañas de Registros, Resultados y Debug. En la pestaña de registros se encuentran, como su nombre indica, los registros definidos en el archivo de configuración con su respectiva organización. En el apartado de resultados se ofrece una lista con el resultado de las clasificaciones que la aplicación cliente va recibiendo del servidor de emulaciones. Por último, la pestaña Debug aporta una herramienta para enviar peticiones manualmente al servidor y recibir las respuestas. Además permite cambiar el estado de los leds de la placa para verificar de una forma rápida el correcto funcionamiento de la conexión, del microprocesador MicroBlaze y del hardware de la tarjeta.

En la barra de botones inferior se encuentran las funciones que más frecuentemente se utilizan en la aplicación. EL botón “Start” Automatiza el proceso de arranque del emulador, concretamente establece a 1 el valor del bit correspondiente en el registro de control del emulador. En realidad, esta función tiene el mismo efecto que acceder al panel de subrregistros (Figura 6.6) y escribir el valor 1 en el subrregistro “Start”.

El botón “Stop” realiza la acción de parar la emulación abortando la transmisión de resultados.

El botón “Read all Regs” envía una petición de lectura de todos los registros y actualiza su valor en la interfaz gráfica. “Send all Regs”, por su parte, envía el valor de todos los registros modificados o no en el software cliente y los almacena en la FPGA. Estas acciones permiten tener un control total del contenido de los registros de comunicaciones entre el procesador MicroBlaze y el periférico emulador.

Por último, el botón “Reset” envía un comando específico que activa el bit específico del registro de control. De esta forma se puede resetear remotamente el emulador para prepararlo para una nueva campaña de emulación.

En la zona inferior del panel principal de la interfaz gráfica aparece una barra de progreso, una etiqueta que indica numéricamente el porcentaje de emulación y una barra de estado. En la barra de estado aparece información necesaria durante todo el proceso de emulación, el tiempo restante estimado y el tiempo transcurrido. Además muestra permanentemente el estado de conexión (Conectado o desconectado) y la dirección IP del servidor de emulaciones con el que se ha establecido la conexión si procede.

6.6.2. Ventana de Registros

Una parte muy importante de la aplicación, a la que se le ha dedicado especial interés, es la ventana de gestión de registros 6.6. Mediante esta ventana el usuario es capaz de establecer valores a determinados grupos de bits, además de leer y consultarlos para conocer resultados o el estado del emulador.

Así mismo, este panel permite almacenar registros para cargarlos posteriormente. Con esta avanzada funcionalidad, el usuario podrá crear diferentes perfiles de entradas al circuito emulador y cambiar entre ellas cargandolas con un simple click de ratón en vez de tener que volver a cargar los valores en los subrregistros. Esto garantiza una agilidad alta a la hora de trabajar con el emulador.

Capítulo 7

INTEGRACIÓN

7.1. Conexiones

Para el correcto funcionamiento de la tarjeta utilizada en el proyecto, es necesario realizar las conexiones que aparecen en la figura 7.1. Únicamente la comunicación serie es opcional, pudiendo eliminar el conector RS-232, ya que sólo tiene sentido en el ámbito de la depuración.

Para la comunicación Ethernet se utiliza el conector RJ45 de la imagen. Este cable deberá estar conectado a través de un router o switch a la red a la que desee tener acceso. Para que el sistema pueda funcionar desde cualquier lugar del mundo debe estar conectado a un router con salida a internet.

La carga del bitstream en la FPGA y del programa software que ejecutará el procesador embebido se usa el puerto JTAG. Este puerto además permite depuración en tiempo real de las aplicaciones. En este proyecto no se ha utilizado esta última función del puerto JTAG, se ha empleado el puerto serie por su simplicidad. Una simple sentencia “printf” en el código imprime la cadena de caracteres en el programa elegido para escuchar del puerto serie, en este caso HyperTerminal.

El último cable conectado a la tarjeta es el de alimentación, necesario para mantenerla funcionando. En caso de desconexión, es posible guardar tanto el bitstream de la FPGA como el programa software en una memoria no volátil y proceder a la grabación al encender la alimentación.

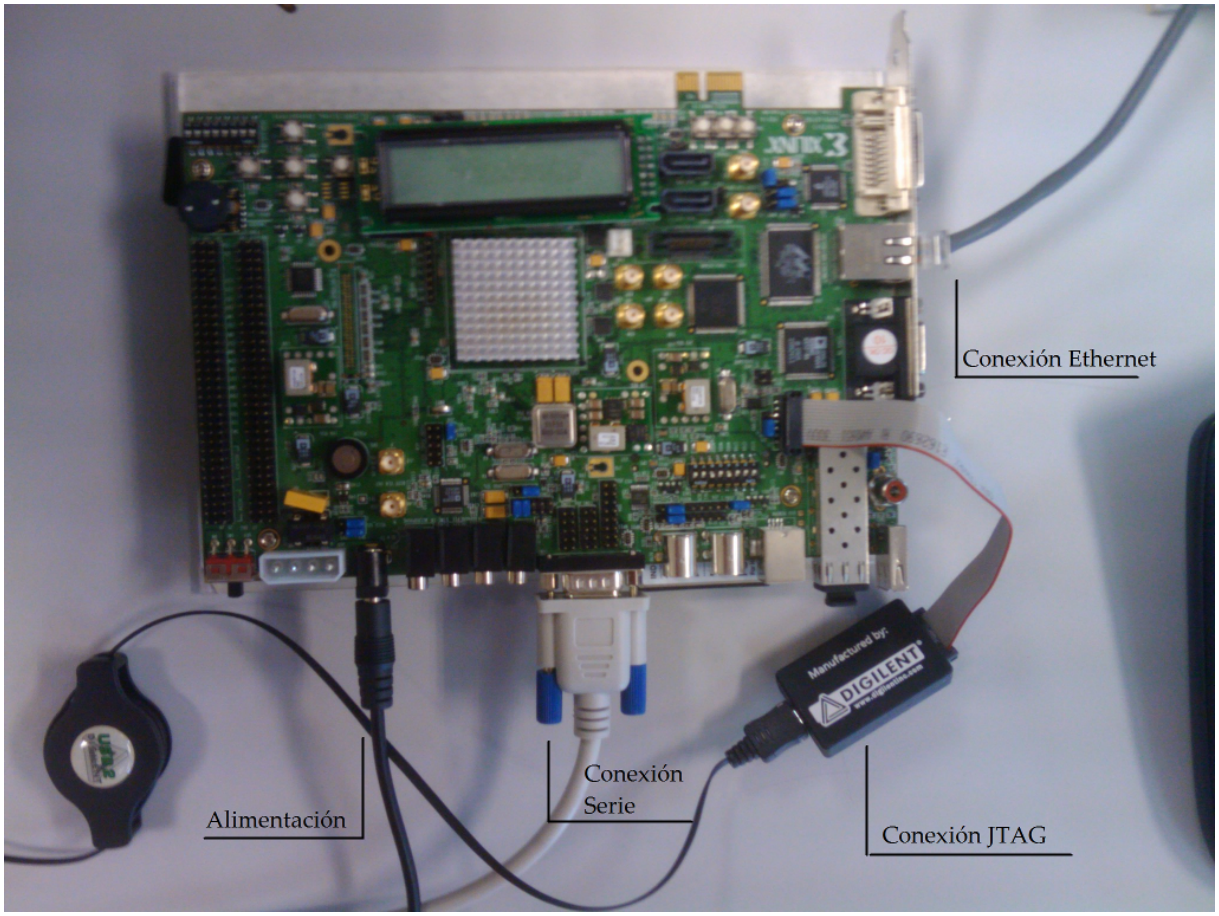


Figura 7.1: *Conexiones de la tarjeta XUPV5.*

7.2. Construcción del emulación

Tal y como se ha descrito durante este proyecto, el proceso para poder llevar a cabo una campaña de inyección está compuesto por los pasos generales que se detallan en los siguientes epígrafes.

7.2.1. Generar la netlist estructural del circuito

Primeramente se genera una netlist estructural del circuito a analizar. Esta netlist no es más que un aplanamiento de la jerarquía dejando la descripción hardware a nivel de puerta del circuito en un sólo archivo. Para generarla se hace uso del software Synplify PRO, software ya descrito en el capítulo 2. En este paso es necesario indicar al programa la entidad de mayor jerarquía.

7.2.2. Instrumentar el circuito

Para realizar este paso se hace uso de una herramienta software desarrollada en el grupo DMA. Esta herramienta realiza una sustitución de todos los elementos de memoria del circuito por sus homólogos modificados. De esta forma al final se obtiene una descripción hardware con un comportamiento idéntico al inicial pero con la capacidad de inyectar fallos indiscriminadamente en el circuito e identificar el estado del mismo. También permite cargar y restaurar el estado del circuito para así realizar campañas de emulación realmente rápidas.

7.2.3. Construir el sistema de emulación

En este paso se requiere la herramienta ISE de Xilinx para construir un sistema completo de emulación. En el se agrupa el circuito instrumentado y el emulador diseñado en el grupo DMA, que se encarga del control y gestión de resultados del la campaña de inyección.

7.2.4. Crear la memoria ROM con el banco de pruebas

Antes de poder lanzar una campaña de inyección es necesario generar una memoria con el banco de pruebas. Esta memoria tendrá tantas palabras como duración tenga el banco de pruebas, uno por cada ciclo de reloj. Esto es necesario para que el circuito ejecute un banco de pruebas conocido mientras se inyectan fallos, de esta manera se obtiene la robustez a

fallos. El tamaño de cada palabra almacenada en esta memoria lo determinan la cantidad de entradas del circuito.

Un ejemplo de esta memoria es el siguiente, donde se ve cómo se instancia un array de std-logic-vector con los valores por defecto.

```
type rom_type is array (0 to addrmax-1) of std_logic_vector (datawidth-1 downto 0);
constant ROM : rom_type := (
-- reset dato1 dato2
'1'&"000"&"00",
'1'&"000"&"00",
'0'&"001"&"01",
'0'&"001"&"01",
'0'&"010"&"10",

others =>
'0'&"000"&"00"&"00" );
```

7.2.5. Síntesis del proyecto

El siguiente paso es generar la síntesis del proyecto completo, llevando algo más de 45 minutos en un PC de gama media actual (intel Dual Core año 2008). Para llevarlo a cabo hay que navegar al menú "Hardware" y seleccionar "Generate Hardware".

Una vez terminado el proceso de síntesis, es necesario cargar el bitstream creado en la tarjeta, para lo cual es preciso seguir los pasos ya detallados en el apartado 7.2.

7.3. Carga del bitstream y el código software en la tarjeta

Una vez sintetizado el proyecto MicroBlaze se procede a grabar la FPGA con el. El software EDK realiza esta acción de forma sencilla seleccionando la opción "Download bitstream" del menú "Hardware". Durante el proceso se ejecutará el software IMPACT y se grabará la FPGA con el diseño desarrollado.

Posteriormente es necesario descargar en el sistema MicroBlaze el código compilado que deberá ejecutar. Sin esta operación el sistema hardware estará generado pero no tendría un software que lo controle. Para introducir el código compilado se siguen los siguientes pasos:

- Crear bibliotecas de software. Generará los archivos de dependencias necesarios para

ubicar los registros y bloques de RAM compartida. Así como las direcciones de los periféricos y diversas configuraciones.

- Lanzar el XMD, con esta operación se entra en modo debug y se crea un canal de comunicaciones para insertar código en el procesador embebido MicroBlaze. En este modo se pueden ver multitud de parámetros y estados del procesador. En el terminal que aparece se introducen estas dos directivas:

```
>> dow microblaze_0/code/lwipdemo.elf
>> con
```

Con este último paso el sistema arrancará y comenzará su ejecución normal.

7.4. Inicio de emulación a través de la interfaz

Una vez la aplicación cliente está lanzada y operativa, es necesario asegurarse de que el PC se encuentra conectado a internet o a la red local a la que esté conectada la tarjeta del servidor de emulación. El primer paso en navegar al menú “Connection” y seleccionar el ítem “Propiedades” para desplegar la ventana “propiedades de conexión” (figura 6.10). En ella se especificará la dirección IP o nombre DNS que identifica a la tarjeta y el puerto asociado a la conexión.

Posteriormente, también en el menú “Connection”, se presiona la opción “Conectar”. Esto iniciará el proceso de establecimineto de la conexión y dejará el sistema sincronizado y preparado para comenzar a realizar peticiones.

El siguiente paso es cargar los datos de emulación en los registros correspondientes, indicando los siguientes parámetros:

- Biestable de comienzo de emulación (Registro 1).
- Biestable de fin de emulación (Registro 2).
- Ciclo del banco de pruebas en el que dará comienzo la emulación (Registro 3).
- Ciclo del banco de pruebas en el que finalizará la emulación (Registro 4).
- Número de ciclos del banco de pruebas (Registro 5).

Por último queda iniciar la emulación presionando el botón “Start”. Durante el proceso será posible conocer el progreso de la emulación, así como el tiempo estimado de finalización. Al terminar el proceso, el resultado se almacenará en un archivo del disco duro junto al ejecutable “.jar” de la aplicación.

Capítulo 8

CONCLUSIONES Y TRABAJOS FUTUROS

8.1. Visión global del trabajo realizado

En este proyecto Fin de Carrera se han utilizado diferentes y muy variadas tecnologías tanto software como hardware. En el apartado de Hardware se ha empleado una plataforma Xilinx con una tarjeta de evaluación y su correspondiente entorno de desarrollo.

En el aspecto software se ha puesto en práctica los conocimientos en programación Java y C adquiridos durante la carrera, así como ciertas nociones de telemática y redes de ordenadores. Algunos de ellos han sido fundamentales en el desarrollo de este Proyecto Fin de Carrera.

Por tanto, este proyecto es la suma de las diferentes vertientes que tiene un desarrollo de esta índole, software, hardware y comunicaciones. Afrontando problemas y proponiendo soluciones específicas a cada aspecto. A continuación se desglosan los hitos importantes que destacan positiva y negativamente en este proyecto. Por último, queda una pequeña reseña de las posibles líneas de trabajo futuras para este proyecto tanto de ampliación como de mejoras.

8.2. Aspectos favorables

Este proyecto apuesta por la velocidad de las comunicaciones por soporté físico Ethernet y la disponibilidad del servicio a través de internet.

Con la solución sugerida en este proyecto se consigue aumentar la velocidad de transferencia de datos y por tanto se posibilita la transmisión del diccionario completo de emulación. Así como permitir el acceso al servidor de emulaciones desde cualquier parte del mundo con conexión a internet.

Otra ventaja de la solución propuesta frente al sistema de partida, es la capacidad de gestionar los datos y registros del emulador de forma intuitiva y rápida con una interfaz gráfica de usuario. Evitando así el uso del lento interfaz serie y sus comandos. La interfaz gráfica también ha sido uno de los puntos centrales del proyecto, cuyas especificaciones y funcionalidades estaban bien definidas desde el inicio.

Por último, un aspecto francamente positivo es la capacidad multiplataforma de la programación en lenguaje Java. Permitiendo ejecutar este mismo programa en máquinas Linux, Windows, Mac y Solaris sin necesidad de recompilar el código. Tan sólo es necesario tener instalada la máquina virtual de Java, que está disponible para numerosos sistemas operativos.

8.3. Aspectos desfavorables

El Mayor aspecto desfavorable de la solución es la baja o nula optimización del api de sockets ofrecido por Xilinx, que finalmente permite apenas 1Mbps. Puede ser de nuevo un cuello de botella en algunos casos que requieran altísimas velocidades te transferencia en pequeñas ráfagas.

El hecho de que el sistema no permita cambiar el circuito bajo pruebas sin resintetizar de nuevo todo el proyecto es un punto negativo que quedará como tareas para trabajos futuros. Puede darse el caso de que una pequeña modificación en el banco de pruebas requiera reconstruir todo el sistema por completo, necesitando fácilmente entre hora y media y dos horas de trabajo.

Por último, aunque se trata de un aspecto menos desfavorable, el incremento de necesidades de área crece. Es un problema inherente a la solución, sin embargo, tal y como se ha visto en el capítulo 4, el incremento es apreciable pero controlado, ya que el tamaño del sistema MicroBlaze siempre será el mismo.

8.4. Trabajos futuros

Posibilidad de cambiar el circuito al que realizar la campaña de inyección vía Ethernet. De esta forma el acceso físico a la tarjeta no es necesario salvo casos de mantenimiento o averías. Para esto sería necesario reprogramar la FPGA para que contenga el circuito

correspondiente.

Por último, otra mejora futura es utilizar el api RAW que provee lwIP para aumentar la velocidad de transferencia. Debido a que Xilinx no indica cuándo ofrecerá una biblioteca de funciones de sockets optimizada, ni tan siquiera si lo hará en algún momento, la línea de trabajo futura deberá ir en la dirección de integrar el api RAW de comunicación Ethernet al proyecto.

APÉNDICES

APÉNDICE A

PRESUPUESTO DEL PROYECTO

En este apéndice se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera. Tales costes, imputables a gastos de personal y de material, se pueden deducir de las Tablas A.1 y A.2.

En la Tabla A.1 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas. Así pues, se desprende que el tiempo total dedicado por el proyectando ha sido de 900 horas. Teniendo en cuenta una tarifas de 60 €/hora, el coste de personal se sitúa en 54.000 €.

El desglose del costo del software privado utilizado queda plasmado en la tabla A.3. No se incluye el sistema operativo utilizado por estar incluido en el precio del ordenador personal de gama media. Tampoco queda incluido el precio de la suite ofimática Office de Microsoft por su mínimo uso, ya que el software utilizado para redactar este documento ha sido TeXnicCenter como entorno de desarrollo y LaTex como compilador. Ambas herramientas gratuitas.

Tabla A.1: *Fases del Proyecto*

Fase 1	*Documentación*	40 *horas*
Fase 2	*Desarrollo del Hardware*	60 *horas*
Fase 3	*Desarrollo del Servidor*	160 *horas*
Fase 4	*Desarrollo del Cliente*	160 *horas*
Fase 5	*Pruebas de estabilidad y mejoras*	320 *horas*
Fase 6	*Documentación*	160 *horas*
Total		900 *horas*

Tabla A.2: *Costes de material*

Ordenador de gama media	800 €
Tarjeta XUPV5-LX110T	1.999 €
Software de distintas empresas	5.071 €
Local (durante 24 meses, con un coste de 120 €/mes)	2.880 €
Gastos varios	700 €

Tabla A.3: *Costes de software*

Xilinx ISE y EDK	4.295 €
Synplify	4.000 €
Total	8.295 €

Teniendo en cuenta que el número de horas laborables en un año asciende a 1.742 y que el número de horas imputadas a este proyecto han sido 900 se llega a que la dedicación ha ascendido a 0.61 años. El precio indicado en la tabla A.3 son referidos a licencias anuales, por tanto el costo total es de 0.61 x 8.295 = 5071 €

Los gastos varios se ven en la Tabla A.4. Entre ellos se tienen en cuenta la conexión a internet y el consumo eléctrico durante la realización de este Proyecto Fin de Carrera. Todo ello asciende a 1.270 €.

Finalmente, en la Tabla A.2 se recogen los costes de material desglosados en equipo informático, hardware y software necesario, local de trabajo, documentación y gastos varios no atribuibles (material fungible, llamadas telefónicas, desplazamientos...). Ascienden, pues, a un total de 8.579 €.

A partir de todos estos datos, el presupuesto total es 82.325 €, mostrado en la Tabla A.5. De este presupuesto se llega a la conclusión de que el mayor desembolso está en los gastos de personal y no en el material usado. Por tanto adecuar el personal al trabajo a realizar es clave para evitar un sobrecoste excesivo.

Tabla A.4: *Gastos varios*

Cable Ethernet x2	10 €
Switch/Router de red	50 €
Cable RS-232	10 €
Conexión a internet (24 meses x 50 €/mes)	1.200 €
Total	1.270 €

Tabla A.5: *Presupuesto*

Concepto	**Importe**
Costes personal	54.000 €
Costes material	8.579 €
Costes de software	5.071 €
Costes varios	1.270 €
Base imponible	68.920 €
I.V.A. (18 %)	12.405 €
TOTAL	82.325 €

Bibliografía

[1] AVNET. Pagina web del distribuidor de hardware avnet. website. http://www.avnet.com/.

[2] M. P. G. L. E. Celia López Ongil, Mario García Valderas. Autonomous fault emulation: A new fpga-based acceleration system for hardness evaluation. *IEEE Transactions on nuclear science*, 54:252–261, 2007.

[3] DMA. Documento explicativo del sistema de emulación autónomo. Document.

[4] F. G. y. E. P. Franciasco Durán. *Programación orientada a objetos con Java*. Paraninfo, Madrid,España, 2007.

[5] M. P. García. *Técnicas de inyección de fallos basadas en FPGAs para la evaluación de la tolerancia a fallos de tipo SEU en circuitos digitales*. PhD thesis, Universidad Carlos III de Madrid, Departamento de Tecnología Electrónica, 2007.

[6] M. Graphics. Pagina web del software modelsim. website. http://model.com/content/modelsim-pe-student-edition-hdl-simulation.

[7] N. W. Group. Rfc1323: Tcp extensions for high performance. RFC1323. http://tools.ietf.org/html/rfc1323.

[8] lwIP. The lwip tcp/ip stack. website. http://www.sics.se/~adam/lwip/.

[9] NetBeans. Pagina web del software de desarrollo java netbeans. website. http://netbeans.org/.

[10] Synopsys. Pagina web del sintetizador synopsys. website. http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/pages/synplifyfeaturecomparisonchart.aspx.

[11] Wikipedia. Artículo sobre netbeans en wikipedia. Article. http://es.wikipedia.org/wiki/NetBeans.

[12] Wikipedia. Protocolo tcp/ip. website. http://es.wikipedia.org/wiki/TCP/IP.

[13] Xilinx. Bus plb, peripheral local bus. website. http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.

[14] Xilinx. Ethernet mac overview. Document. http://www.xilinx.com/support/documentation/ip_documentation/tri_mode_eth_mac_ug138.pdf.

[15] Xilinx. Lightweight ip (lwip) application examples. website. http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf.

[16] Xilinx. Lightweight ip (lwip) application examples ii. website. http://xilinx.eetrend.com/files-eetrend-xilinx/forum/201103/1687-3087-xapp1026.pdf.

[17] Xilinx. Pagina web de la familia de fpgas virtex5. website. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[18] Xilinx. Pagina web del fabricante de la tarjeta de desarrollo xupv5-lx110t. website. http://www.xilinx.com/univ/xupv5-lx110t.htm.

[19] Xilinx. Pagina web del software ise. website. http://www.xilinx.com/products/design-tools/ise-design-suite/logic-edition.htm.

[20] Xilinx. Xilinx core generator overview. Documento. http://www.xilinx.com/ise/products/coregen_overview.pdf.

[21] Xilinx. Xilkernel, a kernel for the xilinx® embedded processors. Document. http://www.xilinx.com/ise/embedded/edk91i_docs/xilkernel_v3_00_a.pdf.