INYECCIÓN DE ERRORES SOBRE FPGAS TIPO VIRTEX-5

Víctor Alaminos Benéitez

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Máster en Ingeniería de Computadores
Convocatoria de Junio 2012 con calificación de SOBRESALIENTE

Fecha
26 / 05 / 2012

Directora:

Hortensia Mecha López

Colaborador:

Juan Antonio Clemente Barreira

Autorización de difusión

Autor
Víctor Alaminos Beneitez

Fecha
26 / 05 / 2012

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "INYECCIÓN DE ERRORES SOBRE FPGAS TIPO VIRTEX-5", realizado durante el curso académico 2011-2012 bajo la dirección de Hortensia Mecha López y con la colaboración externa de dirección de Juan Antonio Clemente en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

La computación reconfigurable es una tecnología prometedora capaz de proporcionar un compromiso interesante entre flexibilidad y prestaciones en el mismo dispositivo. Esta característica es especialmente interesante en campos tales como la aviación o las misiones espaciales, donde un amplio ámbito de aplicaciones deben ser ejecutadas eficientemente en un área reducida. Otro campo de interés es las aplicaciones nucleares, con requerimientos en tiempo real.

Sin embargo, en estos entornos el dispositivo que procesa la información normalmente está expuesto a altas dosis de radiación, ya sea proveniente directamente del espacio exterior, o de materiales radioactivos, la cual puede causar errores espontáneos en el funcionamiento del sistema. Esto hace imprescindible desarrollar técnicas que evalúan estos errores, para así ser capaces de reaccionar adecuadamente a ellos.

En este proyecto se han desarrollado una serie de programas y procesos para emular la alteración que una partícula cósmica produciría sobre la memoria de configuración de un diseño implementado sobre una FPGA, es decir, un bitflip. Estos programas se han integrado en una plataforma de depuración de errores, con lo que se consigue una plataforma de inyección de errores que permite evaluar el impacto de dicho bitflip espontáneo sobre el comportamiento de un circuito. Este sistema ha sido implementado en una FPGA Xilinx™ Virtex-5 aunque funcionaría igualmente sobre FPGAs tipo Virtex-4. La técnica de inyección de errores desarrollada en este trabajo se ha probado usando dos aplicaciones reales representativas.

Desde las primeras versiones de esta técnica, se han realizado una serie de importantes mejoras en el proceso de inyección de los bitflips y restauración del sistema, con lo que se ha conseguido una técnica que presenta varias ventajas respecto a otros sistemas existentes en la literatura. En primer lugar, la técnica no es intrusiva. En segundo lugar, el tiempo necesario para inyectar un solo bitflip y evaluar el error producido es mucho menor. Finalmente, teniendo en cuenta el estado del arte que se conoce sobre este tema, es la primera técnica de inyección de bitflips que ha sido desarrollada e implementada en una FPGA Virtex-5.

Palabras clave

Hardware reconfigurable, FPGA, Virtex-5, reconfiguración parcial, reconfiguración dinámica, inyección de errores, bitflip, ICAP, aplicaciones aeroespaciales.

Abstract

Reconfigurable computing is a promising technology able to provide an interesting trade-off between flexibility and performance in the same single device. This feature is especially interesting in fields such as aviation or space missions, where a wide range of applications must be efficiently executed in a reduced area. Another field of interest is nuclear applications, with real-time requirements.

However, in these environments the device processing the data is usually exposed to a high amount of radiations, either coming directly from the outer space, or radioactive materials, which may cause spontaneous faults in the system. This makes fundamental to develop techniques that specifically evaluate these faults, in order to be able to tackle them.

In this proyect a set of programs and processes have been developed to emulate the alteration that a cosmic particle would produce over the configuration memory of a design implemented on a FPGA, that is a bitflip. These programs have been integrated in a error-evaluation platform, thereby producing a fault-injection platform that makes possible to evaluate the impact of this spontaneous bitflip in the behaviour of a circuit. This system has been implemented in a Xilinx™ Virtex-5 FPGA although it would work equally on Virtex-4 FPGAs. The developed fault-injection technique in this proyect has been tested using two representative real-world applications.

Since the early versions of this technique, a group of improvements have been done in the bitflip injection process and the system restoration, after all, the obtained technique presents a number of advantages with respect to other existing state-of-the-art systems. Firstly, the technique is non-intrusive. Secondly, it greatly improves the time that is needed to carry out a single bitflip, as well as to evaluate the produced error. Finally, taking into account the state-of-the-art that is known about this subject, this is the first bitflip-injection technique that has been developed and implemented in a Virtex-5 FPGA.

Keywords

Reconfigurable hardware, FPGA, Virtex-5, partial reconfiguration, dynamic reconfiguration, fault-injection, bitflip, ICAP, space applications.

Índice general

Agradecimientos

Quiero agradecer con este trabajo, a mi familia, a mi novia, mis amigos y compañeros que me han dado fuerzas para llevar a cabo este proyecto. También quiero agradacer a mi directora Hortensia Mecha, como al colaborador de este proyecto Juan Antonio Clemente, que gracias a su apoyo y motivación, me han ayudado en todo lo posible, sobre todo cuando estaba atascado y así conseguía salir adelante. Así como quiero agradacer a todos mis profesores en toda mi formación académica, ya que gracias a ellos (en mayor o menor medida), he conseguido llegar donde estoy y poder realizar este trabajo.

Capítulo 1

Introducción

1.1. Motivación

Durante las 2 últimas décadas se ha producido un incremento de uso de las *Field Programable Gate Arrays* (FPGAs) basadas en memoria SRAM (SRAM-FPGAs) para implementar aplicaciones de lógica digital, debido principalmente a su alta densidad de recursos configurables y la capacidad ilimitada de reconfiguración, ya sea total o parcial. En particular, en el sector aeronáutico y aeroespacial, su uso está muy extendido ya que los dispositivos situados en aviones o satélites normalmente deben procesar grandes cantidades de información antes de enviarlas a la Tierra, debido al reducido ancho de banda existente en el canal de comunicación que conecta ambos. En aplicaciones nucleares, su uso es realmente muy interesante debido a sus restricciones de tiempo real muy severas.

Este incremento ha venido acompañado de un aumento de la preocupación sobre los efectos que pueda causar la radiación sobre las mismas. En particular los efectos de los llamados Single Event Upsets (SEUs), producidos por el impacto de una partícula cósmica al colisionar con el dispositivo, ya que los dispositivos están expuestos a altas dosis de radiación [1], [2]. El resultado de estos efectos son la alteración no solamente en los flip-flops y *Random Acess Memoriy Blocks* (BRAMs) del diseño del usuario, sino también del contenido de la *Static Random Access Memory* (SRAM) usada para la memoria de configuración de la FPGA. Un SEU puede inducir un bitflip en la memoria de configuración SRAM, lo cual

puede alterar la funcionalidad del diseño [1], pudiendo provocar fallos en su ejecución.

Experimentos de radiación indican que el porcentaje de SEUs en las FPGAs incrementa en un factor del 4.74 % cuando el diseño decrementa de los 600nm a los 350nm con la correspondiente reducción del voltaje Vcc de 5V a 3.3V [3]. Por ello, Xilinx esta diseñando FPGAs con memorias de configuración más robustas, pero aun así muy vulnerables. Por otra parte, distintos grupos de investigación han llevado a cabo estudios sobre técnicas de protección de circuitos como la *Triple Modular Redundancy* (TMR) [4], [5]. Esta técnica consiste en replicar el circuito 3 veces y procesar las 3 salidas resultantes votándolas para generar una sola salida (el proceso de votación consiste en seleccionar la salida que más veces se repita). Si alguna de las instancias del sistema falla, se puede corregir el resultado con las otras dos y de esta manera enmascarar el error producido. El problema de esta técnica es que consume muchos recursos hardware, ya que implica multiplicar el coste del hardware por 3, más el coste del circuito que implementa el votador. Para reducir costes hardware, es interesante realizar un estudio y sólo aplicar la redundancia en la parte más sensible del circuito.

Sea cual sea la técnica de protección empleada, surge la necesidad de utilizar algún método para evaluar la calidad de las soluciones y por tanto la vulnerabilidad de un determinado diseño. Estos métodos son los llamados de "inyección de errores" que permiten simular o emular la caída de una partícula cósmica sobre el circuito, es decir, la alteración de la memoria de configuración, y evaluar los errores producidos en la salida del mismo.

1.2. Trabajo Relacionado

En general, las técnicas de inyección de errores se pueden dividir en 3 grupos: basadas en simulación, basadas en software y basadas en hardware [6].

Las técnicas basadas en simulación consisten en insertar un fallo en el modelo de un circuito y entonces simularlo. Se pueden utilizar simuladores específicos, o simular un modelo HDL usando un simulador comercial. Estas técnicas son capaces de evaluar los efectos de los fallos en el proceso de diseño, pero son muy costosas en tiempo. Además, su efectividad depende de la precisión del modelo.

Insertar un fallo en una rutina de ejecución en un sistema es en lo que consisten las técnicas basadas en software. Esta rutina de fallo es usualmente lanzada usando un mecanismo de interrupción. La inyección puede ser producida muy rápidamente debido a que las rutinas se ejecutan a la frecuencia de operación del sistema. Sin embargo, estos métodos son muy intrusivos y poco fiables ya que sólo es posible inyectar errores en las partes accesibles por el software del sistema.

Por último, las técnicas más ventajosas de las 3 son las basadas en hardware, ya que funcionan a frecuencias más altas y permiten trabajar sobre la implementación final del circuito, inyectando los errores directamente en el sistema. Las plataformas hardware existentes en la literatura más representativas de este tipo de técnica son FLIPPER [7] y FTUNSHADES [8], las cuales utilizan dos FPGAs Xilinx Virtex-II. Las plataformas están compuestas por una FPGA que sirve como controlador del experimento (C-FPGA), conectada a un PC que configura la plataforma e interpreta el análisis de post-inyección. La otra FPGA sirve para implementar el circuito a testear (S-FPGA o Device Under Test "DUT").

FLIPPER utiliza la primera FPGA (C-FPGA) para inyectar los bitflips en la memoria de configuración de la segunda FPGA (DUT) mediante un sistema de registros muy complejo. Esta plataforma ha sido validada de forma experimental en [9], comparando las probabilidades de fallo obtenidas por la plataforma con las obtenidas inyectando partículas y produciendo SEUs con un acelerador de protones. Los problemas que presenta FLIPPER

son su alto coste, al necesitar 2 FPGAs y un diseño ad-hoc para las dos placas. En segundo lugar resulta muy intrusiva, puesto que evalúa el funcionamiento del sistema inyectando bitflips en todo el área de la FPGA, como si el diseño ocupase completamente la misma. Por último, debido al diseño ad-hoc para las placas, no es fácilmente portable a otras tecnologías.

En la plataforma FTUNSHADES se programan dos copias del circuito en la S-FPGA: uno será el circuito *golden* (correcto) y el otro, el circuito radiado. Se realizan modificaciones inyectando errores en la memoria de configuración correspondiente al circuito "radiado" de la FPGA, y luego se comparan los resultados obtenidos por ambos circuitos. Esta plataforma alcanza un rendimiento muy alto, sin embargo tiene dos inconvenientes. El primero es que es una plataforma muy intrusiva, ya que implementa dos copias del diseño a testear en el dispositivo bajo test, por lo que el diseño difiere del diseño final que va a ser utilizado. Además, los SEUs inyectados en la S-FPGA pueden también modificar el diseño *golden*, comprometiendo así el test completo. Finalmente, el diseño ad-hoc de las placas hace la plataforma muy costosa.

En cuanto a tecnologías más modernas, se ha desarrollado una plataforma basada en hardware para una Xilinx Virtex-4 [10], la cual también usa dos FPGAs y un PC, de la misma forma que las anteriores plataformas (una FPGA funciona como controladora y la otra, como circuito a testear). La FPGA controladora inyecta los fallos utilizando reconfiguración parcial dinámica a través del puerto ICAP de la DUT. Esta plataforma es costosa, ya que utiliza 2 FPGAs y muy lenta, debido a que en algunos casos, la inyección produce un fallo en el propio interfaz ICAP del sistema, con lo que es necesario realizar una reconfiguración de toda la FPGA a través del interfaz externo de configuración.

Por otra parte, entre las técnicas existentes de protección de circuitos, podemos encontrar en la literatura varios modelos para reducir el coste de la técnica TMR. Se han investigado distintas alternativas, donde tras un profundo conocimiento del circuito, se replican sólo partes del mismo. Las basadas en experimentación, se caracterizan por reproducir la radiación del medio al cual va destinado el dispositvo. La desventaja principal de estas técnicas

es que tienen un setup y un coste experimental muy elevado, y ademas la incertidumbre de la localización del error inyectado. Otro tipo de técnicas, muy rápidas en ejecución, son las técnicas analíticas, las cuales intentan estimar los errores que pudiesen producir un fallo en el sistema. Estas técnicas utilizan aproximaciones probabilísticas o mediante métodos de gráfos, los cuales son muy rápidos en ejecución. Sin embargo, estas técnicas requieren un modelo muy preciso del circuito para evaluar correctamente la sensibilidad de los errores, y sólo proporcionan estimaciones probabilísticas.

Vistas las debilidades de las plataformas existentes y la necesidad de evaluar la calidad de las distintas técnicas de protección, en este proyecto se propone realizar, a partir de una plataforma de depuración de circuitos desarrollada en esta facultad, una nueva plataforma capaz de realizar una inyección total de errores en un circuito determinado, y que funciona para las FPGAs actuales tipo Virtex-5.

1.3. Objetivo y contribuciones

Como se ha dicho anteriormente, el objetivo principal de este proyecto es obtener una plataforma que simule los impactos de las partículas cósmicas en un diseño implementado sobre una FPGA del tipo Virtex-5, insertando bitflips en su memoria de configuración, para luego poder observar el efecto de estos bitflips en el funcionamiento del circuito implementado. Además, para reducir su coste, se pretende diseñar una plataforma de inyección de errores basada en hardware sobre una sola FPGA y así que el coste de la implementación en términos de área sea considerablemente menor que otros sistemas [7], [8], [9], [10]. También como objetivos, se pretende que sea una plataforma no intrusiva, es decir, no modifique la implementación final; que mejore el tiempo (reduciéndolo) necesario para inyectar un solo bitflip y en evaluar el error producido [6], [10]; y que sea una plataforma válida para FPGAs del tipo Virtex-5, las cuales son más evolucionadas que las Virtex-II utilizadas en casi todas las plataformas de inyección de errores basadas en hardware. La implementación de esta plataforma se realiza sobre la herramienta de depuración NESSY, que consta de un software de interfaz de usuario que corre en un PC y de un circuito hardware implementado en una FPGA del tipo Virtex-5 que se mantendrá comunicada con el PC. A la herramienta NESSY se le ha añadido la funcionalidad de reconfiguración de la FPGA, y para validar su funcionamiento, se ha realizado una inyección de bitflips en cada uno de los bits de la memoria de configuración correspondientes al diseño evaluado. El resultado es una plataforma de inyección de errores basada en hardware con una serie de mejoras respecto a otras plataformas del mismo tipo presentadas en la sección anterior 1.2. Además, esta plataforma es fácilmente portable a cualquier versión de las FPGAs tipo Virtex.

Capítulo 2

Entorno de desarrollo Hardware y Software

En este capítulo se va a explicar el entorno de desarrollo de la plataforma de inyección de errores. Por una parte, se utilizará una FPGA para implementar el hardware del circuito a testear y un sistema de control del mismo y comunicación con el interfaz de usuario que se ejecutará sobre un PC. Por otra parte, distintas herramientas software para generar la aplicación de usuario, y el software que se ejecutará sobre el circuito de control, basado en Microblaze.

2.1. Arquitectura Hardware

Una parte de la plataforma de inyección de errores va a implementarse sobre una FPGA XC5VLX110T de la familia Virtex-5. Dicha FPGA está empotrada en una placa de prototipado XUPV505-LX110T (Figura 2.1). A continuación se van a comentar algunas características de la placa de prototipado, de las FPGAs tipo Virtex-5, y en particular de la Virtex-5 LX110T usada en este proyecto.

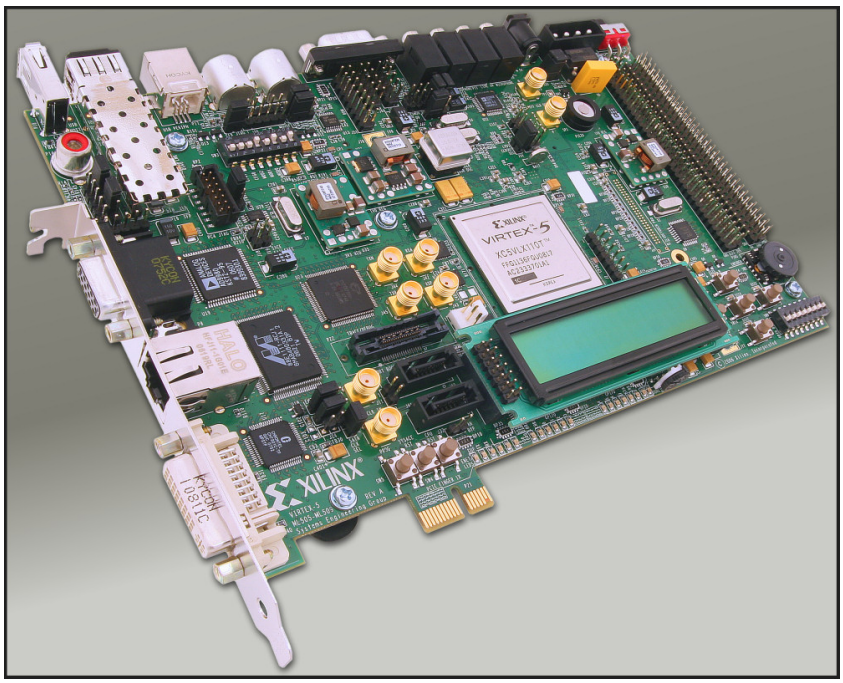

Figura 2.1: *Placa de prototipado XUPV505-LX110T*

2.1.1. Placa de desarrollo XUPV505-LX110T

Las placas de desarrollo XUP aportan algunos componentes periféricos a la FPGA que va empotrada en la misma. Concretamente, la placa XUPV505-LX110T (Figura 2.1), dispone de los siguientes componentes periféricos: una memoria DDR2 RAM que puede ser de 64 MB hasta 2 GB, un puerto Ethernet 10/100, códecs de audio y vídeo, pines de entrada y salida para una tarjeta Compact Flash, puertos PS/2, switches, leds, puertos USB, puertos XSGA de vídeo y conectores SATA, entre otros. Además, aunque disponemos de gran variedad

de componentes periféricos, como se ha mencionado al principio de este capítulo, para la realización de este proyecto, sólo se han utilizado los necesarios para programar la FPGA (puerto JTAG) y el puerto serie de la UART de la placa XUPV5 usando el protocolo RS232.

2.1.2. FPGA tipo Virtex-5

Como se ha mencionado en los apartados 1.1 y 1.3, este proyecto se enfoca en las FPGAs y sus problemas con la tolerancia a la radiación. Las FPGAs son circuitos electrónicos prefabricados reconfigurables que contienen bloques de lógica cuya funcionalidad e interconexionado puede ser programado. Esto permite programar la FPGA con nuevos circuitos y reemplazarlos por otros nuevos cuando se necesite. Esto es posible ya que la configuración del circuito que se quiere programar se almacena en la memoria de configuración de la FPGA, que suele ser volátil, y donde se puede borrar y almacenar una nueva configuración.

Las FPGAs aparecen como una evolución de las conocidas *Complex Programable Logic Device* (CPLDs). Las ventajas que tienen las FPGAs respecto a los CPLDs son que las FPGAs poseen una mayor densidad de puertas y además contienen módulos de alto nivel, como sumadores o multiplicadores y bloques de memoria empotrados en la propia matriz de interconexionado. Junto con estas características existen algunas otras que hacen que la FPGA sea una arquitectura que posee una mayor flexibilidad en comparación con los CPLDs.

Por otra parte, la principal ventaja que ofrecen las FPGAs en comparación con los convencionales *Application-Specific Integrated Circuits* (ASICs), es que al diseñar un sistema no es necesario su fabricación, sino que la implementación termina con la programación del dispositivo. Esto les da una mayor flexibilidad y una mayor eficiencia en coste respecto a los ASICs, siempre y cuando el volumen de producción no sea excesivamente grande. A estas carecterísticas hay que añadir por supuesto una de las principales de las FPGAs, y es su propiedad de ser reprogramable, con lo que un diseño programado en la FPGA cuando ya no es útil, puede ser borrado y reemplazado por un nuevo diseño, incluso a distancia, algo

que con los ASICs es imposible de realizar.

Una FPGA es programable a nivel hardware, aunque actualmente existen dispositivos programables que llevan además procesadores empotrados capaces de ejecutar programas específicos con lenguajes de alto nivel. Debido a esto, proporcionan las ventajas de un procesador de propósito general y un circuito especializado (sistema empotrado).

La utlización de FPGAs se está extendiendo en todos los ámbitos del diseño de sistemas digitales debido a que los costes y el tiempo de desarrollo son mucho menores. En muchos casos ni siquiera se procede a la posterior fabricación del circuito final, sino que este se utiliza directamente la FPGA.

Aplicaciones

Gracias a las propiedades más importantes de las FPGAs; a saber: gran flexibilidad, capacidad de procesado y a la rápida implementación de circuitos, estos dispositivos se emplean habitualmente en:

- Sistemas aeronáuticos y aeroespaciales.
- Aplicaciones nucleares con restricciones de tiempo real muy severas.
- Módulos de comunicación.
- Procesamiento de señales digitales.
- Formación en el diseño de sistemas hardware.
- Simulación y depuración en el diseño de microprocesadores y microcontroladores.

Arquitectura

Aunque específicamente el dispositivo que se utilizará para este proyecto será una FPGA Xilinx XC5VLX110T, las diferencias importantes en arquitectura respecto de otros modelos de la misma familia Virtex-5 que puedan influir en los resultados de este trabajo, no

distan más allá de su tamaño, es decir, el número de módulos programables o el número y tamaño de los módulos de memoria. Así, la arquitectura básica de las FPGAs de la familia Virtex-5 consiste en una matriz de interconexionado que comunica los módulos *Configurable Logic Blocks* (CLBs), *Digital Signal Processing* del tipo 48E (DSP48E), *Input/Output Blocks* (IOBs), *Blocks RAM* (BRAMs), *Clock Management Tile* (CMT) (evolución de los *Digital Clock Managers* (DCMs)).

CLBs: Los CLBs son los principales elementos configurables de las FPGAs, en ellos se encuentran las *LookUp-Tables* (LUTs), *Flip-Flops*, *Arithmetic and Carry Chains* (Figura 2.2).

Slices	LUTs	Flip-Flops	Arithmetic and Carry Chains	Distributed RAM(1)	Shift Registers(1)
2	8	8	2	256 bits	128 bits

Notes:
1. SLICEM only, SLICEL does not have distributed RAM or shift registers.

Figura 2.2: *Composición de un CLB de la familia Virtex-5*

A diferencia de las anteriores versiones de Virtex, la Virtex-5 solamente posee 2 slices por CLB y están conectados a los demás bloques a través de una matriz de interconexionado (Figura 2.3).

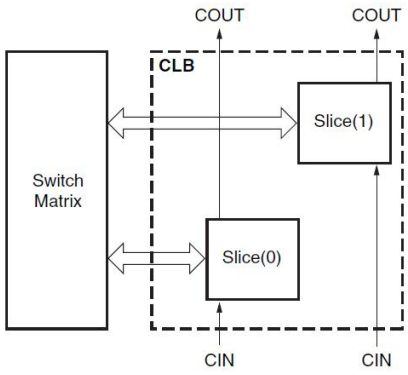

Figura 2.3: *Estructura de un CLB de la familia Virtex-5*

Las LUTs son pequeñas memorias que funcionan como una tabla, las cuales dependiendo de las entradas ofrecen una salida, en función de cómo se hayan programado, como si fuese la

función booleana de las entradas. El hecho de que en las Virtex-5 sólo haya 2 slices es porque las anteriores LUT de 4 entradas y 1 salida booleana se habían vuelto extremadamente limitantes y requerían muchos niveles de lógica para implementar código complejo. Por eso en las versiones Virtex-5 aunque sólo haya 2 slices, las 4 LUT que tiene internamente cada una han sido extendidas a 6 entradas, pudiendo utilizarse como funciones booleanas de 6 entradas y 1 salida o como 2 funciones booleanas de 5 entradas y 1 salida para cada función.

Además en la familia Virtex-5 exiten dos tipos de slices: los slices-L y los slices-M. Los slices de tipo L se usan cuando se vayan a implementar en ellos únicamente lógica combinacional, mientras que los del tipo M se utilizan cuando es necesario almacenar memoria. En la figura 2.4 se puede apreciar la estructura de un slice del tipo L de la familia Virtex-5.

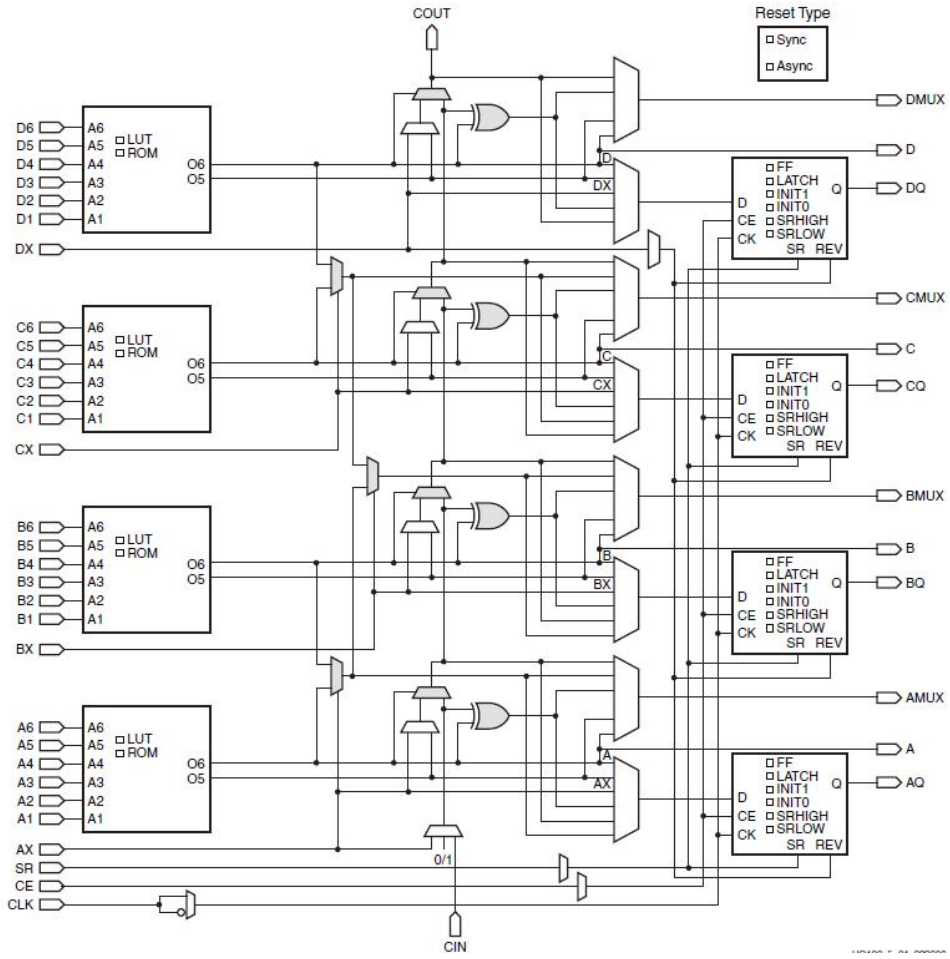

Figura 2.4: *Estructura de un slice-L de la familia Virtex-5*

DSP48E: Los DSPs son módulos específicos que se utilizan para realizar operaciones aritmético-lógicas. Aunque son módulos específicos y no muy flexibles, su uso se basa en que si las operaciones aritmético-lógicas que realizan se programasen en CLBs, ocuparían muchos módulos y recursos. Los DSP48E están compuestos por un multiplicador de 25x18 bits, un sumador de 48 bits y un acumulador (Figura 2.5).

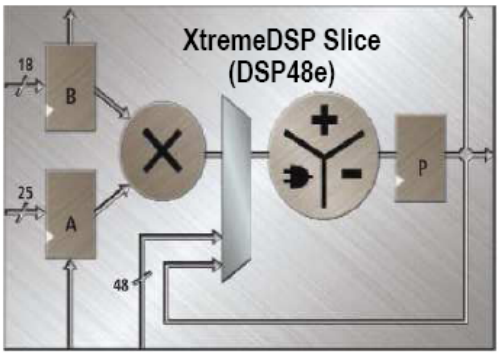

Figura 2.5: *Estructura de un DSP48E de la familia Virtex-5*

IOBs: Los IOBs son los módulos que comunican la FPGA con el exterior y suelen estar situados rodeando la matriz de bloques lógicos. Estos módulos, como los CLBs o DSPs, se pueden programar para que se comporten como puertos de salida, entrada o entrada-salida (Figura 2.6).

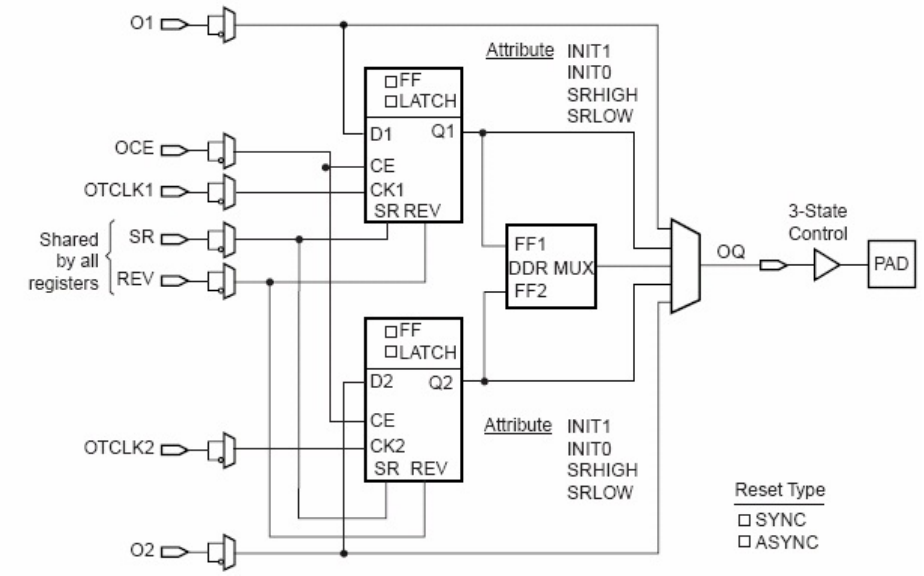

Figura 2.6: *Estructura de un IOB de la familia Virtex*

BRAM: Los BRAM son bloques de memoria RAM empotrados y se sitúan en columnas a lo largo de la FPGA. Tienen 36 KB de memoria cada uno. Su configuración no es interesante en este estudio ya que sólo se utilizarán como memorias para almacenar datos.

CMT: Los módulos DCMs presentes en los modelos antiguos de Virtex, han sufrido una evolución a los llamados CMT, los cuales están compuestos por 2 DCMs y un *Phase-Locked-Loop* (PLL) (Figura 2.7). Los DCMs son módulos que manipulan la señal de reloj, pudiendo multiplicar o dividir su frecuencia, recondicionar el *duty cycle* de la señal de reloj, y elimina los *clock skew*. El componente PLL, es un módulo analógico con realimentación, cuyo objetivo es que la señal de reloj que se genera sea lo más perfecta posible.

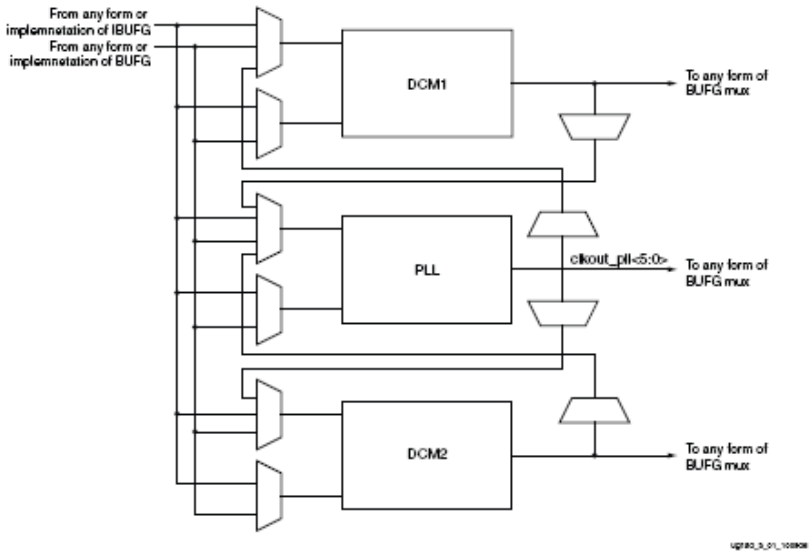

Figura 2.7: *Estructura de un CMT de la familia Virtex-5*

2.1.3. Virtex-5 XC5VLX110T

Aunque todo el estudio de inserción de bitflips presentado en este trabajo se ha realizado sobre cualquier FPGA del tipo Virtex, la implementación final se ha realizado sobre una FPGA Virtex-5 XC5VLX110T y las características que tenemos que tener en cuenta son:

CLBs: Posee una matriz de 160 x 54 (fila x columna) de CLBs con un total de 17.280 Slices y un máximo de 1.120 Kb de memoria RAM entre todos los CLBs.

DSP48E: En este tipo de FPGA se pueden encontrar una columna de DSP48E con un total de 64 módulos.

BRAMs: Se puede hacer uso de un total de 148 BRAMs de un tamaño de 36 Kb y cada bloque puede ser también usado como dos independientes de 18 Kb cada uno, teniendo un total de 296 bloques. En total se puede contar con 5.328 Kb de memoria RAM distribuida entre todos los bloques BRAM.

CMTs: Para manipular la señal de reloj, este tipo de FPGA cuenta con 6 CMTs compuestos cada uno por 2 DCMs y 1 PLL.

Estos módulos serán bastate referenciados en el transcurso de esta memoria, en especial los CLBs y DSPs, por lo que es necesario tenerlos en cuenta en los posteriores capítulos.

2.2. Herramientas Software

Para la realización de la parte software de la plataforma de inyección de errores, se han utilizado distintas herramientas, sobre todo de Xilinx: Xilinx ISE 12.1, Xilinx EDK 12.1, Xilinx SDK 12.1, iMPACT 12.1 y PlanAhead 12.1, para la parte relacionada con la implementación sobre la FPGA. La herramienta software que se ha utilizado para desarrollar el programa en el PC ha sido NetBeans.

2.2.1. Xilinx ISE

Xilinx *Integrated Software Environment* (ISE) es la herramienta de diseñeo de sistemas digitales que se ha utilizado, más concretamente la versión 12.1. Con esta herramienta se pueden crear, verificar, simular, sintetizar e implementar diseños basados en una FPGA o CPDL (Figura 2.8). Ha sido utilizado para crear los códigos de lenguaje VHDL de los diseños o circuitos que se pretenden testear, verificarlos, sintetizarlos y simularlos, para comprobar que eran correctos.

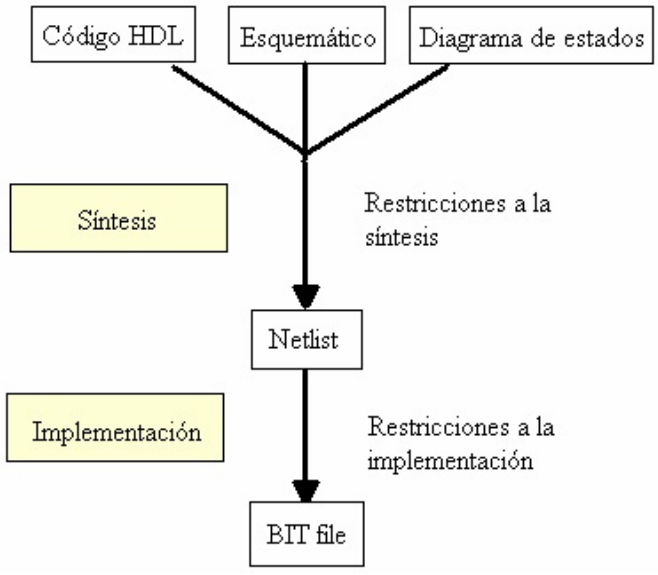

Figura 2.8: *Flujo de diseño de circuitos en Xilinx ISE*

El editor que se utiliza dentro de esta herramienta para realizar el diseño es el *Project*

Navigator. Los diseños se pueden programar en código de lenguaje VHD (VHDL o Verilog), o mediante esquemáticos y diagramas de estados, transformándolos en código.

Una vez generado el diseño en el editor, en este caso en particular programado en lenguaje VHDL, se puede sintetizar. La función de síntesis transforma el código inicial a una *netlist* descrita en un lenguaje interno para un hardware concreto. Se definen restricciones temporales, asignación de pines y optimizaciones de cada uno de los componentes a partir de la descripción de comportameinto del sistema.

El paso de implementación está dividido en tres fases: *translate*, *map* y *place&route*. Lo que se hace en este proceso es enlazar los módulos con las condiciones indicadas en el fichero de restricciones (.ucf) y reducirla a primitivas de Xilinx. A continuación se adapta al dispositivo en el cual se va a volcar el diseño y se colocan y se rutan los componentes físicos finales. En el fichero de restricciones (.ucf) se especificarán los pines a los que se deberá conectar cada señal de entrada y salida y su tecnología. Se pueden añadir restricciones temporales, como retardo máximo de una señal o frecuencia del reloj, y colocación de bloques de diseño del circuito en la región deseada de la FPGA.

El último paso será la generación del bitstream (.bit) que será lo que finalmente se vuelque en la memoria de configuración de la FPGA. Este fichero contiene toda la información para que la FPGA sea configurada por completo, desde la localización de los dispositivos, hasta los elementos de rutado.

2.2.2. Xilinx EDK

La herramienta Xilinx *Embedded Development Kit* (EDK) permite diseñar sistemas empotrados complejos para luego configurarlos sobre la FPGA. Este software integra a Xilinx ISE y usa los procesos de síntesis e implementación de la misma forma para el sistema empotrado que se desea implementar. Además, permite realizar aplicaciones software que luego irán mapeadas en memoria y serán ejecutadas por los distintos procesadores que se implementen en el sistema. Con esta herramienta se ha generado el sistema empotrado con

el que se programará la FPGA (Figura 2.9).

A diferencia del *Project Navigator* de Xilinx ISE, EDK provee de módulos hardware de más alto nivel para desarrollar los sistemas, llamados *cores*. EDK permite desarrollar un sistema, añadiendo y conectando los distintos cores que se vayan introduciendo en el sistema, proporcionando una visión más global. Los distintos cores disponibles pueden ser desarrollados por Xilinx o, por el propio u otros usuarios e integrarlos al sistema simplemente añadiéndolos y luego conectándolos al resto de cores. Para cada uno de ellos se ha de seleccionar la región de memoria donde va a ir mapeado su software, dependiendo de en qué bus esté conectado. También se puede configurar la estructura de los buses y cómo los módulos están conectados a ellos.

2.2.3. Xilinx SDK

Xilinx *Software Development Kit* (SDK) es una versión modificada de Eclipse que está integrado con Xilinx EDK, obteniendo así un IDE donde poder desarrollar y compilar el código en lenguaje C de los procesos software que el procesador del sistema empotrado hardware ejecutaría. Para ello, en un proyecto SDK se mantiene una dependencia con el hardware diseñado en el EDK, importando las librerías del hardware del sistema diseñado.

Este software también proporciona una serie de funciones de manejo básico de los cores añadidos al sistema, pudiendo así accedir facilmente a ellos.

2.2.4. iMPACT

iMPACT es el software desarrollado por Xilinx para programar sus FPGAs. Permite comunicarse y configurar la FPGA de varias formas, además de que se realiza automáticamente mediante *boundary scan*. Una vez que se ha establecido la conexión entre la FPGA e iMPACT, solo hay que determinar qué bitstream utilizar y la FPGA se programará con ese bitstream de configuración.

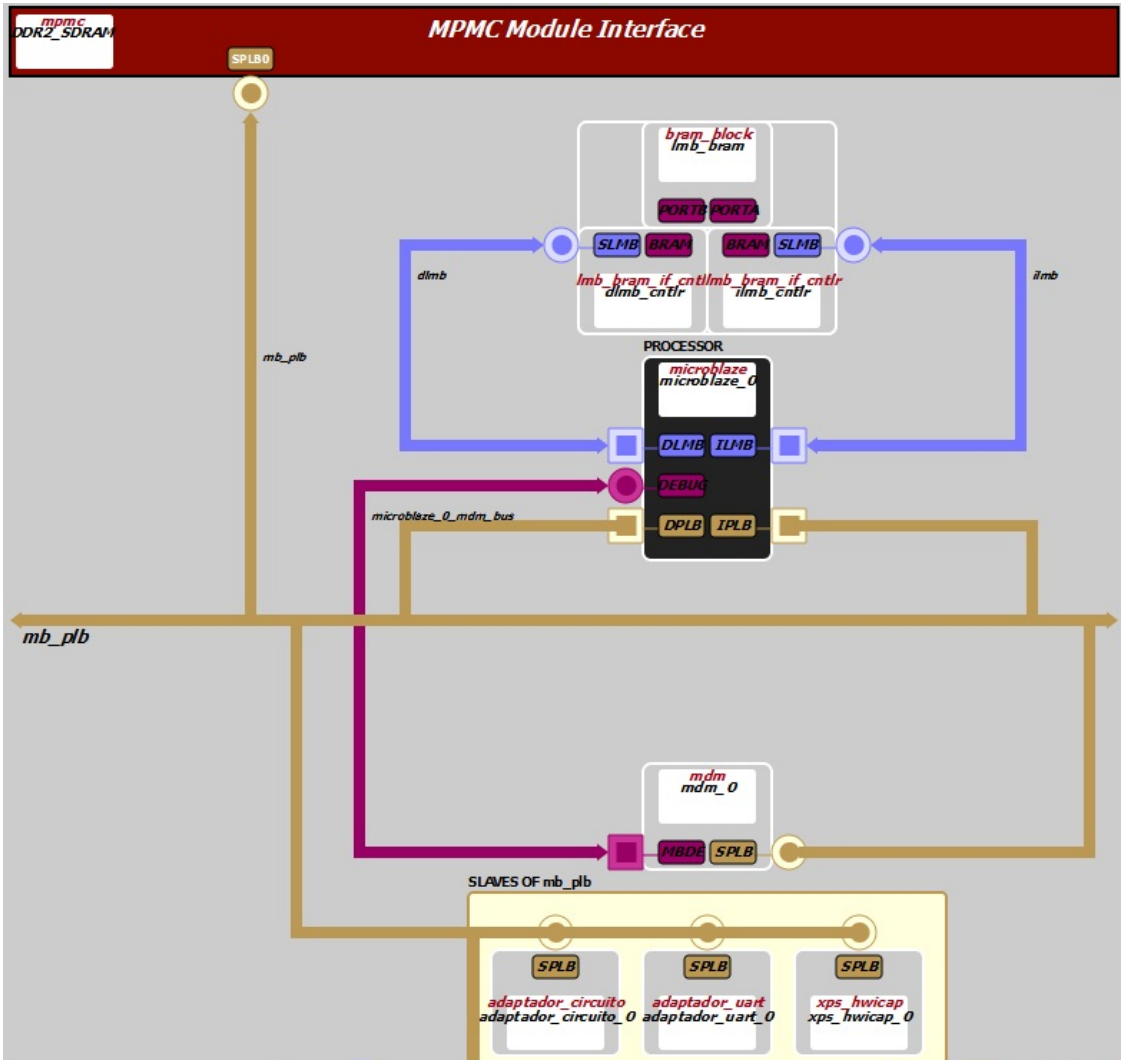

Figura 2.9: *Visión general del sistema empotrado en EDK*

2.2.5. PlanAhead

Esta herramienta sirve para decidir dónde ubicar cada uno de los módulos del sistema, definidos por las *netlist* correspondientes (.ngc). También permite decidir qué módulos van a ser reconfigurables parcialmente y cuáles van a ser módulos estáticos del sistema. Por último, permite realizar el proceso de implementación y generar el fichero de configuración (.bit), tanto el global como los de los módulos parciales, en caso de que existan.

2.2.6. NetBeans

Como editor y compilador para crear las aplicación en código JAVA, se ha utilizado el editor y compilador NetBeans. Esta herramienta permite una fácil construcción de la interfaz de usuario, una amplia visión general del proyecto y diferentes ayudas que le hacen ser el editor y compilador JAVA preferido por nuevos desarrolladores.

Capítulo 3

NESSY: una plataforma de depuración de circuitos

Como se ha comentado anteriormente, la plataforma de inyección de errores presentada en este trabajo parte de otra plataforma de depuración de errores para Virtex-5 llamada NESSY, desarrollada en la Facultad de Informática de la Universdad Complutense de Madrid en el proyecto de fin de carrera de Felipe Serrano "*Desarrollo de una plataforma de inyección de errores sobre hardware reconfigurable*". La arquitectura de la plataforma NESSY (Figura 3.1) consiste en un sistema software que se ejecuta en un PC y un sistema hardware basado en microprocesador implementado sobre una FPGA, donde también se implementa el circuito bajo prueba. El programa ejecutado en el PC está escrito en el lenguaje de programación JAVA, y el procesador del sistema hardware está implementado como un Microblaze y se programa con el lenguaje de programación C.

La comunicación entre el PC y la FPGA se hace por medio de dos canales, uno para configuración y el otro para envío y recepción de datos. Para la configuración de la FPGA se utlizara el puerto JTAG de la placa XUPV5 que soporta la FPGA, mientras que para la comunicación de datos se utilizara la UART de la placa, utilizando para ello el protocolo RS232. Además de enviar y recibir datos, el canal de datos también tiene la finalidad de sincronizar las ejecuciones del proceso software del PC y el programa ejecutado en el Microblaze de la FPGA. Sin embargo, el protocolo RS232 es muy lento por lo que su uso, ya sea para

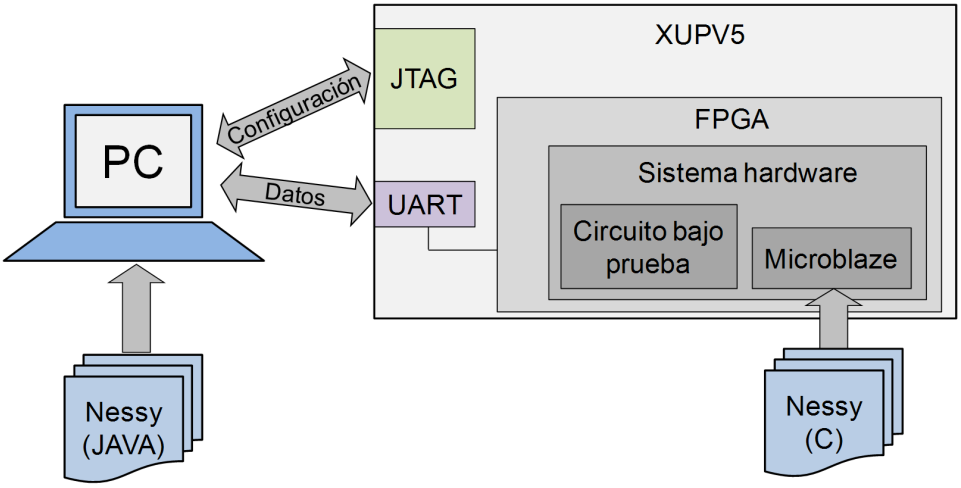

Figura 3.1: *Arquitectura de la plataforma NESSY*

sincronización, envío o recepción de datos, ha sido limitado al mínimo imprescindible.

3.1. Arquitectura Hardware de NESSY

La creación de un sistema independiente en la plataforma hardware vino dada sobre todo por el cuello de botella que producía el uso del interfaz serie de comunicación de la UART de la placa XUPV5, usando el protocolo RS232. Como se ha comentado anteriormente, es un protocolo excesivamente lento y dado que el volumen de datos a manejar es considerablemente grande, se optó por la creación de un sistema hardware con la ayuda de la herramienta EDK. Dicho sistema hardware (Figura 3.2) está formado por un procesador Microblaze conectado mediante un bus LMB (*Local Memory Bus*) a sus memorias de instrucciones y datos. Además tiene una memoria DDR2 RAM adicional, un adaptador para la UART de la placa y una interfaz que sirve para adaptar el circuito a testear. Todos estos módulos están conectados a un bus PLB por el que se comunicarán a una frecuencia de reloj de 100 MHz.

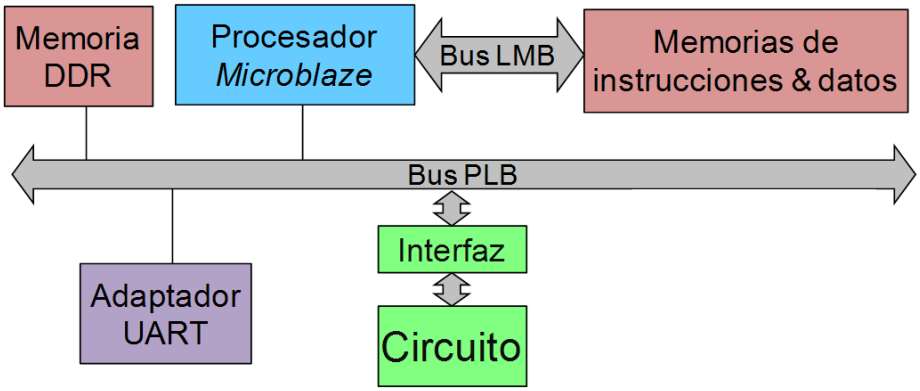

Figura 3.2: *Sistema hardware*

3.1.1. Microblaze

El procesador empotrado Microblaze es un *soft-processor* de arquitectura RISC, optimizado para su implementación en FPGAs de Xilinx. Es un procesador altamente configurable, que permite seleccionar un conjunto específico de características requeridas por su diseño (Figura 3.3). Posee registros de propósito general de 32 bits, instrucciones de 32 bits con 3 operandos y 2 modos de direccionamiento, y direcciones de bus de 32 bits.

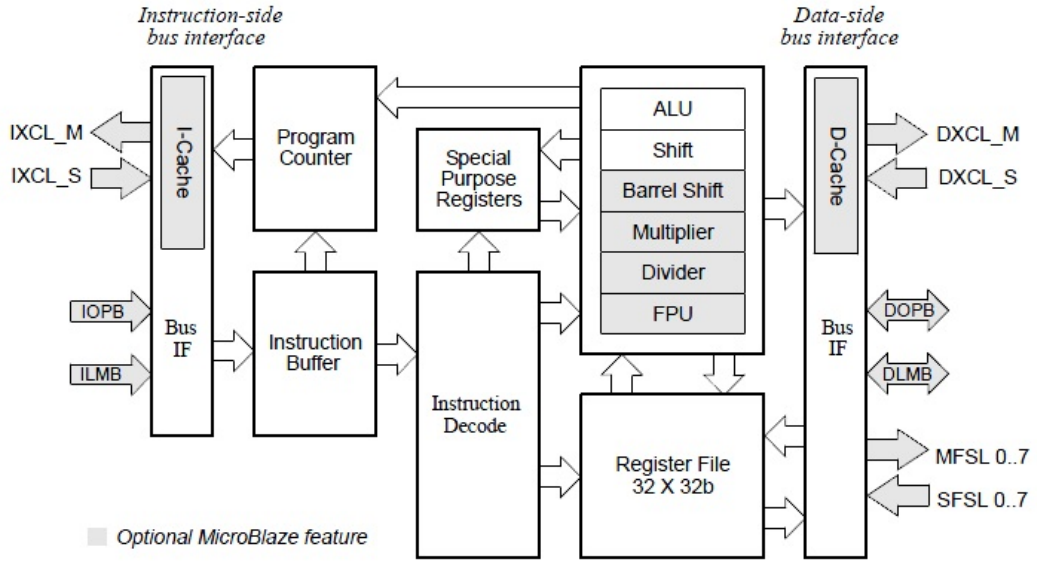

Figura 3.3: *Diagrama funcional del procesador Microblaze*

La versión de Microblaze usada en este proyecto ha sido la 7.30a y se usa para el proceso y gestión del hardware añadido a la plataforma. Este procesador está escuchando el puerto serie a la espera de algún comando enviado desde el sistema software del PC y según sea el comando recibido se encargará de ejecutar el programa situado en una memoria RAM interna de la FPGA a la que está conectada por el bus LMB.

3.1.2. Buses PLB y LMB

Para la implementación del sistema hardware desarrollado durante este proyecto, se ha utilizado un bus PLB para comunicar el precesador Microblaze con el resto de los componentes del sistema.

El bus LMB es un bus especial usado solamente para conectar el Microblaze con los puertos de datos e instrucciones a dispositivos de alta velocidad, normalmente memorias BRAM.

En este proyecto, se han usado también dos buses LMB conectados al Microblaze que van a dos controladores de BRAM, uno para datos y otro para instrucciones, para poder leer a la vez un dato y una instrucción (Figura 3.4).

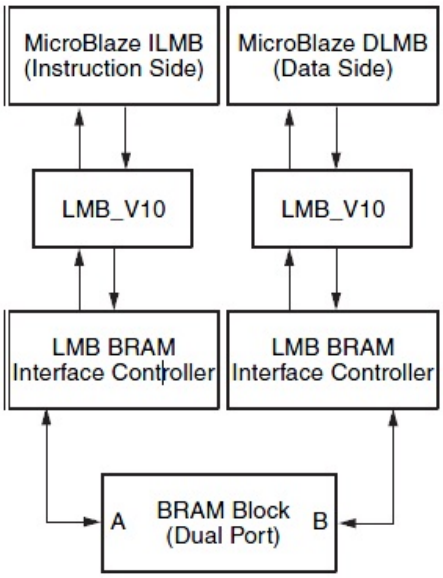

Figura 3.4: *Uso del bus LMB conectado al Microblaze*

3.1.3. Memoria DDR2 RAM

Debido al volumen de datos que se procesan en la plataforma de inyección de errores presentada en este trabajo y a que se pretende utilizar la comunicación por el puerto serie lo menos posible, ha sido necesario almacenar todos estos datos en la memoria DDR2 RAM de la placa XUPV5. La placa XUPV5 posee una memoria DIM DDR2 RAM de 512MB. El controlador utilizado para acceder a la RAM, que directamente provee Xilinx, es el controlador de memoria multi-puerto. Este controlador proporciona acceso a memoria desde 1 hasta 8 puertos, y permite conectarlos a los Microblaze a través de los buses PLB u OPB.

3.1.4. *Adaptador UART*

Este módulo es el encargado de comunicar la placa con el exterior mediante el protocolo de comunicación RS232. Posee una interfaz de comunicación con el bus PLB, de forma que mediante la escritura/lectura de diferentes registros, el Microblaze puede intercambiar datos con el exterior.

3.1.5. *Adaptador circuito*

El uso del módulo *Adaptador circuito* es para generalizar el sistema, de tal forma que cualquier diseño que se utilice en la herramienta NESSY se pueda conectar al bus PLB a través de este adaptador. Este módulo genérico tiene 1 entrada como señal de reloj para el circuito a testear, 1 entrada de 32 bits como datos y 1 salida del circuito de 32 bits. Esto indica que con este adaptador se puede trabajar con circuitos que tengan 1 reloj, 1 entrada y 1 salida de hasta 32 bits.

3.1.6. *Circuito*

Simplemente es el módulo generado automáticamente por NESSY mediante el código vhdl que se le proporcione y que irá conectado al módulo *Adaptador circuito* para que se conecte al PLB y al resto de componentes.

3.2. Sistema Software

El sistema software de NESSY en el PC consiste en el interfaz de usuario y controla el flujo de ejecución de las tareas ejecutadas en el Microblaze. Cuando se realiza una operación en esta aplicación, NESSY lanza un comando a través del puerto serie, el cual está escuchando el Microblaze de la FPGA y por tanto actuará en consecuencia. Así, ademas de controlar el flujo de ejecución, se mantiene una comunicación y una sincronización entre ambas plataformas.

La ejecución de todos estos pasos está controlado por el usuario a través de la interfaz gráfica (GUI) (Figura 3.5). La GUI se ejecuta en el PC y, cuando el usuario da una orden, envía un comando a la FPGA para que el Microblaze realice su trabajo.

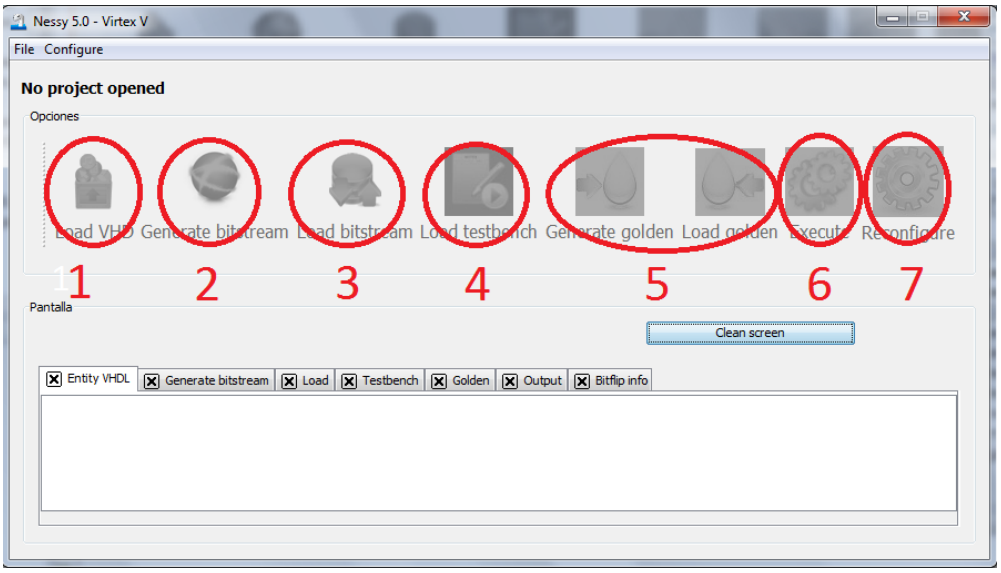

Figura 3.5: *GUI de NESSY*

Los distintos botones disponibles en el menú permiten realizar las siguientes funciones:

1. *Load HDL*
2. *Generate bitstream*
3. *Load bitstream*
4. *Load testbench*

5. *Generate/Load golden*

6. *Execute*

7. *Reconfigure*

3.2.1. Cargar HDL (*Load HDL*)

En este primer paso se selecciona el archivo con el código *top* del circuito bajo test (.vhd), el cual se parsea para obtener el número de entradas, salidas y sus nombres, para posteriormente poder adaptarlo con el módulo *Adaptador circuito*.

3.2.2. Generar bitstream (*Generate bitstream*)

En el segundo paso NESSY realiza automáticamente el proceso de síntesis e implementación del sistema hardware junto con el código del circuito previamente cargado,y así terminará generando el bitstream para configurar la FPGA. Este proceso se realiza usando las herramientas Xilinx ISE 12.1, EDK 12.1 y PlanAhead12.1.

3.2.3. Cargar bitstream (*Load bitstream*)

En este paso, usando la herramienta de configuración de FPGAs Xilinx iMPACT 12.1, se realiza la configuración completa del sistema hardware en la FPGA. Tras la ejecución de este comando, la FPGA, el Microblaze está esperando a que lleguen a través del puerto serie los comandos para ejecutar.

3.2.4. Cargar testbench (*Load testbench*)

La ejecución del circuito testeado se realiza mediante el uso de un testbench, una serie de valores para las señales de entrada, las cuales se encuentran en un fichero. Cuando se ejecuta este paso, la plataforma en el PC envía al Microblaze un comando para indicar que va a enviar por el puerto serie dicho fichero de testbench, así como el tamaño del mismo,

para luego proceder a enviar el testbench por completo. El Microblaze irá leyendo por la entrada de la UART todos estos datos, y los almacenará en la memoria DDR2 RAM.

3.2.5. Generar/Cargar golden (*Generate/Load golden*)

Esta orden sirve para la generación o carga de un archivo con los resultados correctos de la ejecución del circuito sin fallos. Para generar el *golden*, simplemente se ejecuta el tesbench seleccionado, explicado en el paso 3.2.4, en el circuito bajo test. El Microblaze recibe el comando para generar el resultado *golden*, coloca los datos del testbench en la entrada del circuito y guarda la salidas generadas por la ejecución del circuito en una zona de la memoria DDR2 RAM asignada para el *golden*. La segunda opción sería cargar el golden por medio de un archivo, el cual sería enviado por el puerto serie y el Microblaze se encargaría de ir cogiendo los datos por la UART e ir enviándolos a la zona de la memoria DDR2 RAM asignada para el *golden*.

3.2.6. Ejecutar (*Execute*)

Con esta opción, se envía al Microblaze el comando de ejecutar, con lo que el Microblaze ejecuta el circuito bajo prueba usando como entradas los datos del testbench. Una vez que se ejecuta el circuito, se comprueban las salidas con su correspondiente dato del *golden*. Una vez terminada la ejecución y comparación de los resultados, se devuelven al PC a través del puerto serie los datos obtenidos, ya sea si ha habido un fallo y en qué ciclo de reloj ha surgido o si por el contrario la ejecución ha sido correcta.

3.2.7. Reconfigurar (*Reconfigure*)

Este paso realiza la evaluación exhaustiva del impacto de la inyección de bitflips en el circuito bajo prueba. Se explicará con más detenimiento en el próximo capítulo, ya que es la principal contribución de este trabajo.

Capítulo 4

Metodología de inyección de errores

En este capítulo se entrará en detalle sobre las funcionalidades añadidas para convertir a NESSY en una plataforma de inyección de errores basada en hardware, para FPGAs tipo Xilinx Virtex-5 y con las ventajas que se proponían en los objetivos 1.3.

Para conseguir realizar la inyección de errores, es decir, insertar bitflips en un tiempo mínimo, se ha añadido un nuevo módulo, el *Internal Configuration Access Port* (ICAP), el cual sirve para reconfigurar internamente la FPGA sin necesidad de hacerlo externamente a través del puerto externo JTAG. Con esto se libera la plataforma del problema de las comunicaciones entre el PC y la FPGA y se le da mayor independencia a la FPGA.

La idea de reconfigurar internamente la FPGA viene ligada a la idea de inyectar los errores en el bitstream parcial de configuración del circuito al que se quiere testear mediante la inserción de bitflips. Para ello, haciendo uso de PlanAhead, como se ha explicado anteriormente, se crea un bitstream parcial únicamente del circuito a testear. Éste se envía a la memoria DDR2 RAM de la FPGA donde queda almacenado para su posterior tratamiento. Esto solamente ocurre una vez, por lo que sólo habrá que enviar el bitstream parcial una vez a través del puerto serie RS232, limitando así el uso de de este protocolo tan lento. A continuación, el programa que ejecuta el procesador Microblaze se encarga de insertar el bitflip, reconfigurar el módulo del circuito bajo test, y ejecutarlo para obtener los errores. La ejecución tiene lugar haciendo uso de un tesbench proporcionado por el usuario, consistente en un conjunto de valores representativos de las entradas del circuito bajo test. Los resultados

obtenidos se comparan entonces con los valores correctos del circuito o *golden*. Estos resultados *golden* se han obtenido previamente en una ejecución del circuito original sin errores. Así se comprueba si la inserción del bitflip ha producido un error en el funcionamiento del circuito y por lo tanto en su ejecución.

Como es de esperar, ha habido que introducir cambios en el sistema hardware (ICAP) y también en el sistema software, tanto en el software de la aplicación NESSY en la plataforma PC como el software que se ejecuta en el procesador Microblaze. Además, para poder construir un bitstream de reconfiguración parcial, se ha tenido que hacer un estudio del protocolo de configuración, así como de los comandos que constituyen el bitstream parcial. Para ello, se ha realizado el diseño de una aplicación en código JAVA que lee un bitstream de configuración y crea un fichero de texto en el que se desglosa y se muestra de una forma vistosa los comandos que comprende un bitstream.

4.1. Modificaciones en el Sistema Hardware

Como se ha comentado en la introducción de este capítulo, para darle independencia a la plataforma hardware y evitar la configuración de la FPGA externamente mediante el puerto JTAG, se ha añadido al sistema hardware existente de NESSY el módulo ICAP (Figura 4.1), que permite hacer reconfiguraciones parciales internamente.

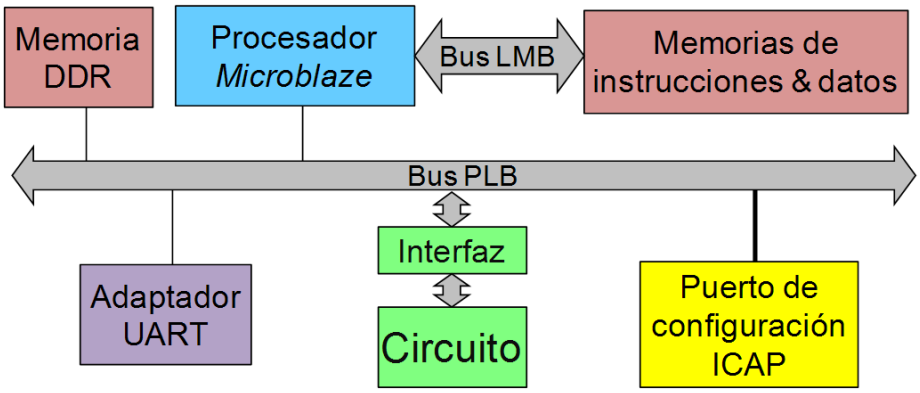

Figura 4.1: *Nuevo sistema hardware*

4.1.1. ICAP

El módulo ICAP proporciona una interfaz interna de reconfiguración parcial para las FPGAs de la familia Xilinx Virtex-5. Existen otras opciones más lentas, como los protocolos SelectMap o JTAG que necesitan un controlador externo para enviar los datos de reconfiguración, lo cuál se pretende evitar. El módulo ICAP está conectado al bus PLB y de ahí a los demás componentes. Permite la lectura y escritura de CLB LUTs y de las propiedades de los CLB Flip-flops. Ofrece soporte para los procesadores empotrados Microblaze y trabaja a una frecuencia máxima de 100 MHz en la Virtex-5.

El módulo añadido al sistema hardware es un core que ofrece Xilinx (hwicap v4.00a) para la Virtex-5. Para poder utilizar este core, el bitstream parcial se almacena en la memoria BRAM de la FPGA, y una vez que el Microblaze activa la señal de iniciar la reconfiguración, el ICAP se encarga de ir leyendo el bitstream de configuración de la BRAM a través del bus PLB e ir escribiéndolo en la memoria de configuración de la FPGA.

Las primitivas que se utilizan para este módulo y que ejecuta el Microblaze son:

XHwIcap_LookupConfig(): función que se encarga de configurar el ICAP dependiendo del dispositivo.

XHwIcap_CfgInitialize(): función que sirve para inicializar el ICAP.

XHwIcap_DeviceWrite(): función que lee los comandos de configuración y los escribe en la memoria de configuración de la FPGA.

4.2. Modificaciones en el Sistema Software

Tanto la plataforma software que se ejecuta en el PC como la que se ejecuta en la FPGA han ido sufriendo una serie de cambios para así transformar la plataforma en una herramienta de inyección de errores. Para realizar la inyección de errores hay que llevar a cabo un proceso en el cual están implicadas ambas plataformas software, tanto la del PC como el software ejecutado en la FPGA.

Los pasos del sistema software que han sufrido cambios para así obtener una plataforma de inyección de errores son:

- *Generate bitstream*: En la nueva versión de este paso, que era el segundo paso del proceso de depuración de errores, el usuario debe indicar la región de la FPGA donde se debe colocar el circuito bajo test. Entonces NESSY realiza automáticamente el proceso de síntesis e implementación del sistema hardware junto con el código del circuito previamente cargado, y así terminará generando el bitstream para configurar la FPGA. Ahora NESSY también tiene que generar otro bitstream parcial para la región de la FPGA que le ha sido indicada por el usuario; en este bitstream parcial es donde se inyectarán los bitflips. Este proceso se realiza usando las herramientas Xilinx ISE 12.1, EDK 12.1 y PlanAhead 12.1.

- *Reconfigure*: Este paso realiza una evaluación exhaustiva del impacto de la inyección de un bitflip en el circuito bajo prueba. Para ello, se lanzan dos procesos: uno en el PC que envía un comando a través del puerto serie, y se inicia otro en el Microblaze. Ambos procesos se ejecutan en paralelo y se sincronizan haciendo uso del puerto serie y del protocolo de comunicación RS232.

Para que en el proceso de generación del bitstream parcial mediante la herramienta PlanAhead 12.1 sitúe el circuito en la región de la FPGA que ha indicado el usuario, se utilizó la opción de crear un AREA_GROUP para el módulo controlador de la memoria DDR2 RAM, y otro sólo para el circuito bajo test, y un último para el resto de elementos

del sistema hardware (que en lo sucesivo llamaremos "sistema hardware estático". Dichos AREA_GROUPs se declaran en el fichero de restricciones del usuario (.ucf) y se les asigna una región en forma de rectángulo de la FPGA, simplemente especificando las coordenadas de los slices de las esquinas inferior izquierda, y la esquina superior derecha (Figura 4.2). Precisamente éstos son los datos de coordenadas que se solicitan al usuario al principio de la opción de generar el bitstream (*Generate bitstream*).

```
INST "adaptador_circuito_0" AREA_GROUP = "AG";
INST "adaptador_uart_0" AREA_GROUP = "AG";
INST "clock_generator_0" AREA_GROUP = "AG";
INST "dlmb" AREA_GROUP = "AG";
INST "dlmb_cntlr" AREA_GROUP = "AG";
INST "ilmb" AREA_GROUP = "AG";
INST "ilmb_cntlr" AREA_GROUP = "AG";
INST "lmb_bram" AREA_GROUP = "AG";
INST "mb_plb" AREA_GROUP = "AG";
INST "mdm_0" AREA_GROUP = "AG";
INST "microblaze_0" AREA_GROUP = "AG";
INST "proc_sys_reset_0" AREA_GROUP = "AG";
INST "xps_hwicap_0" AREA_GROUP = "AG";
AREA_GROUP "AG" RANGE=SLICE_X0Y0:SLICE_X107Y70;

INST "DDR2_SDRAM" AREA_GROUP = "AG_memory";
AREA_GROUP "AG_memory" RANGE=SLICE_X0Y0:SLICE_X11Y159;

INST "circuito_0" AREA_GROUP = "pblock_circuito_0";
AREA_GROUP "pblock_circuito_0" RANGE=SLICE_X24Y140:SLICE_X107Y159;
AREA_GROUP "pblock_circuito_0" RANGE=DSP48_X0Y56:DSP48_X0Y63;
```

Figura 4.2: *Declaraciones de AREA_GROUPs en el fichero de restricciones del usuario*

Para que el proceso de reconfiguración sea lo más rápido posible, había que limitar lo más posible la comunicación entre la plataforma del PC y la FPGA a través del puerto serie. Para ello, todo el trabajo de generación de bitstreams parciales, inserción de bitflips, procesamiento, etc., recae en el Microblaze de la FPGA y de este modo, el PC sólo se encarga de recibir resultados y procesarlos. Así pues, por un lado el proceso que se ejecuta en el Microblaze inyecta los bitflips en la memoria de configuración (donde se encuentran los bits de configuración que implementan el circuito bajo prueba), ejecuta el circuito usando el

testbench, compara los resultados con los del *golden* y envía los resultados de las comparaciones al PC usando el puerto serie. Por otro lado, el proceso que se ejecuta en el PC recibe estos resultados y los imprime en la interfaz gráfica de NESSY, y en un fichero de *log*.

4.2.1. Proceso de reconfiguración en el Sistema Software

Dadas las restricciones, lo único que hace el proceso de la plataforma del PC es enviar a la FPGA una única vez el bitstream parcial del circuito a testear, (que se generó con el botón *Generate bitstream*), mediante el puerto serie usando el protocolo RS232. Hecho esto, el proceso del PC se mantiene sincronizado mientras en la FPGA se van realizando las evaluaciones de las inyecciones de bitflips. Para ello, el PC permanece esperando hasta que recibe una palabra de sincronización por el puerto serie, seguida de los resultados que generó la evaluación de la inyección del bitflip. Estos datos son: el ciclo en el que la ejecución del circuito empezó a fallar, cuál es el valor del resultado erróneo y cuál es el resultado que se debería haber obtenido (*golden*), para poder comparar entre ambos. Estos datos se muestran en la interfaz gráfica de NESSY junto con el bloque, palabra y bit en el que se insertó el bitflip. Esta información la posee directamente el PC al estar sincronizado en todas las iteraciones en las que se inyecta el bitflip, por lo que no necesita usar el puerto serie para ello. Por otro lado, si en la ejecución del circuito no ha habido error, entonces simplemente se indica que la inyección de un bitflip en ese bit no se ha producido ningún error. Toda esta información de la evaluación de las inyecciónes de los bitflips también se imprime en dos archivos de texto o *log*; uno de ellos tiene una estructura similar a la que se imprime en la interfaz gráfica de NESSY, más cómoda para la lectura del usuario, y la otra es en forma de tabla para poder procesar los resultados con la herramienta MATLAB. Una vez que el proceso ha finalizado, al haberse recorrido todos los bits de configuración del circuito bajo prueba y al haberse inyectado un bitflip en cada uno de ellos, también se imprime, tanto en la interfaz gráfica como en los ficheros de *log*, un resumen del proceso de esta evalución exhaustiva. Los datos que se imprimen en este resumen son: el tiempo transcurrido,

el número de reconfiguraciones realizadas, el número de reconfiguraciones erróneas, y el porcentaje de reconfiguraciones erróneas respecto del total de reconfiguraciones realizadas.

4.2.2. Proceso de reconfiguración en el Sistema Hardware

El proceso de inyección de bitflips que se ejecuta en el sistema hardware consiste en 3 pasos bien diferenciados, los cuales están representados en la Figura 4.3.

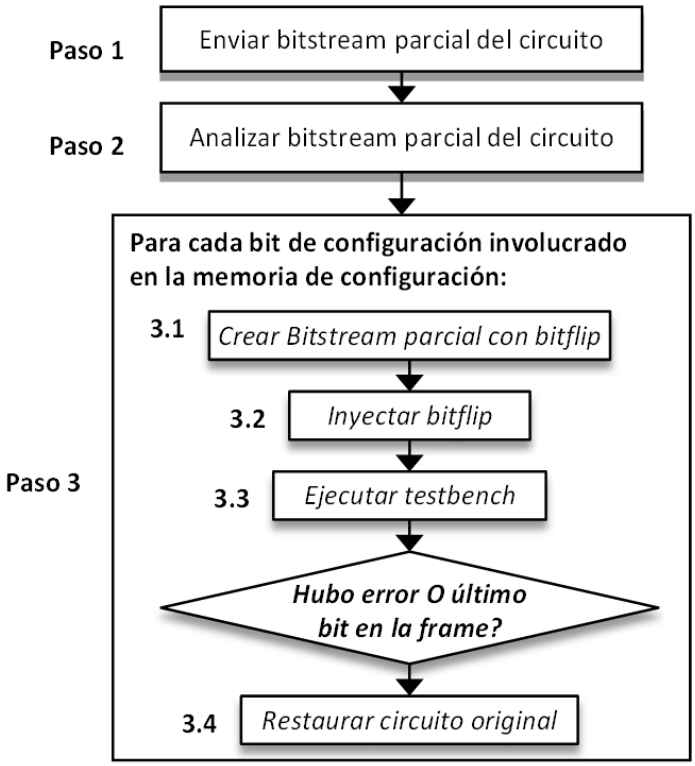

Figura 4.3: *Organigrama del proceso de inyección de bitflips*

En primer lugar (Paso 1), el PC envía al Microblaze el bitstream parcial correspondiente al circuito a testear (sin la inserción de ningún bitflip), y el Microblaze lo guarda en la memoria DDR2 de la FPGA.

En el segundo paso (Paso 2), se hace un análisis de este bitstream parcial, para así identificar sus *mapas de configuración*. Un mapa de configuración es un bloque de intrucciones del bitstream que configuran un conjunto de recursos lógicos adyacentes (CLBs, Block RAMs...) cuyos recursos estén en la misma región de reloj de la FPGA. Una región de reloj comprende

una sección de 20 CLBs de altura y de largo la mitad del dispositivo (Figura 4.4), de modo que cada sección de reloj posee su propia señal de reloj. El proceso de búsqueda de los mapas de configuración será explicado con más detalle en la siguiente sección 4.3.1. Identificar los mapas de configuración es necesario en el paso siguiente para saber la frame específica que se debe modificar en la inyección de un único bitflip (una frame es el segmento mínimo direccionable en la memoria de configuración de la Virtex-5).

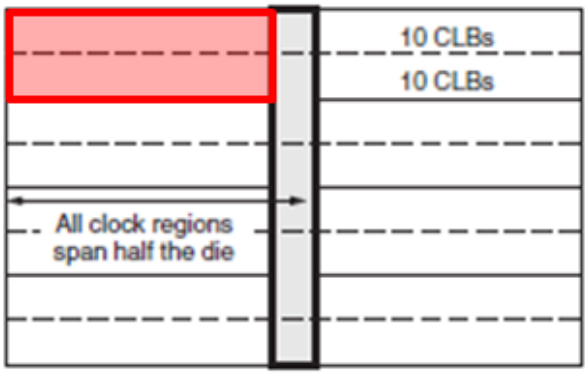

Figura 4.4: *Región de reloj en una FPGA tipo Virtex-5*

Finalmente, en el Paso 3 se realiza la inyección de bitflips para cada bit de configuración en la memoria de configuración que implementa el circuito bajo prueba. Este proceso se compone de 4 subpasos:

En el primero (3.1) se inyecta un bitflip en el bit correspondiente de la memoria de configuración. Para ello, el Microblaze genera un nuevo bitstream, el cual aparece en la Figura 4.3 como “Bitstream parcial con bitflip". Este bitstream sólo contiene una escritura en el registro *Frame Address Register* (FAR) y una escritura de una frame en el registro *Frame Data Register Input* (FDRI). Estos dos comandos escriben el nuevo valor de los bits de configuración (todos los bit iguales al original excepto el del bitflip) para la frame que se corresponde al bit que ha sido modificado. La información obtenida durante el análisis del bitstream del circuito en el Paso 2 se utiliza en este paso para identificar qué frame se debe modificar. El proceso de inyección del bitflip y generación del bitstream parcial está mejor detallado en la siguiente sección 4.3.2.

En el siguiente subpaso (3.2) se carga el bitstream parcial con bitflip en la FPGA usando el puerto de configuración ICAP, para así inyectar el bitflip generado en el subpaso previo. Este bitstream es muy pequeño en comparación con el original que implementa el circuito bajo prueba, debido a que sólo configura una frame.

Una vez que el bitflip ha sido inyectado, en el subpaso 3.3 se ejecuta el testbench en el circuito modificado, se comparan los resultados de la ejecución con los valores almacenados en el *golden* y se envía la comparación de resultados al PC usando el puerto serie. Si ambos resultados son diferentes, la inyección de bitflip ha generado un error.

Finalmente, se restaura el circuito original para eliminar el bitflip inyectado en el subpaso 3.2. Para ello, se carga otra vez el bitstream completo del circuito bajo test (subpaso 3.4). Sin embargo, si el bitflip no ha generado ningún error, se puede restaurar el circuito original al inyectar el siguiente bitflip, ya que el valor escrito en el FDRI para la inyección del bit n restaurará el valor del bit $n-1, n>1$. Esto es cierto a menos que el bit que tiene que ser restaurado sea el último de una frame. Por consiguiente, el circuito original completo se restaura sólo cuando el bitflip haya generado un error. Sin embargo, cuando el bit modificado sea el último de la frame es necesario hacer una restauración especial, para ello en el mismo proceso en el que se genera el "Bitstream parcial con bitflip", también se genera un "Bitstream parcial sin bitflip" o "Bitstream parcial restorer". Como se puede suponer, este bitstream parcial contiene exactamente los mismos datos que el "Bitstream parcial con bitflip" pero sin haber insertado el bitflip, por lo que éste es un bitstream parcial que restaura el bitflip previamente insertado y que sólo reconfigura una frame.

4.3. Proceso de generación de bitflips

4.3.1. Búsqueda de mapas de configuración

Para realizar la búsqueda de los mapas de configuración, hay que leer comando a comando el bitstream parcial almacendo en la memoria DDR2 RAM. Mientras se van leyendo todos los comandos, se van buscando los mapas de configuración del bitstream parcial. Un mapa de configuración es un bloque de intrucciones del bitstream que configuran un conjunto de recursos lógicos adyacentes (CLBs, Block RAMs...) cuyos recursos estén en la misma región de reloj de la FPGA. De este modo, un bitstream contiene tantos mapas de configuración como bloques de recursos lógicos no adyacentes utilice el circuito o aunque los recursos lógicos sean adyacentes, se encuentren en distintas regiones de reloj (Figura 4.5). Cada mapa de configuración se compone de una escritura en un registro de dirección existente en

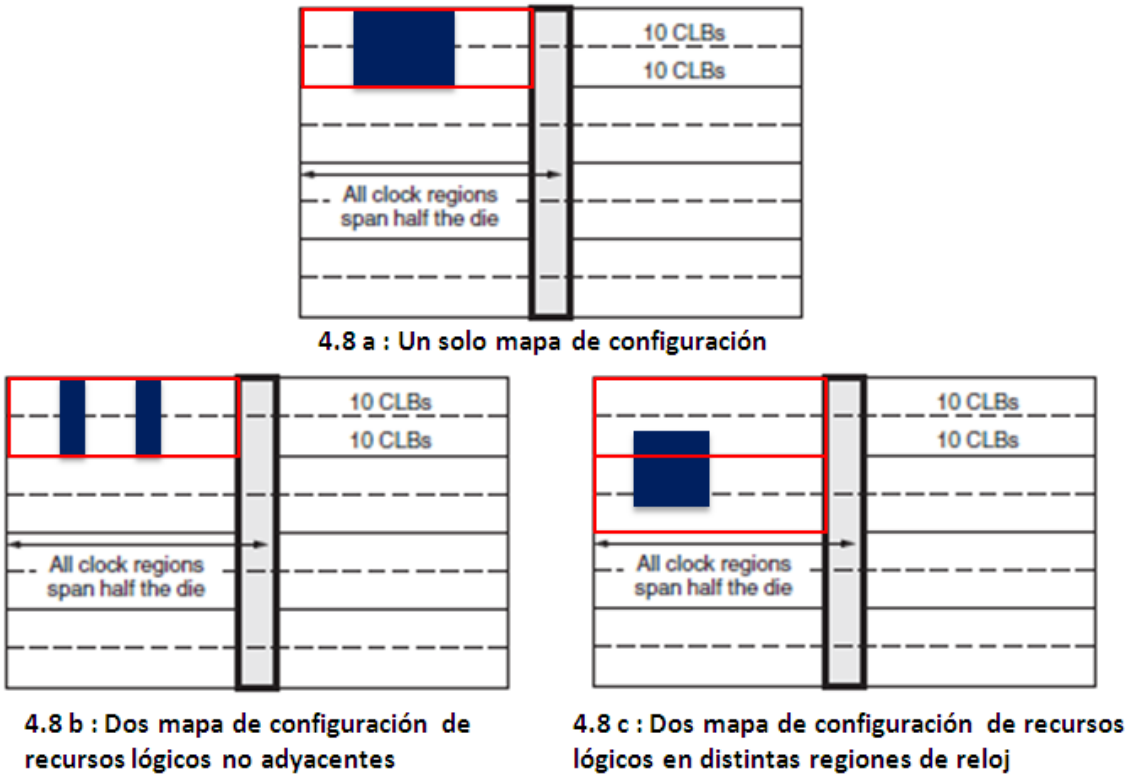

Figura 4.5: *Posibles mapas de configuración*

la circuitería de configuración de la Virtex-5, el FAR, seguido por una o más escrituras en un registro de datos, el FDRI (Apéndice A.1.3). El FAR identifica qué frame será reconfigurada, mientras que el FDRI contiene los bits de configuración que serán cargados en ella. De esta manera, existen tantos mapas de configuración como comandos de escritura en el registro FAR seguidos por un comando de escritura en el registro FDRI. Cada vez que se encuentra uno, se incrementa el contador con el número de mapas de configuración. Cuando se ha leído todo el bitstream parcial, se almacena en una posición de memoria el número de mapas de configuración que se ha sido capaz de encontrar, así como su tamaño. Identificar los mapas de configuración y el tamaño de los mismos, es necesario en el paso de generación del bitstream parcial con bitflip para saber la frame específica que se debe modificar en la inyección de un único bitflip.

4.3.2. Generación del bitstream parcial con bitflip

Para poder construir un bitstream de reconfiguración parcial, primeramente se ha tenido que hacer un estudio del protocolo de configuración, así como de los comandos que constituyen el bitstream parcial (ApéndiceA). Para ello se ha realizado el diseño de una aplicación en código JAVA para leer un bitstream de configuración y crear un fichero de texto en el que se desglosen y se muestren de una forma vistosa los comandos que comprende un bitstream.

Virtex V Configuration Viewer

Virtex V Configuration Viewer es el nombre de la aplicación que ha sido diseñada para desglosar los comandos de configuración que componen un bitstream de configuración. Esta aplicación ha sido programada en código JAVA y se ha utilizado el editor y compilador NetBeans para compilarla. Consta únicamente de un archivo ejecutable llamado *Virtex_ V_ Configuration_ Viewer.exe* debido a que está diseñada para funcionar con bitstreams de configuración de FPGAs de la familia Virtex-5. Sin embargo, esta aplicación también puede leer bitstreams de las familias anteriores de Virtex ya que los comandos incluidos en el fichero de configuración se van heredando y son los mismos, con el añadido de

comandos nuevos. El único problema sería que habría que añadir el IDCODE de la FPGA que se quiere utilizar para que la renozca. Como el objetivo de la herramienta era desglosar bitstreams de configuración sobre Virtex-5, solamente reconoce los IDCODE de los números de serie de las FPGAs de la familia Virtex-5, sin embargo introducir los números de serie de otras Virtex para que reconozca el IDCODE es extremadamente fácil.

Ejecución: Esta aplicación se ejecuta por consola de una forma sencilla. Una vez abierta la consola del sistema operativo y situarse en el directorio donde se encuentra la aplicación, el comando para ejecutarla sería:

java -jar Virtex_V_Configuration_Viewer <ruta/nombre_del_fichero>.bit

Dependiendo de la configuración de la máquina virtual de JAVA y del tamaño del fichero (.bit) que se quiera interpretar, puede que la memoria de la máquina virtual de JAVA no sea suficiente para el volumen de datos que recoge un .bit. En este caso sería necesario un par de directivas para que este tamaño aumentase y no diese problemas. Finalmente, el comando para ejecutar la aplicación sería:

java -jar -Xms512m -Xmx512m Virtex_V_Configuration_Viewer <ruta/nombre_del_fichero>.bit

Resultado: Como resultado de la ejecución de esta aplicación, se crea un fichero de texto que se puede abrir con cualquier editor de texto, nombrándolo con el nombre del archivo de configuración que ha sido introducido como parámetro y con la extensión (.v5cv), que son las siglas de la aplicación. En las figuras 4.6 y 4.7, se puede observar el aspecto final de la ejecución de la aplicación con un bitstream de configuración. En el fichero aparecen los comandos en hexadecimal y en binario, para tener una mejor visión tanto de los bytes que forman el comando como de los bits del comando, y una pequeña explicación de la funcionalidad de cada uno de ellos.

```
FFFFFFFF - 11111111111111111111111111111111
Dummy Word
FFFFFFFF - 11111111111111111111111111111111
Dummy Word
000000BB - 00000000000000000000000010111011
Bus Width Sync Word
11220044 - 00010001001000010000000001000100
Bus Width Detect
FFFFFFFF - 11111111111111111111111111111111
Dummy Word
FFFFFFFF - 11111111111111111111111111111111
Dummy Word
AA995566 - 10101010100110010101010101100110
Sync Word
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 1
30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
00000007 - 00000000000000000000000000000111
Packet Data: Word 1 CMD Register = RCRC Command
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 1
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 2
30018001 - 00110000000000011000000000000001
Type 1 Write 1 Words to IDCODE
02AD6093 - 00000010101011010110000010010011
Packet Data: Word 1 Device ID = 0x02AD6093 (XC5VLX110T)
30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
00000001 - 00000000000000000000000000000001
Packet Data: Word 1 CMD Register = WCFG Command
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 1
30002001 - 00110000000000000010000000000001
Type 1 Write 1 Words to FAR
00018F00 - 00000000000000011000111100000000
Packet Data: Word 1 Frame Address Register = 0x00018F00
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 1
300045ED - 00110000000000001000010111101101
Type 1 Write 1517 Words to FDRI
00200000 - 00000000001000000000000000000000
Packet Data: Word 1 Frame Data Input Register = 0x00200000
00100000 - 00000000000100000000000000000000
Packet Data: Word 2 Frame Data Input Register = 0x00100000
```

Figura 4.6: *Ejemplo del resultado de la ejecución de la aplicación Virtex V Configuration Viewer*

```
3071  00000000 - 00000000000000000000000000000000
3072  Packet Data: Word 1515 Frame Data Input Register = 0x00000000
3073  00000000 - 00000000000000000000000000000000
3074  Packet Data: Word 1516 Frame Data Input Register = 0x00000000
3075  00000000 - 00000000000000000000000000000000
3076  Packet Data: Word 1517 Frame Data Input Register = 0x00000000
3077  3000C001 - 00110000000000001100000000000001
3078  Type 1 Write 1 Words to MASK
3079  00001000 - 00000000000000000001000000000000
3080  Packet Data: Word 1 Mask Register = 0x00001000
3081  30030001 - 00110000000000110000000000000001
3082  Type 1 Write 1 Words to CTL1
3083  00000000 - 00000000000000000000000000000000
3084  Packet Data: Word 1 Control Register 1 = 0x00000000
3085  30008001 - 00110000000000001000000000000001
3086  Type 1 Write 1 Words to CMD
3087  00000003 - 00000000000000000000000000000011
3088  Packet Data: Word 1 CMD Register = DGHIGH/LFRM Command
3089  20000000 - 00100000000000000000000000000000
3090  Type 1 NOOP Word 1
3091  20000000 - 00100000000000000000000000000000
3092  Type 1 NOOP Word 2
3093  20000000 - 00100000000000000000000000000000
3094  Type 1 NOOP Word 3
3095  20000000 - 00100000000000000000000000000000
3096  Type 1 NOOP Word 4
3097  ........
3098  20000000 - 00100000000000000000000000000000
3099  Type 1 NOOP Word 100
3100  20000000 - 00100000000000000000000000000000
3101  Type 1 NOOP Word 101
3102  30002001 - 00110000000000000010000000000001
3103  Type 1 Write 1 Words to FAR
3104  00EF8000 - 00000000111011111000000000000000
3105  Packet Data: Word 1 Frame Address Register = 0x00EF8000
3106  30000001 - 00110000000000000000000000000001
3107  Type 1 Write 1 Words to CRC
3108  3B963C53 - 00111011100101100011110001010011
3109  Packet Data: Word 1 CRC Register = 0x3B963C53
3110  30008001 - 00110000000000001000000000000001
3111  Type 1 Write 1 Words to CMD
3112  0000000D - 00000000000000000000000000001101
3113  Packet Data: Word 1 CMD Register = DESYNCH Command
3114  20000000 - 00100000000000000000000000000000
3115  Type 1 NOOP Word 1
```

Figura 4.7: *Cont. Ejemplo del resultado de la ejecución de la aplicación Virtex V Configuration Viewer*

Flujo del programa: Antes de explicar el flujo del programa, vamos a detallar cómo es la composición de un bitstream de configuración.

Primeramente, el archivo bitstream de configuración (.bit) es un archivo de compuesto por palabras de 32 bits o 4 bytes. Cada uno de estos 4 bytes forman un comando o palabra de configuración. Los comandos (*Dummy Word*, *Sync Word*, *NOOP Word*) únicamente están formados por uno de estos comandos de 4 bytes, mientras que los otros comandos se dividen en dos o más comandos seguidos. En el primero de ellos, indica qué tipo de paquete es, ya sea de tipo 1 o 2, si es un comando de escritura o de lectura, en qué registro van a almacenarse los datos siguientes y cuál es la cantidad de datos que se enviarán seguidamente (ApéndiceA). Según el número de datos que se haya indicado en este comando, seguidamente vendran dicho número de palabras con los valores que se escribirán en el registro indicado.

Ejemplo:

```
13  AA995566 - 10101010100110010101010101100110
14  Sync Word
```

Figura 4.8: *Ejemplo del comando de sincronización*

Este comando (AA995566) sirve como palabra de sincronización, para empezar la configuración. Se puede observar que es una única palabra, que funciona solo pues no necesita escribir en ningún registro ningún valor.

```
35  30002001 - 00110000000000000010000000000001
36  Type 1 Write 1 Words to FAR
37  00018F00 - 00000000000000011000111100000000
38  Packet Data: Word 1 Frame Address Register = 0x00018F00
```

Figura 4.9: *Ejemplo del comando de escritura en el registro FAR*

En este comando (30002001), se indica que es un comando de tipo 1, de escritura y que se escribirá en el registro FAR un solo dato. Mientras que la siguiente palabra (00018F00) es el valor que se almacenará en el registro FAR.

El flujo de programa, consta de 3 sencillos pasos:

En el primer paso, se empienzan a leer todos los bytes del bitstream. Hay que tener en cuenta que los primeros bytes contienen la información del nombre del archivo, fecha de creación, etc. Toda esta información sólo se lee hasta que se alinea la lectura de bytes con la lectura de la palabra de sincronización, dado que para ésto sirve este comando. Entonces se continúan leyendo todos los demás bytes directamente en grupos de 4 bytes, que forman cada palabra, como se ha explicado anteriormente.

En el segundo paso, se traducen dichos comandos indicando si es un comando de escritura, de lectura, para qué registro va dirigido, etc., utilizando una serie de máscaras y constantes de datos que determinan la funcionalidad de cada comando.

Por último se genera el nuevo archivo con toda la información conseguida por la traducción de los distintos bytes como se mostró en las figuras 4.6 y 4.7.

Como vesión posterior a esta aplicación, fue necesaria la implementación de otra aplicación, Virtex V Configuration Cleaner. Esta aplicación fue diseñada por la necesidad de tener un bitstream parcial limpio, es decir, un bitstream sin su cabecera, que contiene información sobre su fecha de creación, nombre, etc. Dado que el bitstream parcial se necesita enviar a la FPGA y almacenarlo para luego tratarlo e insertarle bitflips, esta información puede ser además de innecesaria, molesta, ya que no es un número determinado de bytes, sino que es variable y pueden quedar así los comando desalineados, provocando errores. Por eso se decidió realizar esta aplicación que simplemente elimina la información del archivo y deja el bitstream parcial (.bit) limpio, únicamente con comandos de configuración. Su ejecución es similar que la de la aplicación Virtex V Configuration Viewer, únicamente hay que cambiar el nombre del ejecutable. Como resultado de la ejecución de esta aplicación, se creará un nuevo archivo bitstream de configuración (.bit), únicamente con comandos de configuración, un bitstream limpio.

Inyección de bitflips

El proceso que realiza la inyección de los bitflips, se encarga de leer los comandos del bitstream parcial almacenado en la memoria DDR2 RAM para luego irlos escribiendo uno

a uno en otra región de la memoria DDR2 RAM. En esta copia del bitstream parcial es donde se insertará el bitflip y además será el bitstream parcial que se envíe al ICAP para realizar la reconfiguración. Por ello, en la construcción del bitstream parcial hay que tener en cuenta que el código CRC al insertar el bitflip ya no es el mismo, así que si se introdujese el comando de comprobación del CRC, al realizar la reconfiguración éste se comprobaría y al no ser coherente, la reconfiguración fallaría. Para solucionar esto, en el "bitstream parcial con bitflip" hay que suprimir el comando de escritura del código CRC en su registro, además de desactivar la comprobación del CRC poniendo a '1' el bit 28 del registro de opciones de configuración COR0.

Este proceso se invoca cada vez que se quiere insertar un bitflip, y para reducir el tiempo de ejecución, en vez de leer todos los mapas de configuración del bitstream cada vez que se invoca, utiliza la información proporcionada por el proceso de búsqueda de los mapas de configuración. Así, dependiendo de la posición del bit donde se quiere insertar el bitflip, se obtiene el mapa de configuración donde se encuentra ese bit, (pudiendo así ignorar la lectura de los demás mapas de configuración), simplemente saltando tantas posiciones en la memoria como tamaño tenga el mapa de configuración. Una vez hecho esto, dicho mapa de configuración no se lee completamente, sino que dependiendo del bit a modificar se calcula la frame donde se encontraría dentro de este mapa de configuración y se salta hasta la posición que ocupa la primera palabra de dicha frame. Entonces se calcula la dirección de memoria donde se encontraría esta frame en la memoria de configuración de la FPGA y con ello se actualiza el valor en el comando de escritura en el registro FAR. Únicamente se lee una frame y se inserta el bitflip en el bit deseado, y sólo se copia esta frame para la construcción del mapa de configuración del "bitstream parcial con bitflip". Con esta metodología se consigue que, independientemente del número de mapas de configuración o del tamaño del bitstream parcial, el tiempo de ejecución sea siempre el mismo, ya que sólo se lee una frame, todas las demás frames de configuración se saltan, pues no son necesarias para la inserción del bitflip.

Capítulo 5

Resultados Experimentales

Para evaluar la eficiencia de la nueva versión de NESSY como palataforma de inyección de bitflips, se han usado dos aplicaciones reales: un contador de 8 bits y un filtro FFE. Es importante destacar que en ambos experimentos ha sido necesario tomar la siguiente consideración adicional: *Xilinx PlanAhead no puede asegurar que los cables que comunican el sistema hardware estático no crucen la región parcialmente reconfigurable*. En otras palabras, es posible que algunos bits en la memoria de configuración que configura la región reservada para el circuito bajo prueba configuren ciertos cables que comunican el sistema hardware estático. Si se inyecta un bitflip en dichos bits, el sistema podría dejar de funcionar. NESSY es capaz de manejar estas situaciones: si el tiempo transcurrido entre la inyección del bitflip y la recepción de los resultados del testbench excede un *timeout*, el sistema completo se reconfigura de nuevo y la inyección de bitflips se reanuda desde el bitflip siguiente al que causó la caída del sistema. Sin embargo, esta opción es muy costosa en tiempo. Por ello, para acelerar el proceso de inyección de bitflips y obtener así resultados realistas, en ambos experimentos se ha seleccionado manualmente una región libre de cables que comuniquen el sistema hardware estático para situar el circuito bajo test, haciendo uso de la herramienta Xilinx FPGA Editor.

En ambos experimentos, se ha decidido situar el circuito bajo prueba en la esquina superior derecha de la FPGA, mientras que el sistema hardware estático se sitúa en la parte inferior del dispositivo.

Para medir los tiempos que posteriormente en cada experimento se indican, ha sido utilizado el módulo *plb_ Timer*. Este módulo se conecta al bus PLB del sistema. Es simplemente un contador, mediante el cual, utilizando las primitivas (*XTmrCtr_ Reset()*, *XTmrCtr_ Start()*, *XTmrCtr_ Stop()*, *XTmrCtr_ GetValue()*) se puede contar el número de ciclos de reloj que transcurre entre la primitiva para empezar a contar (*XTmrCtr_ Start()*) y la primitiva para parar de contar (*XTmrCtr_ Stop()*). Contando el número exacto de ciclos que tarda el proceso que queremos medir, y teniendo en cuenta que el reloj trabaja a 100 MHz es fácil hacer cálculos y obtener de una manera bastante precisa el retardo.

5.1. Contador de 8 Bits

El contador de 8 bits es una aplicación pequeña que se utilizó en primer lugar para realizar test rápidos. Este circuito sólo necesita 2 slices (o 1 CLB, que contiene 2 slices). Sin embargo, como se ha comentado anterioremente, la unidad de configuración mínima en una Virtex-5 es una frame, la cual engloba la configuración de la 36ª parte de una columna entera de 20 CLBs. Por tanto, el bitstream correspondiente al circuito parcial realmente ocupa una columna completa de CLBs, que es la región parcialmente reconfigurable mas pequeña que PlanAhead puede generar.

En este experimento, NESSY realiza un test exhaustivo de los 46080 bits de configuración de la columna de CLBs del circuito en 3 minutos y 15 segundos. Cada test completo de bitflip tarda una media de 4.23 ms. Esto comprende generar e inyectar un único bitflip, ejecutar un testbench de 200 entradas en el circuito modificado y restaurar al circuito original si es necesario. Sin embargo, sólo la creación e inyección de un bitflip es mucho más rápida: 0.63 ms. Esto es dos órdenes de magnitud más rápido que otras soluciones aportadas en la literatura [6], [10], que consiguieron 620 y 53 ms por inyección de un único bitflip, respectivamente. De los 46080 bitflips inyectados, sólo 617 causan una ejecución errónea del circuito testeado, lo cual es un 1.3 % del total.

La figura 5.1 muestra el número de bitflips que produjeron error en el circuito, por cada frame existente en la columna de CLBs usada (una columna de 20 CLBs contiene 36 frames). Las frames 4-5, 30-31 y 35 son más propensas a errores que las demás. Sin embargo, los bitflips no afectan cuando caen en las frames 26-29. Este es un resultado interesante, ya que indica que esta parte del circuito no tiene por qué ser protegida contra radiaciones.

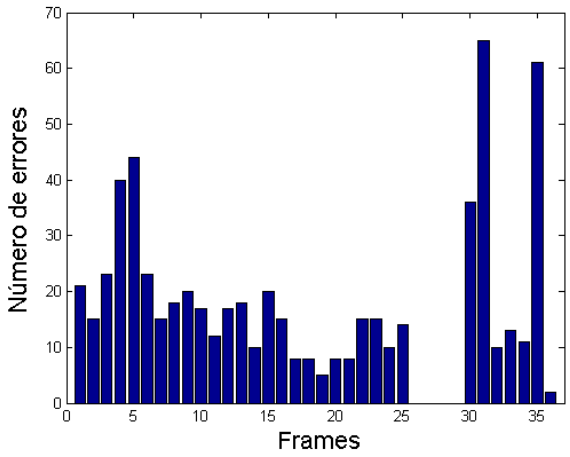

Figura 5.1: *Número de bitflips por frame que causaron errores en el contador 8 bits*

5.2. Filtro FFE

Un filtro FFE es un tipo específico de filtro de respuesta finita de impulsos. Cuando se sintetiza e implementa este circuito en la FPGA, EDK infiere alrededor de 200 slices y 32 DSPs empotrados existentes en la Virtex-5. Sin embargo, se puede restringir el uso de DSPs en el código vhdl del filtro FFE, consiguiendo así sintetizar este circuito sin usar DSPs. Añadiendo en el fichero vhdl del *Adaptador circuito*, generado automáticamente por NESSY, los atributos que se indican con un rectángulo en la Figura 5.2, se consigue que el componente FFE (circuito a testear), se implemente sin el uso de DSPs. Incluyendo esta opción, los recursos necesarios para su implementación son 1680 slices, lo cual es un 9.72 % de los slices existentes en la FPGA. Se ha utilizado esta opción para evaluar el impacto de la inyección de bitflps en un circuito de tamaño mayor.

En este caso, Nessy realiza la inyección exhaustiva de bitflips en 5 horas, 21 minutos y 9 segundos usando un testbench con 200 valores diferentes para sus entradas. Esto da como resultado un testeo de bitflip de 9.76 ms de media. El motivo del incremento de este tiempo con respecto al contador de 8 bits (4.23 ms) es debido al retardo adicional introducido al restaurar el bitstream parcial de la FFE, el cual es considerablemente mayor (1680 slices) que el del contador de 8 bits (sólo 2 slices). Sin embargo, el tiempo de inyección de bitflips

```
architecture Behavioral of circuito is

    attribute use_dsp48: string;

    ------------------ SEÑALES DEL CIRCUITO ----------------------
    signal SENAL_ERROR_OUT          : std_logic_vector(0 to 21);
    signal SENAL_DATAOUT          : std_logic;
    --------------------------------------------------------------


    ---------------------- COMPONENTE ----------------------------
    COMPONENT FFE PORT (
        signal RX : in std_logic_vector(0 to 7);
        signal CLK : in std_logic;
        signal RESET : in std_logic;
        signal ERROR_OUT : out std_logic_vector(0 to 21);
        signal DATAOUT : out std_logic
        );
    END COMPONENT;
    --------------------------------------------------------------

    attribute use_dsp48 of FFE : component is "no";
```

Figura 5.2: *Código vhdl del circuito del filtro FFE sin DSPs*

es exactamente el mismo que en el anterior experimento (0.63 ms). Esto se debe a que este tiempo no depende del tamaño del circuito testeado, ya que la inyección de un bitflip involucra sólamente la configuración de una frame de la FPGA.

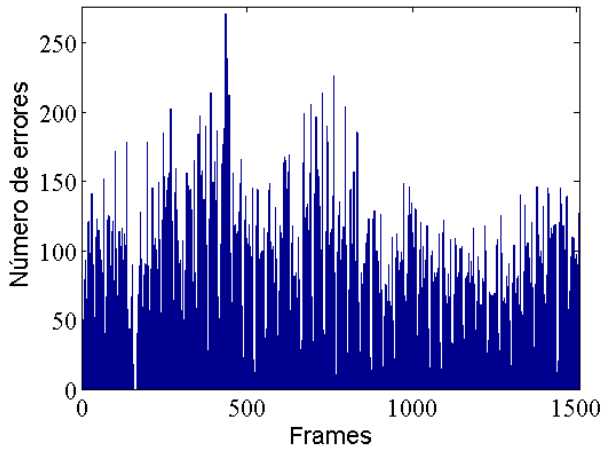

Figura 5.3: *Número de bitflips por frame que causaron errores en el filtro FFE*

En este experimento se realizaron un total de 1971200 inyecciones, de las cuales 113771 produjeron error (5.8 % del total). La figura 5.3 muestra cómo se distribuyen los errores entre las 1512 frames usadas para la implementación de este circuito. Tal y como muestra la figura, las frames 157-167 no se vieron afectadas por los bitflips.

Capítulo 6

Conclusiones

En este proyecto se ha presentado la metodología diseñada para la técnica de inyección de bitflips en mapas de configuración que se ha desarrollado para emular la alteración que una partícula cósmica produciría sobre la memoria de configuración de un diseño implementado sobre una FPGA. Dicha técnica ha sido introducida en la herramienta NESSY, dando lugar a una nueva plataforma de inyección de errores basada en hardware, sobre una FPGA tipo Virtex-5. Esta nueva versión de NESSY presenta numerosas ventajas con respecto a otras que se pueden encontrar en la literatura. En primer lugar, no es intrusiva, ya que el usuario puede decidir exactamente dónde colocar el circuito a testear. En segundo lugar, resulta menos costosa que otras plataformas, al necesitar un PC y sólo una FPGA. Por último, permite acelerar la inyección de errores, ya que la emulación de un SEU se ha reducido en al menos 2 órdenes de magnitud respecto a otras palataformas existentes.

Capítulo 7

Trabajo Futuro

Dado que este ha sido el primer paso de una nueva plataforma de inyección de errores en una FPGA, todavía se pueden realizar mejoras y evoluciones en la plataforma, para añadir funcionalidades nuevas y conseguir una plataforma aún más completa. Más detalladamente, en el futuro se quieren incluir varias herramientas automáticas que permitan mejorar la fiabilidad de un determinado diseño, teniendo en cuenta los resultados obtenidos por NESSY sobre la vulnerabilidad de las distintas partes del mismo durante la inyección de errores. Además, se quiere incluir la posibilidad de emular fallos en los que el impacto de una partícula cósmica afecta a más de una celda de memoria, es decir, multi-bitflips, efecto cada vez más cercano con las nuevas tecnologías.

Capítulo 8

Publicaciones

Como consecuencia del esfuerzo realizado, se ha conseguido una plataforma de inyección de errores con unos buenos resultados y unas mejoras importantes respecto a otras plataformas existentes. Como se ha mencionado anteriormente, la plataforma NESSY posee un coste de implementación en términos de área mucho menor que otros sistemas. No es una plataforma intrusiva. Mejora el tiempo en inyectar un solo bitflip y la evaluación del error producido es hasta 2 ordenes de magnitud menor. Y teniendo en cuenta las plataformas existentes, es la primera plataforma de inyección de errores que ha sido implementada en una FPGA Virtex-5.

Gracias a estas mejoras respecto de los otros sistemas existentes, se ha llevado a cabo la escritura de 2 artículos para dos congresos, uno de ellos en español y el otro en inglés, explicando la plataforma de inyección de errores NESSY, y los resultados obtenidos, así como sus ventajas frente a las otras plataformas de inyección de errores.

La Sociedad de Arquitectura y Tecnología de Computadores (SARTECO) organiza sus Jornadas Sarteco del 19 al 21 de septiembre de 2012 (JS2012) en la ciudad de Elche (España). En estas jornadas, se lleva a cabo la XII Jornadas de Computación Reconfigurable y Aplicaciones (JCRA) en la cual ha sido aceptado el artículo titulado *"NESSY: Una implementación de una plataforma de inyección de errores de bajo coste en una FPGA Virtex-5"*.

Para la 25 edición del 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI & Nanotechnology Systems, DFT 2012, que tendrá lugar del 3 al 5 de Octubre en

Austin, Texas, U.S.A., se ha enviado un artículo, titulado *"NESSY: An Implementation of a Low-cost Fault-injection Platform on a Virtex-5 FPGA"*. Dicho artículo está en trámites de revisión hasta el 1 de julio, cuando se sabrá si ha sido aceptado.

Bibliografía

[1] Anderson, K., *Low-Cost, Radiation-Tolerant, On-Board Processing Solution*, IEEE Aerospace Conference, pp. 1-8, 2005.

[2] Ziegler, J. F. and Lanford, W. A. *The Effect of Sea-Level Cosmic Rays on Electronic Devices*, Journal of Applied Physics, Vol. 52, No. 6, pp. 4305-4312, 1981.

[3] Dutton, B. F. and Stroud, C. E., *Built-In Self-Test of Embedded SEU Detection Cores in Virtex-4 and Virtex-5 FPGAs*, International Conference on Embedded Systems & Applications (ESA), pp. 149-155, 2009.

[4] Ichinomiya, Y., Tanoue, S., Amagasaki, M., Iida, M., Kuga, M. and Sueyoshi, T., *Improving the Robustness of a Softcore Processor against SEUs by Using TMR and Partial Reconfiguration*, IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 47-54, 2010.

[5] Stott, E. A., Sedcole, N. P. and Cheung, P. Y. K. *Fault tolerance and reliability in field-programmable gate arrays*, IET Computers & Digital Techniques, Vol. 4, No. 3, pp. 196-210, 2010.

[6] Battezzati, N., Sterpone, L. and Violante, M., *A new low-cost non intrusive platform for injecting soft errors in SRAM-based FPGAs*, Industrial Electronics, IEEE International Symposium on Industrial Electronics (ISIE), pp. 2282-2287, 2008.

[7] Alderighi, M. et al., *Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform*, IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT), pp. 105-113, 2007.

[8] Sterpone, L., Aguirre, M., Tombs, J. and Guzman-Miranda, H., *On the design of tunable fault tolerant circuits on SRAM-based FPGAs for safety critical applications*, Design, Automation and Test in Europe (DATE), pp. 336-341, 2008.

[9] Alderighi, M. et al., *Experimental Validation of Fault Injection Analyses by the FLIPPER Tool*, IEEE Transactions on Nuclear Science, Vol. 57, No. 4, pp. 2129-2134, 2010.

[10] Monson, J. S., Wirthlin, M. and Hutchings, B., *Fault Injection Results of Linux Operating on an FPGA Embedded Platform*, Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 37-42, 2010.

[11] Virtex-5 FPGA User Guide UG190 (v5.4) March 16, 2012, Xilinx.

[12] Virtex-5 FPGA Configuration User Guide UG191 (v3.10) November 18, 2011, Xilinx.

[13] Difference-Based Partial Reconfiguration, Application Note: Virtex Architectures, XAPP290, Emi Eto, Xilinx.

[14] Parris, Matthew G., Sharma, Carthik A. and Demara, Ronald F., *Progress in autonomous fault recovery of field programmable gate arrays*, ACM Computing Surveys, Vol. 45, No. 4, pp. 31:1-31:30, 2011.

[15] Ilias, A., Papadimitriou, K. and Dollas, A., *Combining Duplication, Partial Reconfiguration and Software for On-line Error Diagnosis and Recovery in SRAM-Based FPGAs*, IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 73-76, 2010.

[16] Shih-Fu Liu et al., *Increasing Reliability of FPGA-Based Adaptive Equalizers in the Presence of Single Event Upsets*, IEEE Transactions on Nuclear Science, Vol. 58, No. 3, pp. 1072-1077, 2011.

[17] Jones, L., *Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices*, Xilinx Application Note, XAPP714 (v 1.5), Xilinx Inc., 2007.

[18] Chapman, K. and Jones, L., *SEU Strategies for Virtex-5 Devices*, Xilinx Application Note, XAPP864 (v1.0.1), Xilinx Inc., 2010.

Apéndice A

Detalles de configuración de FPGAs Virtex-5

A.1. Visión general del bitstream de configuración

El bitstream de las Virtex-5 contiene tanto comandos de configuración de la lógica de la FPGA, como datos de configuración. El tamaño del bitstream completo depende del tipo de dispositivo Virtex-5 utilizado. En este proyecto se utilizará la Virtex-5 XC5VLX110T con un total de 31.118.848 bits de configuración.

Un bitstream de la Virtex-5 está compuesto por 3 secciones:

1. Autodetección del ancho de bus (*Bus Width Auto Detection*)
2. Sincronización (*Sync Word*)
3. Configuración de la FPGA

A.1.1. Autodetección del ancho de bus (*Bus Width Auto Detection*)

El bus de configuración puede ser de 8, 16 ó 32 bits. El sistema es capaz de autodetectarlo. El proceso de auto detección del ancho de bus (*Bus Width Auto Detection*) está insertado al principio de todo bitstream. Aparece antes que la palabra de sincronización (*Sync Word* A.1.2), y en los modos de configuración serie se ignora. Para detectar el ancho del bus sólo se tienen en cuenta los últimos 8 bits del bus paralelo (D[0..7] en la Figura A.1),

y dependiendo de la secuencia de bytes recibida, la lógica de configuración puede automáticamente detectar el ancho de bus. Se envía a través de D[0..7] el byte 0xBB y seguidamente, la palabra 0x11220044 por el bus completo (Figura A.1).

D[24:31]	D[16:23]	D[8:15]	D[0:7]	Comments
0xFF	0xFF	0xFF	0xFF	
0x00	0x00	0x00	0xBB	Bus Width Pattern
0x11	0x22	0x00	0x44	Bus Width Pattern
0xFF	0xFF	0xFF	0xFF	
0xFF	0xFF	0xFF	0xFF	
0xAA	0x99	0x55	0x66	Sync Word
...	...	...	...	...

Figura A.1: *Proceso de detección del ancho de bus de configuración*

A continuación se comprueba qué se ha recibido por D[0..7] y dependiendo del tamaño del bus de configuración las posibles opciones son:

- 0x11: El ancho de bus detectado es de 8 bits.
- 0x22: El ancho de bus detectado es de 16 bits.
- 0x44: El ancho de bus detectado es de 32 bits.

Si el byte econtrado después del byte 0xBB no es 0x11, 0x22 o 0x44, la máquina de estados del ancho de bus se resetea en busca de una próxima secuencia de bytes válida para la detección del ancho de bus de configuración.

A.1.2. Sincronización (*Sync Word*)

Para alinear la lógica de configuración con los paquetes de datos de 32 bits del bitstream, se utiliza una palabra especial de sincronización (*Sync Word*). Hasta la palabra de sincronización no se procesa ningún comando de configuración; sin embargo, el tamaño del bus de configuración debe ser detectado correctamente para modos de configuración paralela, antes de que la palabra de sincronización pueda ser detectada. La palabra de sincronización tiene este formato (Figura A.2):

31:24	23:16	15:8	7:0
0xAA	0x99	0x55	0x66

Figura A.2: *Palabra de sincronización (Sync Word)*

A.1.3. Configuración de la FPGA

La memoria de configuración de una FPGA Virtex-5 está organizada en múltiples frames (unidad mínima de configuración) que se distribuyen en forma de cuadrícula en el dispositivo. Estas frames son el segmento mínimo direccionable en la memoria de configuración de la Virtex-5, por lo que todas las operaciones deben de actuar sobre frames de configuración completas. El tamaño de una frame es de 41 palabras, y se corresponde con una altura de 20 CLBs y un ancho de 1 bit (Figura A.3).

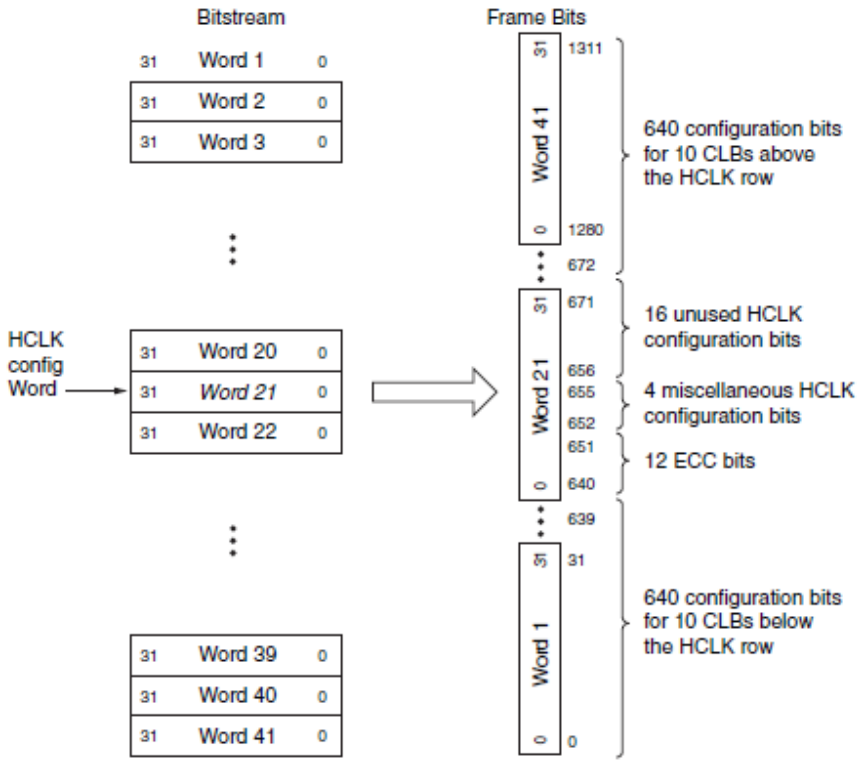

Figura A.3: *Palabras de configuración en un bitstream y bits de configuración en una frame*

Comandos de configuración

Todos los comandos del bitstream de configuración de la Virtex-5 son lecturas o escrituras en los registros de configuración.

Tipos de paquetes El bitstream de configuración de la FPGA consiste en 2 tipos de comandos: Tipo 1 y Tipo 2.

Paquete Tipo 1: Los paquetes de Tipo 1 son de lectura y escritura de registros. Se necesitan 5 bits para direccionar los 20 registros de configuración disponibles en las FPGAs tipo Virtex-5. Están formados por una cabecera más datos. La sección de la cabecera del paquete de Tipo 1 es siempre una palabra de 32 bits (Figura A.4) donde se especifica el código de operación (*Opcode*), el registro sobre el que se va a realizar, y el número de palabras que se van a leer o escribir. Seguido de la cabecera de Tipo 1 hay una sección de

datos de Tipo 1, formada por número de palabras de 32 bits igual al especificado en la parte *Word Count* de la cabecera, formando así el paquete completo de Tipo 1.

Header Type	Opcode	Register Address	Reserved	Word Count
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxx

Notes:
1. "R" means the bit is not used and reserved for future use.

Figura A.4: *Cabecera del paquete Tipo 1*

La sección *Opcode* de la cabecera del paquete indica si el paquete es de lectura, escritura o ninguna de las dos, según se muestra en la Figura A.5.

Opcode	Function
00	NOP
01	Read
10	Write
11	Reserved

Figura A.5: *Formato del código de operación (Opcode)*

Paquete Tipo 2: El paquete de Tipo 2, sigue siempre a un paquete de Tipo 1, cuando se van a escribir grandes bloques, y la sección *Word Count* del paquete de Tipo 1 se queda pequeña (11 bits) para especificar el número de palabras. En ese caso, en el paquete de Tipo 1 *Word Count* se pone a 0 y se manda a continuación un paquete de Tipo 2 que sólo contiene el número de palabras con 27 bits según se muestra en la Figura A.6. En el paquete Tipo 2 no es necesario el direccionamiento del registro de configuración, pues debe de estar indicado en el paquete de Tipo 1 previo. La cabecera de este paquete es siempre una palabra de 32 bits (Figura A.6). Seguido de la cabecera de Tipo 2 hay una sección de datos de Tipo 2, la cual contiene el número de palabras de 32 bits especificado en la parte *Word Count* de la cabecera.

Header Type	Opcode	Word Count
[31:29]	[28:27]	[26:0]
010	RR	xxxxxxxxxxxxxxxxxxxxxxxxxxx

Figura A.6: *Cabecera del paquete Tipo 2*

Registros de configuración Los registros de configuración se seleccionan mediante la cabecera del paquete de Tipo 1 y pueden ser los que se muestran en las tablas de las Figuras A.7 y A.8:

Reg. Name	Read/Write	Address	Description
CRC	Read/Write	00000	CRC Register
FAR	Read/Write	00001	Frame Address Register
FDRI	Write	00010	Frame Data Register, Input Register (write configuration data)
FDRO	Read	00011	Frame Data Register, Output Register (read configuration data)
CMD	Read/Write	00100	Command Register
CTL0	Read/Write	00101	Control Register 0
MASK	Read/Write	00110	Masking Register for CTL0 and CTL1
STAT	Read	00111	Status Register
LOUT	Write	01000	Legacy Output Register (DOUT for daisy chain)
COR0	Read/Write	01001	Configuration Option Register 0
MFWR	Write	01010	Multiple Frame Write Register

Figura A.7: *Registros de configuración existentes en las FPGAs tipo Virtex-5 (1)*

Reg. Name	Read/Write	Address	Description
CBC	Write	01011	Initial CBC Value Register
IDCODE	Read/Write	01100	Device ID Register
AXSS	Read/Write	01101	User Bitstream Access Register
COR1	Read/Write	01110	Configuration Option Register 1
CSOB	Write	01111	Used for daisy chain parallel interface, similar to LOUT
WBSTAR	Read/Write	10000	Warm Boot Start Address Register
TIMER	Read/Write	10001	Watchdog Timer Register
BOOTSTS	Read	10110	Boot History Status Register
CTL1	Read/Write	11000	Control Register 1

Figura A.8: *Registros de configuración existentes en las FPGAs tipo Virtex-5 (2)*

A continuación se van a explicar aquellos registros que se necesitan tener en cuenta en la creación de un bitstream de reconfiguración parcial (más información sobre la configuración de la FPGA tipo Virtex-5 y la composición del bitstream de configuración en [13]).

Registro de direccionamiento de frames FAR: Sirve para direccionar las frames de configuración que primero se escriben en el registro FDRI, y a continuación el circuito de configuración las vuelca en la memoria de configuración de la FPGA, en el valor proporcionado por el registro FAR (Figura A.9).

Address Type	Bit Index	Description
Block Type	[23:21]	Block types are: Interconnect and Block Configuration (000), Block RAM Content (001), Interconnect and Block Special Frames (010 - typically not used by users), and Block RAM Non-Configuration Frames (011 - not used by users).
Top_B Bit	20	Select between top-half rows (0) and bottom-half rows (1).
Row Address	[19:15]	Selects the current row. The row addresses increase from bottom to top.
Column Address	[14:7]	Selects a major column, such as a column of CLBs. Column addresses start at 0 on the left and increase to the right.
Minor Address	[6:0]	Selects a frame within a major column.

Figura A.9: *Descripción del registro de direccionamiento de frames FAR*

Registro CRC: En él se escribe el código de redundancia cíclica para compararlo con los datos del bitstream.

Registro FDRI: En este registro se escriben los datos de las frames de configuración del circuito. Cada frame se escribirá a continuación en la dirección de frame especificada por el registro FAR, y ésta se incrementa automáticamente en uno.

Registro de comandos CMD: Existen comandos de control para la configuración de las frames. Estos comandos se indican mediante el registro CMD. Hay un gran número de comandos distintos y se identifican por su código de comando (Figuras A.10 y A.11).

Command	Code	Description
NULL	00000	Null command.
WCFG	00001	Writes Configuration Data: Used prior to writing configuration data to the FDRI.
MFW	00010	Multiple Frame Write: Used to perform a write of a single frame data to multiple frame addresses.
DGHIGH/ LFRM	00011	Last Frame: Deasserts the GHIGH_B signal, activating all interconnects. The GHIGH_B signal is asserted with the AGHIGH command.
RCFG	00100	Reads Configuration Data: Used prior to reading configuration data from the FDRO.
START	00101	Begins the Startup Sequence: Initiates the startup sequence. The startup sequence begins after a successful CRC check and a DESYNC command are performed.
RCAP	00110	Resets the CAPTURE signal after performing readback-capture in single-shot mode (see "Readback Capture," page 151).
RCRC	00111	Resets CRC: Resets the CRC register.
AGHIGH	01000	Asserts the GHIGH_B signal: Places all interconnect in a High-Z state to prevent contention when writing new configuration data. This command is only used in shutdown reconfiguration. Interconnect is reactivated with the LFRM command.
SWITCH	01001	Switches the CCLK frequency: Updates the frequency of the Master CCLK to the value specified by the OFSEL bits in the COR0 register.

Figura A.10: *Códigos de los registros de comandos CMD (1)*

Command	Code	Description
GRESTORE	01010	Pulses the GRESTORE signal: sets/resets (depending on user configuration) IOB and CLB flip-flops.
SHUTDOWN	01011	Begin Shutdown Sequence: Initiates the shutdown sequence, disabling the device when finished. Shutdown activates on the next successful CRC check or RCRC instruction (typically an RCRC instruction).
GCAPTURE	01100	Pulses GCAPTURE: Loads the capture cells with the current register states (see "Readback Capture," page 151).
DESYNCH	01101	Resets the DALIGN signal: Used at the end of configuration to desynchronize the device. After desynchronization, all values on the configuration data pins are ignored.
CRCC	10000	When Readback CRC is selected, the CRC is calculated after full configuration and reconfiguration. Toggling GHIGH also calculates the CRC. This command can be used when GHIGH is not toggled during reconfiguration (active partial reconfiguration).
IPROG	01111	Internal PROG for triggering a warm boot.
LTIMER	10001	Reload watchdog timer.

Figura A.11: *Códigos de los registros de comandos CMD (2)*

Registro FDRO: Este registro de sólo lectura sirve para hacer readback de los datos de las frames de configuración empezando por la dirección especificada por el registro FAR.

Registro IDCODE: Para comparar compatibilidad, el dato IDCODE proporcionado por el bitstream debe ser el mismo que el IDCODE del dispositivo.

Registro de control CTL0: El registro de control CTL0 es usado para configurar el dispositivo de la Virtex-5 (Figura A.12).

Registro de control CTL1: El registro de control CTL1 es usado para configurar el dispositivo de la Virtex-5. Este registro es reservado.

Name	Bit Index	Description
ICAP_SELECT	30	ICAP Port Select. 0: Top ICAP Port Enabled (default) 1: Bottom ICAP Port Enabled
OverTempPowerDown	12	Enables the System Monitor Over-Temperature power down.
ConfigFallback	10	Stops when CFG fails and disables fallback to the default bitstream. The BitGen option is **ConfigFallback: Enable*/Disable**
SelectMAPAbort	9	Disable abort in SelectMAP when RDWR_B toggles when CS_B is asserted. The BitGen option is **SelectMAPAbort: Enable*/Disable**
GLUTMASK_B	8	Global LUT mask signal. Masks any changeable memory cell readback value.
DEC	6	AES Decryptor enable bit.
SBITS[1:0]	[5:4]	Security Level. 00: Read/Write OK (default) 01: Readback disabled 1x: Both Writes and Reads disabled
PERSIST	3	The configuration interface defined by M2:M0 remains after configuration. Typically used only with the SelectMAP interface to allow reconfiguration and readback. See also "SelectMAP Reconfiguration." 0: No (default) 1: Yes
GTS_USR_B	0	Active-Low global 3-state I/Os. Turns off pull-ups if GTS_CFG_B is also asserted. 0: I/Os 3-stated 1: I/Os active

Figura A.12: *Descripción del registro de control CTL0*

Registro de estado STAT: Este registro de sólo lectura indica el valor de las numerosas señales globales de estado de la FPGA.

Registro de opciones de configuración COR0: Este registro es usado para fijar ciertas opciones de configuración del dispositivo (Figuras A.13 y A.14).

Name	Bit Index	Description
CRC_BYPASS	28	Allows bypass of CRC when a special CRC value is loaded (0xDEFC): 0: CRC enabled 1: CRC disabled
PWRDWN_STAT	27	Changes the DONE pin to a Powerdown status pin: 0: DONE pin 1: Powerdown pin
DONE_PIPE	25	0: No pipeline stage for DONEIN 1: Add pipeline stage for DONEIN The FPGA waits on DONE that is delayed by one StartupClk cycle. Use this option when StartupClk is running at high speeds.
DRIVE_DONE	24	0: DONE pin is open drain 1: DONE is actively driven High
SINGLE	23	0: Readback is not single-shot New captured values are loaded on each successive CAP assertion on the CAPTURE_VIRTEX5 primitive. Capture can also be performed with the GCAPTURE instruction in the CMD register. 1: Readback is single-shot. The RCAP instruction must be loaded into the CMD register between successive readbacks.
OSCFSEL	[22:17]	Select CCLK frequency in Master modes (2 MHz–60 MHz)
SSCLKSRC	[16:15]	Startup-sequence clock source. 00: CCLK 01: UserClk (per connection on the CAPTURE_VIRTEX5 block) 1x: JTAGClk
DONE_CYCLE	[14:12]	Startup cycle to release the DONE pin. 000: Startup phase 1 001: Startup phase 2 010: Startup phase 3 011: Startup phase 4 100: Startup phase 5 101: Startup phase 6 110: Startup phase 7 111: Keep

Figura A.13: *Descripción del registro de opciones de configuración COR0 (1)*

Name	Bit Index	Description
MATCH_CYCLE	[11:9]	Startup cycle to stall in until DCI matches. 000 : Startup phase 0 001 : Startup phase 1 010 : Startup phase 2 011 : Startup phase 3 100 : Startup phase 4 101 : Startup phase 5 110 : Startup phase 6 111 : No Wait
LOCK_CYCLE	[8:6]	Startup cycle to stall in until DCMs lock. 000 : Startup phase 0 001 : Startup phase 1 010 : Startup phase 2 011 : Startup phase 3 100 : Startup phase 4 101 : Startup phase 5 110 : Startup phase 6 111 : No Wait
GTS_CYCLE	[5:3]	Startup cycle to deassert the Global Three-State (GTS) signal. 000 : Startup phase 1 001 : Startup phase 2 010 : Startup phase 3 011 : Startup phase 4 100 : Startup phase 5 101 : Startup phase 6 110 : GTS tracks DONE pin. BitGen option **-g GTS_cycle:Done** 001 : Keep
GWE_CYCLE	[2:0]	Startup phase to deassert the Global Write Enable (GWE) signal. 000 : Startup phase 1 001 : Startup phase 2 010 : Startup phase 3 011 : Startup phase 4 100 : Startup phase 5 101 : Startup phase 6 110 : GWE tracks DONE pin. BitGen option **-g GWE_cycle:Done** 111 : Keep

Figura A.14: *Descripción del registro de opciones de configuración COR0 (2)*

Registro de opciones de configuración COR1 El registro de opciones de configuración COR1 es usado para fijar ciertas opciones de configuración del dispositivo (Figura A.15).

Name	Bit Index	Description
PERSIST_DEASSERT_AT_DESYNCH	17	Enables deassertion of PERSIST with the DESYNCH command
RBCRC_NO_PIN	9	Disables INIT_B as read back CRC error status output pin
RBCRC_EN	8	Continuous readback CRC enable
BPI_1ST_READ_CYCLES	[3:2]	First byte read timing: 00: 1 CCLK 01: 2 CCLKs 10: 3 CCLKs 11: 4 CCLKs
BPI_PAGE_SIZE	[1:0]	Flash memory page size: 00: 1 byte/word 01: 4 bytes/words 10: 8 bytes/words 11: Reserved

Figura A.15: *Descripción del registro de opciones de configuración COR1*

A.2. Composición de un bitstream

En las próximas Figuras A.16, A.17 y A.18 se muestra la secuencia de comandos, palabras y paquetes de configuración de un bitstream típico para realizar una reconfiguración completa.

Configuration Data (hex)	Explanation
FFFFFFFF	Dummy Word
000000BB	BusWidth Word
11220044	8/16/32 BusWidth
FFFFFFFF	Dummy Word
FFFFFFFF	Dummy Word
AA995566	Sync Word
20000000	Type 1 NO OP
30020001	Type 1 write 1 words to WBSTAR
00000000	Warm Boot Start Address
30008001	Type 1 write 1 words to CMD
00000000	NULL
20000000	Type 1 NO OP
20000000	Type 1 NO OP
30008001	Type 1 write 1 words to CMD
00000007	RCRC
20000000	Type 1 NO OP
20000000	Type 1 NO OP
30022001	Type 1 write 1 words to TIMER
00000000	TIMER value
30026001	Type 1 write 1 words to CBC
00000000	CBC value
30012001	Type 1 write 1 words to COR0
0000401D	done@4 m@0 l@0 gts@3 gwe@5
3001C001	Type 1 write 1 words to COR1
00000000	COR1 value
30018001	Type 1 write 1 words to ID
02896093	ID code
30008001	Type 1 write 1 words to CMD
00000009	SWITCH
20000000	Type 1 NO OP
3000C001	Type 1 write 1 words to MASK
00000100	MASK value

Figura A.16: *Secuencia de configuración (1)*

Configuration Data (hex)	Explanation
3000A001	Type 1 write 1 words to CTL0
00000100	CTL0 value
3000C001	Type 1 write 1 words to MASK
00000000	MASK value
30030001	Type 1 write 1 words to CTL1
00000000	CTL1 value
20000000	Type 1 NO OP
20000000	Type 1 NO OP
20000000	Type 1 NO OP
20000000	Type 1 NO OP
20000000	Type 1 NO OP
20000000	Type 1 NO OP
20000000	Type 1 NO OP
20000000	Type 1 NO OP
30002001	Type 1 write 1 words to FAR
00000000	FAR value
30008001	Type 1 write 1 words to CMD
00000001	WCFG
20000000	Type 1 NO OP
30004000	Type 1 write 0 words to FDRI
50000400	Type 2 write 1024 words to FDRI
00000000	Data word 1
	Data word n
	Data word n + 1
00000000	Data word last (1024)
30000001	Type 1 write 1 words to CRC
F1927570	CRC value
30008001	Type 1 write 1 words to CMD
0000000A	GRESTORE
20000000	Type 1 NO OP
30008001	Type 1 write 1 words to CMD
00000003	LFRM

Figura A.17: *Secuencia de configuración (2)*

Configuration Data (hex)	Explanation
3000C001	Type 1 write 1 words to MASK
00000000	Data word 1
3000A001	Type 1 write 1 words to CTL0
00000000	CTL0 register value
20000000	Type 1 NO OP
	...
20000000	Type 1 NO OP
30008001	Type 1 write 1 words to CMD
00000005	START
20000000	Type 1 NO OP
30002001	Type 1 write 1 words to FAR
00FFFF80	FAR value
30000001	Type 1 write 1 words to CRC
845EAE07	CRC value
30008001	Type 1 write 1 words to CMD
0000000D	DESYNCH
20000000	Type 1 NO OP
	...
20000000	Type 1 NO OP

Figura A.18: *Secuencia de configuración (3)*

A.3. Direccionamiento de las frames (*Frame Addressing*)

Una frame es la unidad mínima de información de configuración a la que puede ser accedida. Se puede referenciar a ella como una columna vertical de 1312 bits de alto y 1 bit de ancho que recorre toda la altura de una región de reloj o *row* (fila) de la FPGA.

Una región de reloj o *row* consiste en una pila de bloques básicos (20 CLBs, 40 IOBs, 4 block RAMs, etc.) a los que atraviesa una pequeña franja del HCLK por medio. Entonces, de los 1312 bits mencionados de la frame, 640 bits se encuentran encima de la fraja del HCLK, otros 640 bits debajo y 32 bits se encuentran en dicha franja del HCLK.

Una frame de configuración está dividida dentro del registro de datos de frame (*Frame Data Register* (FDR)) en palabras de 32 bits. Para completar los 1312 bits, la frame está compuesta por 41 palabras de 32 bits (Figura A.19).

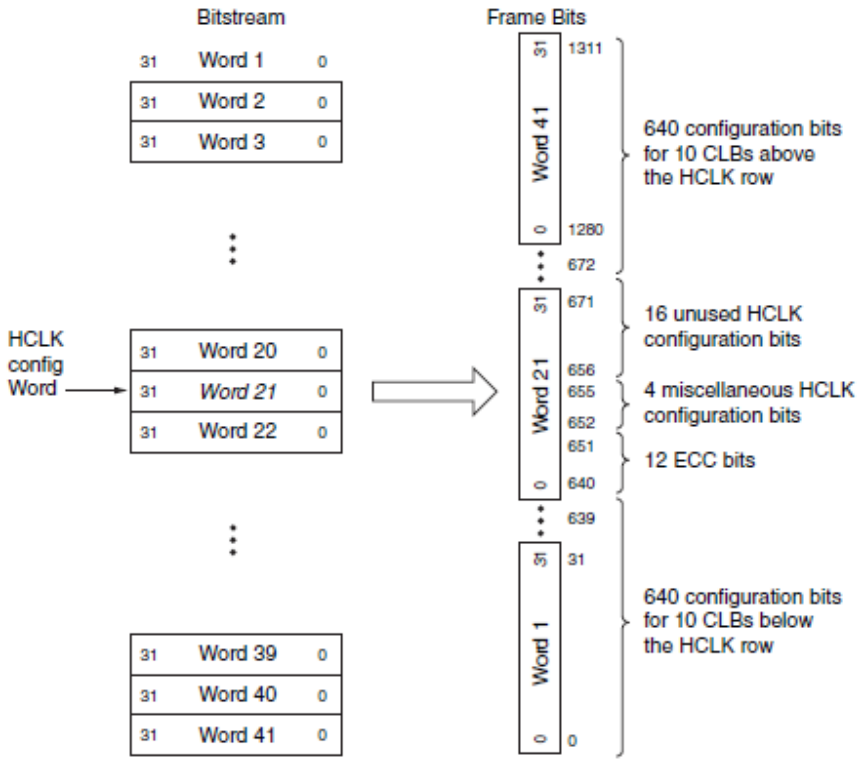

Figura A.19: *Palabras de configuración en un bitstream y bits de configuración en una frame*

Cada frame de configuración en la FPGA tiene una única dirección de 32 bits, dicha dirección se compone de 5 partes más una que no se utiliza (los 8 bits más significativos) (Figura A.20):

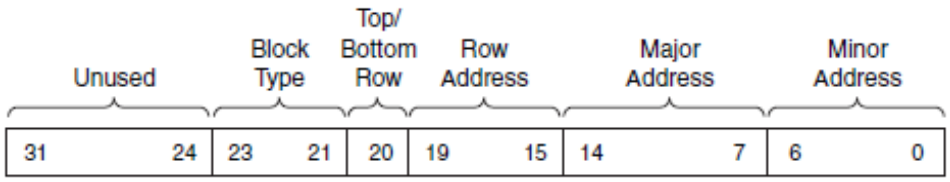

Figura A.20: *División de la dirección de 32 bits de una frame*

A.3.1. Tipo de bloque (*Block Type*)

El tipo de bloque divide el espacio de direcciones de configuración en 8 secciones; sin embargo, solamente 4 son utilizadas en los dispositivos Virtex-5. Esto sirve para dividir las frames de configuración dependiendo de su funcionalidad y de cómo son accedidas.

Los 4 tipos de bloques usados en los dispositivos Virtex-5 son:

- Interconexionado y bloques de configuración
- Contenido de las Block RAMs
- Frames de interconexionado y bloques especiales
- Frame de no-configuración de Block RAMs

A.3.2. Parte superior/inferior (*Top/Bottom Indicator*)

Desde el punto de vista de configuración, la FPGA está dividida en dos partes: la superior e inferior. Sólo se necesita 1 bit para diferenciar entre estas dos posibles mitades.

A.3.3. Dirección de la región de reloj o *row* (*Row Address*)

Dentro de cada mitad en la que está dividida la FPGA, las regiones de reloj o *rows* están numeradas de 0 a 9 empezando siempre por el centro (Figura A.21).

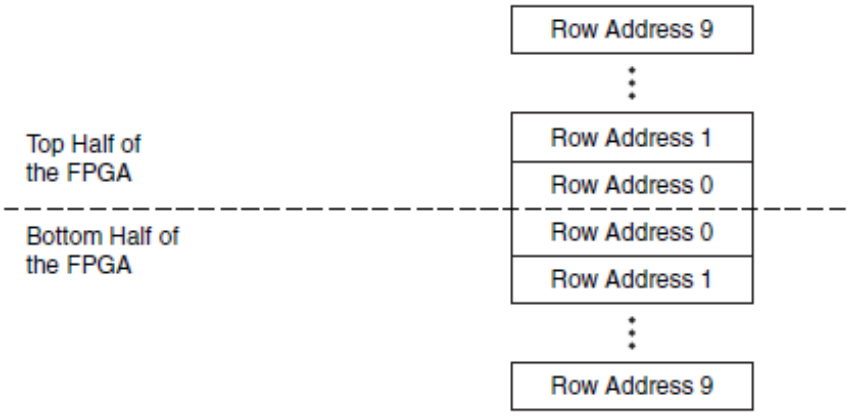

Figura A.21: *Top/Bottom y dirección de la región de reloj o row en la FPGA*

Aunque el hardware del diseño puede soportar el manejo de 20 regiones de reloj o *rows* (10 en cada mitad), dependiendo del tipo de Virtex-5 puede que la FPGA tenga más o menos regiones de reloj o *rows*. En el caso de la Virtex-5 más grande posee un total de 12 regiones de reloj o *rows* (6 en cada mitad) mientras que en la Virtex-5 XC5VLX110T usada en este proyecto, únicamente tiene un total de 8 regiones de reloj o *rows* (4 en cada mitad).

A.3.4. Dirección Major (*Major Address*)

Cada región de reloj o *row* está dividida en columnas, donde cada columna corresponde con un array de bloques básicos (20 CLBs, 40 IOBs, 4 block RAMs, etc.). La dirección major o cada columna de bloques básicos está numerada desde la izquierda hasta la derecha de la FPGA, empezando por 0.

A.3.5. Dirección Minor (*Minor Address*)

Cada columna de bloques básicos, a la que nos referíamos en la sección anterior A.3.4, contiene un cierto número de frames de configuración, las cuales se acceden usando la dirección minor. El número de frames dentro de cada columna depende del tipo de bloque que sea la columna (Figura A.22).

Block	Number of Frames
CLB	36
DSP	28
Block RAM	30
IOB	54
Clock Column	4
Clock Column	4

Figura A.22: *Número de frames (dirección minor) por columna*

Las frames están numeradas desde la izquierda hasta la derecha del bloque, empezando por 0.