UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

TESIS DOCTORAL

Emulación basada en FPGA de los efectos de los single event upsets ocasionados por la radiación en circuitos digitales tolerantes a fallos

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Felipe Serrano Santos

DIRECTORES

Hortensia Mecha López
Juan Antonio Clemente Barreira

Madrid, 2018

Emulación basada en FPGA de los efectos de los single event upsets ocasionados por la radiación en circuitos digitales tolerantes a fallos

TESIS DOCTORAL

Doctorando:
Felipe Serrano Santos

Directores:
Hortensia Mecha López
Juan Antonio Clemente Barreira

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Madrid 2017

Agradecimientos

Quiero agradecer enormemente a mis padres el ánimo y apoyo incondicional que me han dado siempre ya que han sido un pilar fundamental para mi y han hecho que esta tesis sea posible.

A Antonio por haber estado ahí en los momentos de más frustración y haberme sabido escuchar cuando más lo necesitaba.

Agradecer también a mis directores de tesis, Hortensia y Juan Antonio, la paciencia que han tenido conmigo y por todo el trabajo con el que me han ayudado en forma de revisiones, dudas y soluciones a problemas imposibles. También por darme el empujón necesario incontables veces para no tirar la toalla y seguir adelante en los momentos de mayor duda.

Por último, agradecer a toda la gente con la que he trabajado y colaborado durante esta investigación por todo lo que me han aportado.

Índice

Índice de figuras

Índice de tablas

Abreviaturas

- FPGA – Field Programable Gate Array
- SEU – Single Event Upset
- SEE – Single Event Effect
- SEFI – Single Event Functional Interrupt
- SEL – Single Event Latchup
- MBU – Multiple Bit Upset
- CLB – Configurable Logic Block
- LUT – Look Up Table
- FF – Flip Flop
- BRAM – Block Random Access Memory
- DSP – Digital Signal Processor
- CRC – Cyclic Redundancy Check
- VLSI – Very Large Scale Integration
- FT – Fault Tolerant

Resumen

Emulación basada en FPGA de los efectos de los single event upsets ocasionados por la radiación en circuitos digitales tolerantes a fallos

por

Felipe Serrano Santos

Esta investigación explora los efectos de la radiación cósmica en circuitos digitales. El trabajo está dividido en dos partes. La primera parte de este trabajo se centra en poder emular uno de los efectos más comunes causados por dicha radiación: los Single Event Upsets (SEUs). La emulación de SEUs se realiza en dispositivos reconfigurables como son las FPGAs, ya que han ganado especial relevancia en sectores donde la radiación está presente, como son el sector aeroespacial y el de defensa. Esto es debido a las características tan beneficiosas que ofrecen en cuanto a consumo, flexibilidad, coste y prestaciones. Por ello, el principal objetivo ha sido aportar una herramienta de emulación de SEUs que no tenga las limitaciones del resto de herramientas existentes en el estado del arte. Esto significa que la herramienta debe ser no intrusiva, determinista y eficiente. La segunda parte de esta tesis doctoral, se ha centrado en explorar nuevas técnicas y metodologías en el ámbito del diseño digital tolerante a fallos para poder enfrentarse a la problemática que supone trabajar en entornos radiactivos.

El resultado de este trabajo es una herramienta llamada NESSY, que permite inyectar errores en los distintos elementos de memoria de diseños digitales, emulando de esta manera a los SEUs. Debido a la heterogeneidad de las FPGAs, la manera de realizar dichas inyecciones difiere en función del tipo de elemento de memoria que pueda ser afectado por la radiación. Por lo tanto, se distinguen dos casos dependiendo de si el elemento de memoria pertenece a la memoria de configuración del dispositivo, o de si este elemento es un flipflop del propio circuito (común a cualquier diseño digital). Por este motivo, NESSY implementa dos metodologías para inyectar un error. La primera, realiza la inyección del error en bits de la memoria de configuración de las FPGAs, siendo una inyección eficiente, no intrusiva y determinista. Dicha metodología tiene cabida cuando se trabaja con circuitos implementados en FPGAs puesto que la memoria de configuración es propia de estos dispositivos. Para

complementar a la primera, existe la segunda metodología que permite inyectar errores en flipflops. La inyección en flipflops también se realiza de manera eficiente y determinista pero se ha tenido que sacrificar la característica de no intrusividad, con el fin de poder mantener la eficiencia. Dicha metodología es aplicable a cualquier circuito digital y no se limita a los implementados en FPGAs. Esto supone el que sea posible evaluar la vulnerabilidad frente a SEUs de un diseño antes de su fabricación, traduciéndose en un beneficio tanto económico como de calendario.

Gracias al desarrollo de esta herramienta, ha sido posible avanzar en la otra línea principal de esta investigación: el estudio de nuevas técnicas de protección de circuitos. Primero, se han decidido probar diferentes variantes de un método de protección de circuitos bien conocido y validado: la triple redundancia modular (TMR), con el fin de validar la herramienta. Para ello, se han realizado distintas campañas de inyección de errores a un conjunto de circuitos cuando éstos no tenían protección alguna (diseño original), y con distintos niveles de protección basados en la técnica TMR para poder comprobar así, que efectivamente los resultados obtenidos son consistentes a la protección esperada por dicha técnica. A continuación, se han desarrollado técnicas de protección novedosas para familias de circuitos específicas. La primera de ellas se centra en circuitos basados en redes neuronales de tipo Hopfield (HNN) y se basa en aplicar una redundancia parcial inteligente para obtener mejores resultados que los ofrecidos por el TMR. La segunda técnica de protección investigada se centra en circuitos que utilizan módulos hardware de propósito específico conocidos como procesadores digitales de señal (DSPs). Existen multitud de circuitos que hacen gran uso de operaciones aritméticas que puede beneficiarse de dicha técnica.

Abstract

FPGA based emulation of the single event upsets due to radiation in fault tolerant digital circuits

by

Felipe Serrano Santos

This research explores the effects of cosmic radiation on digital circuits. The work is divided in two differentiated parts. The fist one focuses on the emulation of one of the most typical effects caused by radiation: the Single Event Upsets (SEUs). The emulation of the SEUs is carried out on reconfigurable devices such as FPGAs. Said devices have obtained special relevance in sectors where radiation is an issue, such as aerospace and defense. This relevance is due to their useful features involving power consumption, flexibility, cost and performance. The objetive has been to presesnt a fault injection tool that circumvents the limitations of other similar tools in the state of the art, especially in terms of non-intrusiveness and performance. In other words, it has to be non-intrusive, deterministic and efficient. The second part of the doctoral thesis explores new techniques and methodologies in the field of fault tolerant digital circuits in order to palliate the problems related to working in radioactive environments.

The result of this work is a tool called NESSY, which allows to emulate SEUS by means of injecting errors in the memory cells of digital designs. The FPGAs are heterogeneous, so the method to carry out the fault injections is different depending of the type of memory cell that can be affected by radiation. Thus, two possible cases exist: 1) the memory cell belongs to the configuration memory of the device and 2) the memory cell is actually a flipflop in the circuit (present in every digital design). Because of that, NESSY implements two methodologies to inject an error. The first one makes the injection in the bits of the configuration memory of the FPGA in a non-intrusive, efficient and deterministic way. This methodology can be used only when the target circuit is implemented using a FPGA because this kind of memory is present only in such devices. The second methodology makes NESSY able to inject faults in flipflops. This method is also efficient and

deterministic; however, the non-instrusiveness feature had to be eliminated in order to be able to keep it efficient. This second methodology is applicable to every digital design and thus, it is not limited to implementations in FPGAs. These two methodologies altogether allow evaluating the vulnerability of any circuit against SEUs, prior to manufacturing it, thereby saving money and time.

Thanks to the developed tool NESSY, it has been possible to move towards the other main line of the research of this doctoral thesis: the study of new circuit protection techniques. First of all, a well-known protection technique has been tested: the Triple Modular Redundancy (TMR), in order to validate NESSY. For that purpose, several injection campaigns has been carried out on a set of circuits. These circuits have been used with different levels of protection, ranging from the original ones (unprotected, and thus vulnerable to SEUs) to several levels of protection based in TMR. Then, two novel new protection techniques for specific families of circuits have been presented and validated. The first one is used with Hopfield Neural Networks (HNN) and it consists in applying intelligent and partial redundancy to obtain an implementation nearly as robust as a triplicated design, but that considerably reduces its resources consumption. The second technique can be applied to circuits that use specific purpose hardware modules known as Digital Signal Processors (DSPs). There exist a lot of circuits that perform intensive arithmetic operations that can benefit from this technique.

Capítulo 1: Introducción

Los dispositivos reconfigurables, y en especial, las FPGAs basadas en memoria SRAM (SRAM-FPGAs) han ganado popularidad para su uso en campos como la tecnología nuclear, la aviación y las misiones espaciales, por mencionar algunos. Los principales motivos son su alta densidad de recursos, su bajo coste y su capacidad de reconfigurarse. Esta última funcionalidad hace que las FPGAs sean dispositivos versátiles y da lugar a una serie de posibilidades que las implementaciones tradicionales de diseños electrónicos en PCB o ASIC no ofrecen. Estas opciones van desde la opción de multiplexar en el tiempo partes del diseño sobre los mismos recursos para aumentar la cantidad de lógica disponible, hasta que el propio diseño pueda autoreconfigurarse a sí mismo modificando su comportamiento de manera similar a como hacen los algoritmos evolutivos o de inteligencia artificial.

Sin embargo, un problema frecuente en los sectores mencionados, especialmente en el sector espacio, es la alta dosis de radiación a la cual estos dispositivos están expuestos. Esta radiación puede causar errores conocidos como Single Event Effects (SEEs), los cuales pueden alterar el funcionamiento de los componentes electrónicos utilizados en los sistemas embarcados.

Estos eventos ya fueron estudiados en los años 60 por el Naval Research Laboratory (NRL) [39], cuando estudiaban la sensibilidad frente a radiación ionizante en dispositivos MOS. Sin embargo, no fue hasta más adelante cuando se tuvo evidencia real de los mismos debido a la aparición de errores en la electrónica de satélites y aviones. Esto se debe a que hay una relación directa entre la sensibilidad a la radiación ionizante y la tecnología de integración de los dispositivos. Puesto que la tendencia tecnológica es la de disminuir el tamaño de los canales en los transistores y reducir la tensión de alimentación de éstos, se puede prever cómo los SEEs van a ser un problema cada vez más importante en los sistemas electrónicos.

Para entender mejor esta problemática, en primer lugar hay que entender la fuente del problema. Los orígenes de la radiación son variados (Figura 1) pero se pueden clasificar en

dos grandes grupos:

1. Radiación de partículas: por ejemplo, electrones, protones, neutrones o iones pesados.
2. Radiación electromagnética: por ejemplo, proveniente de rayos alfa, rayos X o ultravioleta.

En el caso de las partículas cargadas, éstas interaccionan con el campo magnético terrestre desviándose o siendo atrapadas por éste. Sin embargo, las partículas sin carga como los neutrones o los fotones no interaccionan con el campo magnético y pueden ser encontradas al nivel del mar.

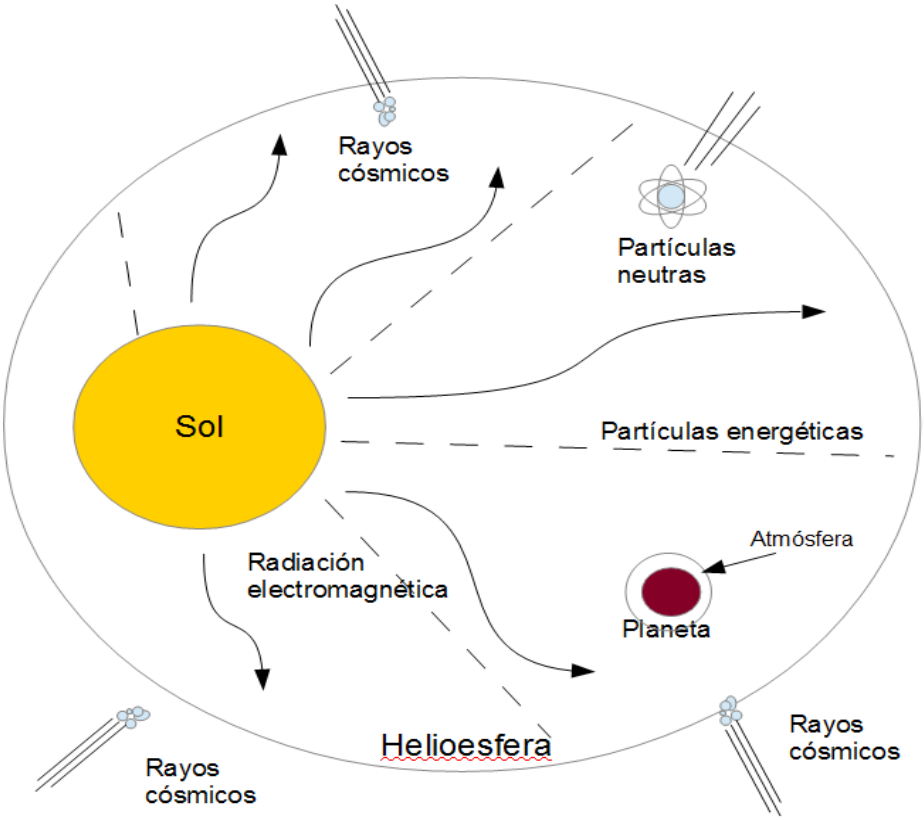

Figura 1: Fuentes de radiación en el espacio

La radiación de partículas energéticas es la causa más importante de efectos dañinos en

circuitos electrónicos y la más difícil de paliar. La interacción de una de estas partículas con un semiconductor puede ocasionar un desplazamiento atómico, lo cual afecta a la estructura interna del semiconductor (a este efecto se le conoce como "microdose desplacement"). Si se produce una acumulación de estos efectos, las propiedades eléctricas del semiconductor se pueden alterar permanentemente [35].

Otro posible efecto de la interacción de este tipo de radiación con un semiconductor es la transferencia de energía a éste. Se puede dar el caso de que en un semiconductor, los electrones de valencia pasen a ser electrones de conducción debido a la energía aportada por una partícula que haya colisionado (Figura 2). Esto puede traducirse en un transitorio de corriente en un dispositivo como puede ser un transistor bipolar, un diodo Zener o un transistor de efecto de campo de tecnología MOS (MOSFET).

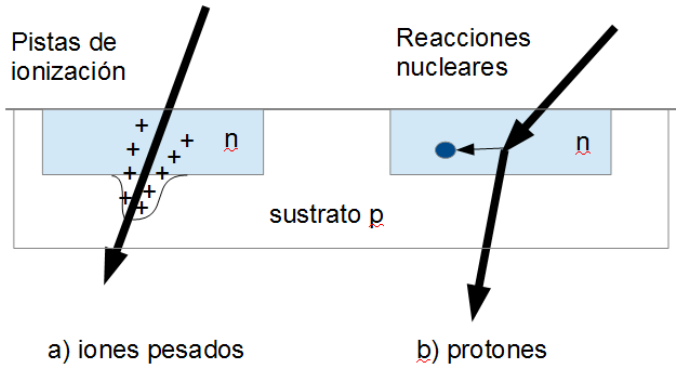

Figura 2: Interacción de partículas energéticas con semiconductores

Si esta situación ocurre en un dispositivo que es parte de un elemento de memoria, puede resultar en un bitflip, esto es, un cambio no intencional en el contenido de la memoria, también llamado Single Event Upset (SEU).

1.1. Efectos de la radiación cósmica

Las principales consecuencias de la radiación cósmica en dispositivos electrónicos son cambios de energía en elementos del dispositivo afectado. Si esto ocurre en un semiconductor, se generan más pares electrón-hueco en función de la energía de la partícula y esto puede dar lugar a cambios de estado de la unión PN no deseados. A estos cambios se les conoce como Single Event Effects (o SEEs por sus siglas en inglés) y los errores derivados de estos cambios se pueden clasificar en dos grandes grupos:

- Errores permanentes ("hard errors").
- Errores no permanentes ("soft errors").

A continuación, se proporcionan más detalles de estos dos tipos de errores.

1.1.1. Errores permanentes

El tipo de error permanente más común se produce en dispositivos que utilizan tecnología CMOS, es el Single Event Latchup (SEL), que consiste en la aparición en su par PMOS-NMOS de un par de transistores bipolares parásitos PNP y NPN (esas estructuras parásitas se pueden ver como una unión PNPN o NPNP, tal y como se muestra en la Figura 3). Dicha estructura no supone ningún problema bajo condiciones normales ya que está "apagada" y no circula corriente por ella. Sin embargo, cuando una partícula impacta con ella, aparece un pulso de corriente en esta estructura y acaba abriendo un camino entre la alimentación y tierra. Esta estructura PNPN o NPNP, que se comporta como un tiristor [20], no entra en corte a no ser que se apague la alimentación. Por lo tanto, debido a la disipación de calor, se puede llegar a quemar el dispositivo.

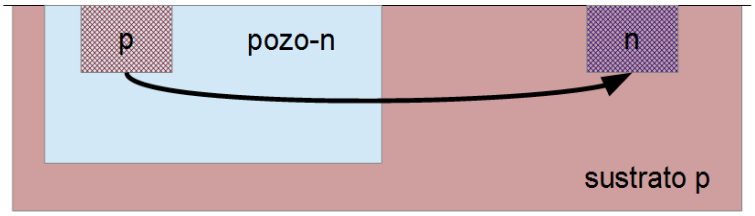

Figura 3: Error permanente de tipo Single Event Latchup

Otra causa de la aparición de un SEL se debe a la progresión en la escala de integración de los transistores. Tecnológicamente, se ha llegado a un punto donde las tensiones de conmutación de los dispositivos y tensiones de alimentación, son comparables a los que genera una partícula incidente [16]. Esto hace posible que una acumulación de energía transmitida a una puerta MOS ponga el transistor en estado activo. El efecto puede ser permanente y dejar el transistor en un estado fijo.

1.1.2. Errores no permanentes

Como ya se ha comentado anteriormente, un Single Event Upset (SEU) es un evento que ocurre cuando una partícula ionizante transfiere energía a un semiconductor que forma parte de un elemento de memoria (Figura 4). Si la energía transferida es suficiente, la carga almacenada por este elemento variará y puede que su nivel lógico llegue a cambiar.

Si una partícula afecta a varios elementos de memoria adyacentes, se produce un Multiple Event Upset (MBU) [36]. Este fenómeno ha pasado desapercibido por su baja probabilidad de ocurriencia con respecto a los SEUs. Sin embargo, según se aumenta la escala de integración de los transistores y éstos son cada vez más pequeños, es más probable que una única partícula ionizante incida sobre varios transistores simultáneamente, por lo que este fenómeno cobra cada vez una mayor importancia.

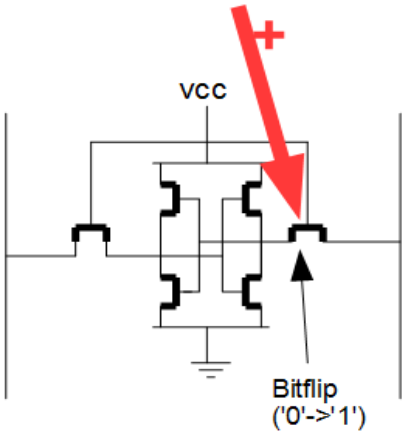

Figura 4: Single Event Upset en una celda de memoria

El segundo tipo de errores no permanentes que se presenta en esta sección son los Single Event Transients (SETs). Cuando la partícula impacta sobre una línea combinacional, durante un periodo de tiempo t se produce un transitorio de corriente en dicha línea debido a la transferencia de energía de la partícula. Esto no es un problema si este transitorio no llega a ser registrado por una señal de reloj, ya que acaba desapareciendo.

Un subtipo particular de SET que podría considerarse una categoría aparte debido a su importancia, son los Single Event Functional Interrupts (SEFIs). Un SEFI no es más que un SET sobre una señal crítica asíncrona como puede ser el reset de un sistema y cuyas consecuencias son graves, puesto que puede suponer la ejecución de una rutina fuera de lugar o el reseteo del sistema completo.

1.2. Hardware dinámicamente reconfigurable

Las FPGAs basadas en memoria SRAM son particularmente sensibles a los SEUs debido a la gran cantidad de memoria que estos dispositivos poseen. En particular, lo es la memoria de configuración cuyo contenido determina la funcionalidad del sistema a nivel lógico. Por tanto, para garantizar el correcto funcionamiento de un sistema implementado sobre estos dispositivos en entornos radioactivos, encontrar una manera de paliar el efecto

de los SEUs es de vital importancia. Dicho problema es abordable a distintos niveles:

1. Nivel físico o de semiconductor: esto es, basándose en la física de los transistores o diodos, variar la estructura de capas o la geometría de los canales para hacerlos más robustos.
2. Nivel de material: probar distintos materiales para la construcción de las uniones PN.
3. Nivel de dispositivo: crear celdas de memoria más complejas o redundantes que enmascaren los SEEs.
4. Nivel RTL: redundancia de partes sensibles a errores para poder detectar y enmascarar los bitflips. Esta tesis doctoral se enmarca en este nivel.
5. Nivel de Aplicación: redundancia de datos por software para ser capaz de detectar posibles errores en los bits de las memorias mediante códigos de detección de errores (CRC, checksum, etc).

Por otro lado, los elementos de memoria de una FPGA no son homogéneos. En el caso de la familia Virtex del fabricante Xilinx, que son las que se han utilizado en esta tesis doctoral, las celdas de memoria susceptibles a SEUs se pueden clasificar en tres grupos:

1. Memoria de configuración.
2. Datos almacenados en Block RAM (BRAM).
3. Estado de los flipflops (Biestables D).

Comparando la relación entre los tres grupos de celdas mencionados, la memoria de configuración es la que mayor cantidad de bits por unidad de superficie tiene en cualquier empaquetado de FPGA, como se indica en la Tabla I. Por tanto, la probabilidad de impacto de una partícula sobre uno de estos bits es mayor.

Tipo		#Bits
Configuración	CLB	46080
	DSP	35840
	BRAM	5120
Flip flops		160
Datos BRAM		5120

Tabla I: Número de bits de configuración para distintos tipos de celdas en una FPGA Xilinx tipo Virtex-5

Dentro de una FPGA, existen distintos recursos con una arquitectura y funcionalidad muy distinta entre sí. En la familia Virtex de FPGAs, como principales recursos dentro de su matriz de elementos configurables se pueden distinguir: los Configurable Logic Blocks (CLBs), destinados a implementar las funciones lógicas; los Digital Signal Processors (DSPs), utilizados para acelerar operaciones aritméticas y los Block Random Access Memories (BRAMs) que tienen como objetivo optimizar el almacenamiento y procesamiento de datos en grandes volúmenes.

Por lo tanto, cada bit de configuración no tiene la misma importancia debido al recurso al que está asociado ya que por ejemplo, no tendrá las mismas consecuencias en un diseño que un DSP cambie su operación de suma por una resta (bit de configuración de control) a que una función lógica varíe de implementar una función OR a una función AND.

En los siguientes apartados, se proporcionará una descripción detallada de cada uno de los elementos reconfigurables mencionados.

1.2.1. Configurable Logic Blocks (CLBs)

Un bloque lógico configurable (o CLB por sus siglas en inglés) es el elemento encargado de implementar las funciones booleanas en las FPGAs de Xilinx. Los CLBs de las FPGAs Virtex-5 están constituidos a su vez por dos slices conectados a una matriz de interconexionado como se muestra en Figura 5, tomada de [46].

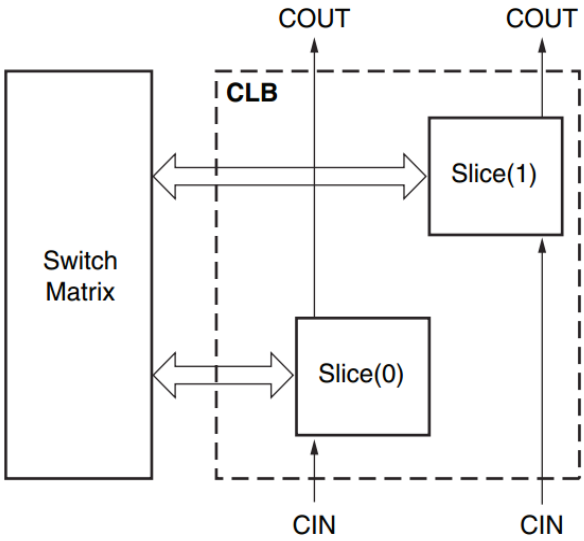

Figura 5: Arquitectura interna de un CLB en FPGAs de tipo Virtex-5

Por cada slice, existen cuatro Look-Up Table (LUT), donde se implementan las funciones combinaciones y cuatro biestables para registrar datos. Además, tiene lógica adicional para determinar el comportamiento de ciertos recursos o transmitir bits de acarreo en determinadas configuraciones. Esto se aprecia en la Figura 6 (tomada de [46]).

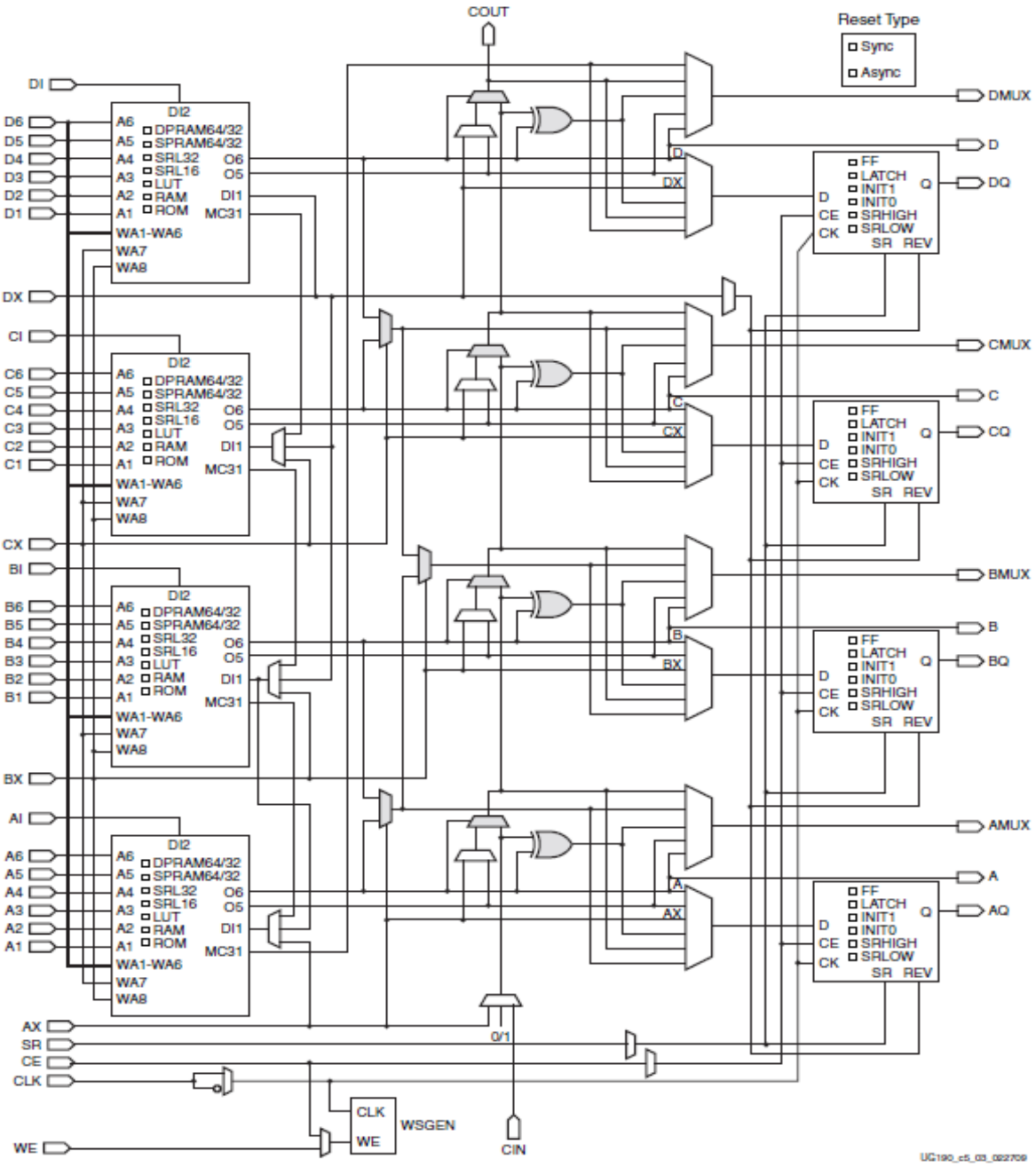

Figura 6: Arquitectura interna de un slice en FPGAs de tipo Virtex-5

Por lo tanto, los bits de la memoria de configuración de los CLBs, son los que determinan el comportamiento de dichas funciones lógicas, si hay acarreo, como se comportan ciertas partes del CLB o cómo se comportan los biestables D (asíncronos, síncronos, reset activo a baja o alta, etc) inferidos en el diseño. Están conectados entre ellos a través de su matriz de interconexionado.

1.2.2. Digital Signal Processors (DSPs)

Las aplicaciones de procesado de señal son cada vez más comunes y últimamente han ganado una mayor importancia, especialmente en aplicaciones utilizadas en el sector aeroespacial y de defensa. Por este motivo, las FPGAs modernas se diseñan con arquitecturas cada vez menos homogéneas, que incluyen recursos de propósito específico dentro de su matriz de bloques configurables. Esto permite ahorrar recursos hardware y ofrecer mejores prestaciones para ejecutar aplicaciones. En el caso de las FPGAs de tipo Virtex-5, existen unos bloques orientados al procesado de señal digital (digital signal processors o DSPs por sus siglas en inglés) capaces de implementar sumas y multiplicaciones registradas de manera eficiente. Ver Figura 7 (tomada de [46]).

Estos bloques permiten implementar más funcionalidad con menos recursos hardware con respecto a los CLBs. Solo son aplicables en ciertas aplicaciones como son los filtros digitales. Es por ello que ciertas implementaciones de diseños no serían realistas si se realizaran sobre los bloques lógicos configurables de la FPGA y por lo tanto, un estudio de la fiabilidad de dichos diseños frente a SEUs no tendría validez.

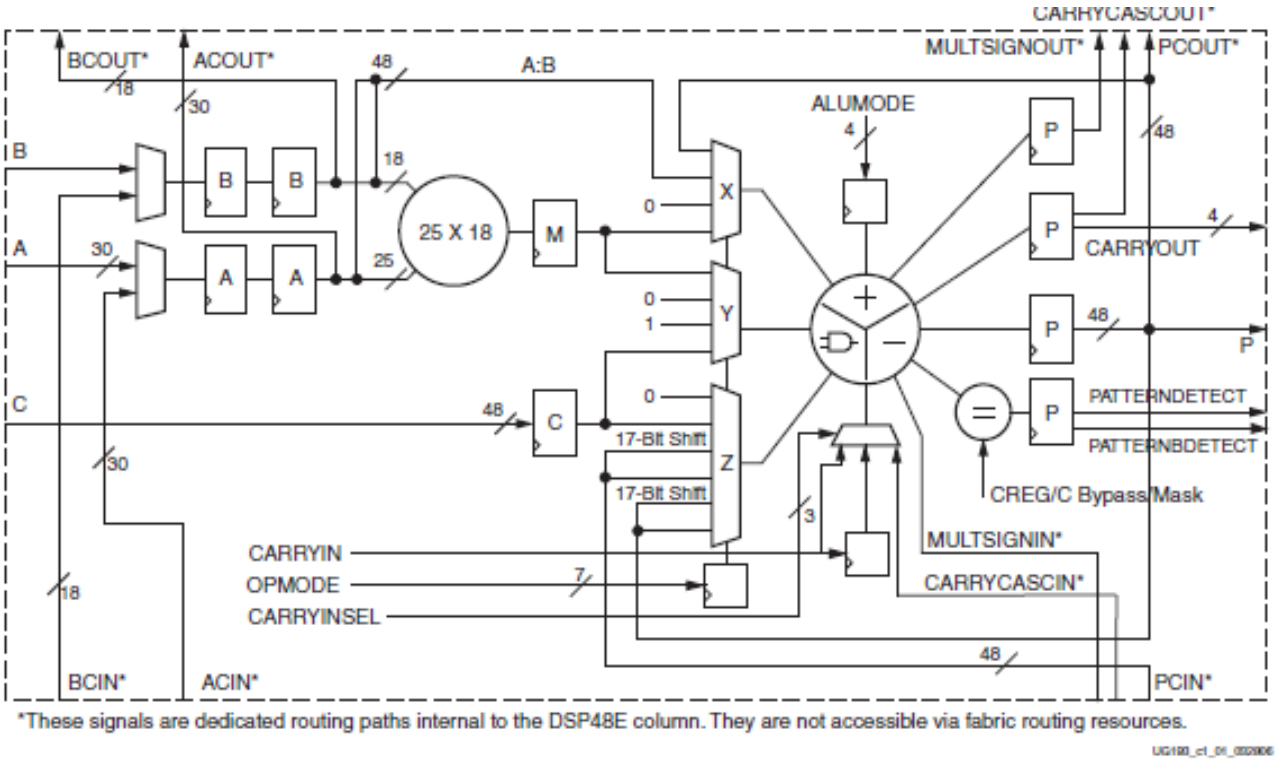

Figura 7: Arquitectura interna de un DSP en FPGAs de tipo Virtex-5

Al igual que los CLBs, los DSPs de las Virtex-5 tienen una memoria de configuración que determina las conexiones internas de éstos. Por lo tanto, la memoria de configuración de los DSPs es la responsable de:

- Agrupar varios DSPs para tamaños de datos grandes mediante acarreos.
- Registrar o no las entradas/salidas mediante el uso de biestables.
- Tipo de operación aritmética a realizar.
- Formato de los datos a procesar (con signo, sin signo, etcétera).

1.2.3. Block Random Access Memory (BRAM)

Las Block RAMs son memorias que están embebidas y repartidas a lo largo de la FPGA. Debido a la alta latencia de las memorias externas, estas memorias ofrecen una alternativa a un flujo de datos continuo y de altas prestaciones, al tener una latencia mínima.

El hecho de que estén embebidas en la FPGA supone que existe cierta información almacenada en la memoria de configuración que está asociada a una funcionalidad en el circuito que se implementa, al igual que el resto de los elementos de la FPGA. Por lo tanto, dicha información se puede clasificar en los siguientes dos tipos:

1. Bits de configuración.
2. Bits de datos.

Son los bits de configuración los que gestionan características tales como el tipo de flanco que se quiere usar o si se registra el dato de salida. Tiene elementos de memoria de las celdas que guardan los datos en sí, lo que convierte a estos elementos en candidatos potenciales para ser afectados por la radiación. Un diagrama a nivel de bloques de la arquitectura de estas BRAM se muestra en la Figura 8 (tomada de [46]).

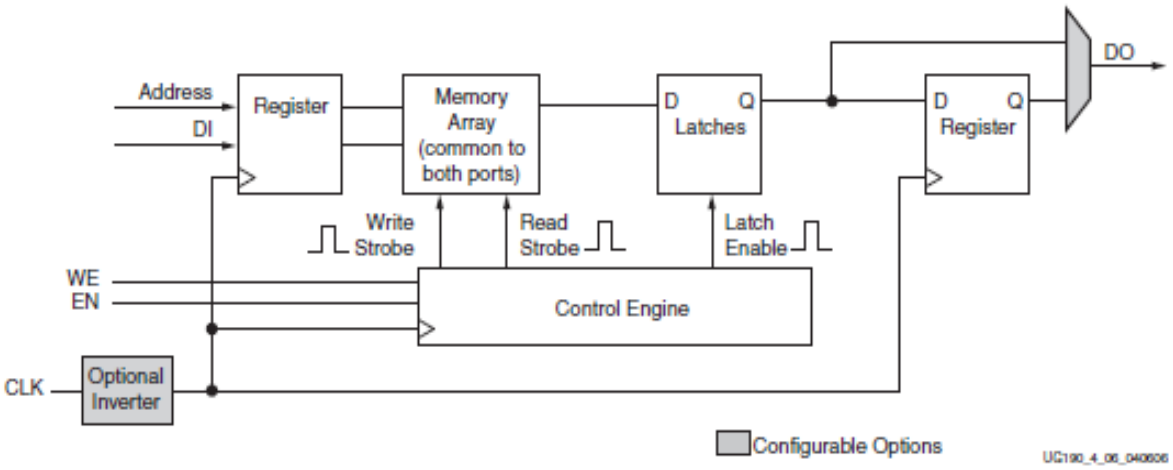

Figura 8: Arquitectura interna de un BRAM en FPGAs de tipo Virtex-5

1.3. Contribuciones de esta tesis doctoral

La principal contribución de esta tesis doctoral consiste en una herramienta llamada NESSY, que permite emular el efecto de los SEUs en circuitos digitales. Para emular un SEU, se pueden utilizar distintos métodos según con qué fin se vaya a utilizar la información del estudio. Hay herramientas que emulan la aparición de SEUs de manera probabilista, siguiendo ciertas distribuciones estadísticas que tratan de imitar el comportamiento de la radiación real. Sin embargo, este método no permite hacer un análisis exhaustivo del efecto, solamente hacerse una idea de cómo de robusto es el diseño de manera global. La herramienta desarrollada en esta tesis doctoral soluciona esto y aborda la emulación de SEUs de tal manera que permite una mayor flexibilidad a la hora de analizar el impacto de la radiación en los circuitos y realiza la inyección de errores de una manera controlada y determinista. En otras palabras, NESSY es capaz de realizar una inyección de errores en todos los bits almacenados en elementos de memoria del diseño, iterando sobre ellos e inyectando únicamente un SEU por cada bit, asegurándose que se cubre la totalidad de los bits del circuito susceptibles a errores.

La manera de inyectar errores difiere en función del tipo de celda de memoria sobre la que se quiere trabajar. Se distingue entre si el elemento de memoria es un flipflop del propio diseño o es parte de la memoria de configuración de un dispositivo reconfigurable, dando lugar a dos métodos de inyección bien diferenciados para la herramienta. Mientras que los primeros pueden existir tanto en FPGAs como en circuitos integrados de aplicación específica (o ASICs por sus siglas en inglés), la memoria de configuración existe únicamente en los primeros. Por lo tanto, si se quiere realizar una inyección de errores en la memoria de configuración, es porque la arquitectura destino del diseño bajo test es una FPGA. En este caso, debido a la flexibilidad que ofrece el hardware reconfigurable, NESSY parte de la base de que las aplicaciones que van a ser examinadas se van a ejecutar en un entorno multitarea que sigue el esquema de la Figura 9.

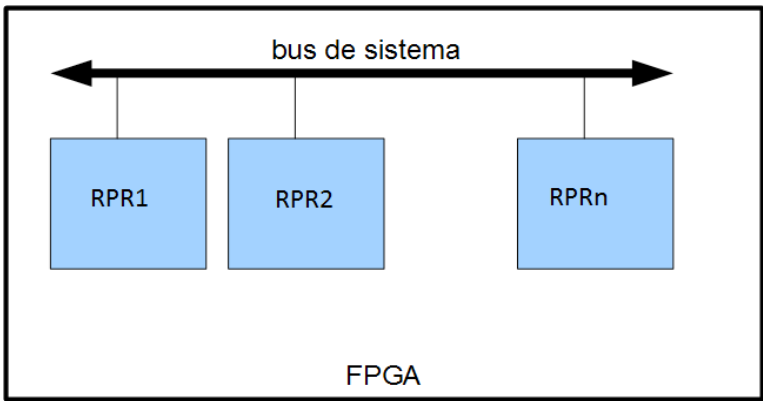

Figura 9: Arquitectura multitarea en FPGAs

Por lo tanto, en el caso de que la arquitectura destino sea la propia FPGA, el circuito bajo test será instanciado en una Región Parcialmente Reconfigurable (RPR) aislada e independiente de las otras, haciendo que el rutado de éste no se vea afectado por el resto del hardware de la plataforma. Cada RPR ejecutará una tarea independiente, dando lugar a una arquitectura modular. Esto permite que la plataforma tenga control total sobre el estado del diseño que se quiere examinar, pudiendo controlar las entradas y el reloj, además de monitorear sus salidas constantemente.

Este modelo de arquitectura no introduce ninguna restricción adicional de tamaño o posicionamiento de las RPRs. Tales arquitecturas son flexibles y facilitan la multitarea

permitiendo hacer uso de la tecnología de reconfiguración parcial. Su eficacia ha sido probada en varios trabajos de investigación del estado del arte [28] [29] [30]. Gracias a todo esto, NESSY destaca respecto a otras plataformas similares principalmente por:

- La no intrusividad en la inyección sobre memoria de configuración: NESSY es transparente para el diseño bajo test cuando se inyectan errores en la memoria de configuración, es decir, el usuario puede definir la RPR en la FPGA para colocar y rutar el circuito bajo test. El diseño original será implementado y no es necesario modificarlo añadiendo ningún hardware de control adicional. Por lo tanto, la herramienta sólo inyectará bitflips en esta región específica y así evaluar el impacto de los bitflips sólo en el circuito que se testea, obteniendo unos resultados que son extrapolables al comportamiento del circuito en su comportamiento nominal. Sin embargo, esto no ocurre en el método de inyección de errores sobre flipflops, ya que en este caso sí se necesita añadir un hardware adicional que afecta al rutado de éste.
- Eficiencia: Es más eficiente que otros sistemas presentados en la literatura [5], [43], [19]. De hecho, inyectar un bitflip con NESSY requiere 98,13µs, que es al menos dos órdenes de magnitud más rápido que con cualquiera de estos otros sistemas. Esto es debido a que la reconfiguración se hace internamente en la misma FPGA. Se proporcionan más en detalles en la sección 2.2.
- Bajo coste: NESSY sólo necesita una única FPGA y un PC.
- Portabilidad y escalabilidad: Nuestro sistema es válido para cualquier FPGA de Xilinx ya que la herramienta se puede reconfigurar con la lógica reconfigurable disponible en cualquier FPGA de Xilinx. Cada vez que se quiera realizar una inyección de errores, la herramienta se reconfigura desde cero en la FPGA destino, adaptándose a su arquitectura.

Debido a la inyección de errores controlada y determinista que realiza NESSY y al estudio que se ha realizado sobre la arquitectura de la memoria de configuración de la familia Virtex, los resultados obtenidos en esta tesis doctoral (capítulo 5) pueden ser clasificados según el bit tratado pertenezca a:

- La matriz de conexionado.

- La lógica del circuito.
- Bit no utilizado por el diseño, pero que está dentro de su región configurable.

La segunda aportación de esta tesis es la validación de técnicas de protección para circuitos (sección 4.1): en primer lugar la triple redundancia modular (TMR por sus siglas en inglés) y en segundo lugar, diferentes variantes de TMR basadas en según qué recursos se utilicen para su implementación o cómo se aplica esta redundancia. En esta validación se han utilizado diseños de la batería de benchmarks ITC'99 desarrollado por investigadores del Politécnico de Torino [9]. Además, se han examinado diseños que utilizan recursos de propósito específico (en concreto DSPs) y se han valorado las diferencias en términos de fiabilidad a SEUs cuando, en lugar de DSPs, se utilizan los recursos de propósito general (CLBs y FFs).

Y como última aportación, se han introducido dos técnicas de protección de circuitos: la primera se centra diseños con arquitectura de red neuronal de tipo Hopfield, y se combina con el TMR (sección 4.2). La segunda se basa en el uso mixto de DSPs y TMR (sección 4.3). Ambas han sido validadas con la herramienta, comparando los diseños reforzados con sus diseños equivalentes no reforzados.

Capítulo 2: Trabajo relacionado

La problemática de los SEUs (o bitflips) es conocida desde hace tiempo y ha sido objeto de estudio por distintos grupos de investigación. La utilización de distintos materiales semiconductores, distintas geometrías en la distribución de capas y sustratos en la fabricación de dispositivos o redundancia software en las aplicaciones que éstos ejecutan, son ramas de investigación abiertas para tratar este mismo problema. Sin embargo, esta tesis se centra en abordar el problema en el nivel conocido como transferencia de registro (register-transfer level o RTL) durante el proceso de diseño.

Existen ya varias metodologías y técnicas de diseño que tienen dicho fin, esto es, reforzar los diseños ante los SEUs. A su vez, hay distintas plataformas de inyección de errores que abordan esta problemática y complementan a dichas técnicas al permitir comprobar su validez. Dichas plataformas hacen un estudio similar al propuesto en esta tesis doctoral facilitando el análisis y estudio del impacto de dichos errores, sin embargo, tienen una serie de limitaciones o incovenientes que NESSY pretende solventar.

En este capítulo se detallan las principales técnicas y herramientas de protección de diseños y de inyección de errores existentes en el estado del arte.

2.1. Técnicas de protección de diseños hardware en FPGAs

Una solución directa para aumentar la fiabilidad de diseños hardware en FPGAs consiste en reforzar los dispositivos sobre los que se implementan los diseños que se quieren proteger. En este caso, eso se traduce en reforzar la arquitectura de las FPGAs por diseño, es decir, mejorar la tecnología subyacente utilizada en su fabricación. Otras técnicas menos

inmediatas pero más sofisticadas donde interviene directamente el diseñador hardware, consisten en aplicar algún tipo de redundancia. Esto da lugar a diferentes maneras de abordar el problema:

- Protección por tecnología.
- Redundancia temporal.
- Redundancia de aplicación.
- Redundancia de datos.
- Redundancia hardware.

Todas los conceptos listados, exceptuando la "protección por tecnología", se basan en la redundancia, que se define como "*el empleo de información, recursos o tiempo adicionales, por encima de los estrictamente necesarios para el correcto funcionamiento de un sistema.*" [21]. A continuación, se explican en detalle cada uno de dichos conceptos.

2.1.1. Protección por tecnología

La solución más inmediata a la problemática de los SEEs es la de hacer dispositivos más robustos a la hora de su fabricación. Existen procesos de fabricación alternativos y materiales especiales utilizados exclusivamente para crear dispositivos resistentes a la radiación. Esto es muy común en el sector aeroespacial y de defensa [4].

El problema de estos dispositivos "radhard" es que se limitan a disminuir su probabilidad de fallo frente a radiación, pero no garantizan al 100% que sus diseños vayan a estar libres de SEEs, por lo que por sí solo, un dispositivo protegido por diseño no basta.

2.1.2. Redundancia temporal

La técnica de la redundancia temporal sólo se puede aplicar para evitar un subconjunto de los SEEs y con ciertas limitaciones [10] Consiste en comparar una función

combinacional con su salida un tiempo Δt después. De esta forma, si se ha producido un transitorio (es decir, un SET) en alguna de sus señales, será detectado ya que las salidas serán distintas en t y en t+Δt. Una manera de implementar un circuito que detecte esta situación, es el que se muestra en la Figura 10.

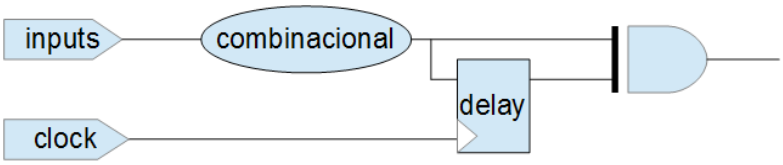

Figura 10: Esquema RTL de redundancia temporal

De esta forma, la salida de este circuito detectará cuando la salida en el instante t y en el instante t+Δt sean distintas. La limitación de este método consiste en que sólo se debe evaluar la salida de error cuando las entradas a la función combinacional hayan permanecido constantes durante al menos un tiempo determinado ya que si no, se podría dar un falso positivo en un cambio de valor legítimo en una entrada.

Además, para poder detectar un SET correctamente, el retardo introducido para la comparación debe ser mayor a al menos, el tiempo que tarda el SET en desaparecer, por lo que la frecuencia máxima de trabajo del circuito, estará limitada por este tiempo.

2.1.3. Redundancia por aplicación

Esta técnica se centra en el nivel de aplicación o software. Consiste en el análisis de datos mediante comprobaciones y duplicaciones, añadiendo instrucciones adicionales al código ejecutado.

No puede detectar ciertos errores hardware, como por ejemplo fallos en el propio procesador, controlador de memoria o lógica de control. Además, esta técnica penaliza el rendimiento del sistema al tener que ejecutar instrucciones adicionales para la comprobación de los datos.

2.1.4. Redundancia de datos

Existen varios métodos que se centran en la fiabilidad frente a SEEs a nivel de datos.

Una técnica muy utilizada es la sobrecarga de datos con códigos de detección de errores como son los códigos Hamming o el método de chequeo de redundancia cíclica (o CRCs por sus siglas en inglés). Esta técnica no protege frente a todo tipo de SEEs, ya que solo tiene en cuenta errores producidos en elementos de memoria, esto es, los SEUs. Además, tampoco se puede aplicar cuando la probabilidad de error en una misma palabra de memoria es alta, ya que si se dan 2 SEUs simultáneos en intervalos de tiempo pequeños los códigos Hamming o CRCs no pueden corregir palabras con más de 1 error.

Sin embargo, existe una técnica sencilla de evitar varios errores en la misma palabra. Dicha técnica conocida como “bit interleaving” consiste en realizar un intercalado de datos en las memorias, es decir, situar bits adyacentes a nivel lógico en posiciones físicamente separadas de la memoria para así evitar MBUs [41].

2.1.5. Redundancia hardware

Este tipo de redundancia consiste en incorporar hardware adicional, normalmente con el fin de detectar fallos o conseguir una alta tolerancia ante los mismos. Es la técnica más frecuentemente utilizada y en la que se apoya esta tesis. Existen multitud de tipos de redundancia hardware, los cuales se explican a continuación.

2.1.5.1. Redundancia pasiva

Esta técnica explota el concepto de “enmascaramiento de los errores”, es decir, ocultar las consecuencias que éstos puedan tener en el correcto funcionamiento de los circuitos. En cualquiera de las múltiples versiones que actualmente existen de esta técnica, no es necesario que se realice ninguna acción externa por parte del sistema que controla el funcionamiento del circuito o del usuario final.

El primer tipo de técnicas de redundancia pasiva que se comentará es la **Redundancia Triple Modular (TMR)**, la cual tiene como concepto básico ejecutar tres copias del mismo circuito, de manera que cada una de ellas realice individualmente todos los cálculos y se genera una única salida en función de la comparación entre las salidas de las tres copias. La Figura 11 muestra un esquema de aplicación de esta técnica. Como se puede observar, las tres salidas son procesadas por un selector de mayoría (o votador) que genera un único resultado (la salida será la entrada del votador que más veces se repita). De esta manera, si cualquiera de las tres copias del circuito falla, el error se puede corregir mediante las otras dos. Esta técnica es tolerante a fallos en solo una de las copias del circuito.

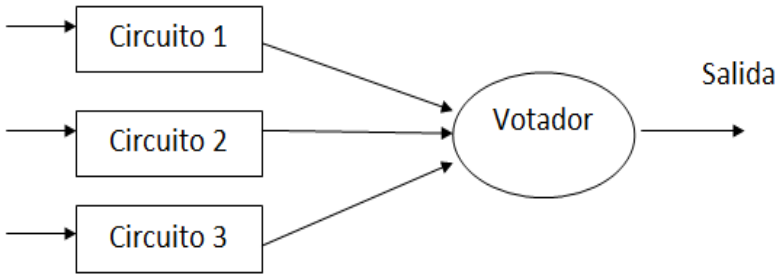

Figura 11: Esquema de implementación de TMR

TMR es una parametrización del método genérico de redundancia N-modular [21] donde N=3. En particular, el TMR es la técnica de redundancia más popular debido al compromiso que ofrece entre fiabilidad y recursos necesarios. Con un N=3 el sistema es capaz de detectar y corregir fallos simples (SEUs), que es el objetivo de esta tesis. El número mínimo de copias necesarias para que esta técnica funcione es 3, puesto que para un N=2, en caso de fallo de una copia redundante, no habría forma de distinguir cuál ha sido la errónea y cuál la correcta y por lo tanto, sólo se podría detectar que ha habido un fallo sin la posibilidad de enmascararlo.

La expresión de fiabilidad de un sistema redundante N-modular es:

$$R_{NMR} = \sum_{i=1}^{n} \left(\frac{N}{i} \cdot (1 - R_M) \cdot R_M{}^{(N-1)}\right)$$

Donde N es el número de módulos redundantes y n es el número de fallos que puede

detectar.

Por lo tanto, en un sistema TMR, la probabilidad de que el conjunto funcione, suponiendo que no falle el sistema de votación, es la suma de las probabilidades de tres módulos funcionando y dos módulos funcionando (ya que, como se ha mencionado anteriormente, sólo es tolerante cuando hay fallo en una de las copias o en ninguna). Es decir:

$$R_{TMR} = R_M^{\,3} + 3R_M^{\,2} \cdot (1 - R_M) \; = \; 3R_M^{\,2} \; - \; 2R_M^{\,3}$$

Las ventajas de la redundancia N-modular son la recuperación autónoma e inmediata de un circuito afectado por un SEU, alta tolerancia a los SEUs y que la conversión de un sistema no redundante a uno redundante es muy sencilla.

Sin embargo, entre las desventajas se puede encontrar que esta técnica es muy costosa, tanto en área como en consumo de energía y que no es tolerante a fallos en el votador. Esto puede corregirse incorporando también redundancia en dicho módulo como se muestra en la Figura 12 [22].

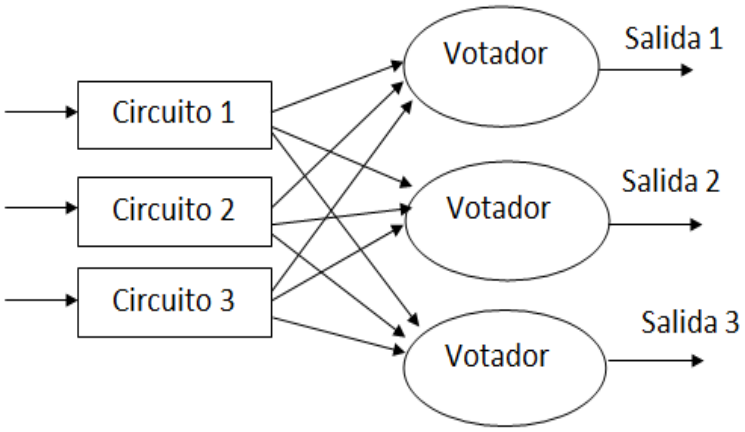

Figura 12: Esquema de implementación de TMR con votador triplicado

El segundo tipo de técnicas de redundancia pasiva es el **TMR selectivo**. Éste es una mejora del TMR para reducir coste en recursos ya que en esta técnica se triplica sólo una parte del circuito en vez de su totalidad.

Mediante esta técnica, el diseñador debe identificar qué región (o regiones) del diseño es

más conveniente proteger, haciendo un estudio previo de las regiones que son más o menos críticas en el sistema, para así proteger en primer lugar las más críticas [34].

La principal ventaja de esta técnica es que tiene menor penalización en área, ya que en el TMR se triplica el circuito entero, mientras que con el TMR selectivo hay dos copias reducidas del circuito y una del circuito original, tal y como se ve en la Figura 13.

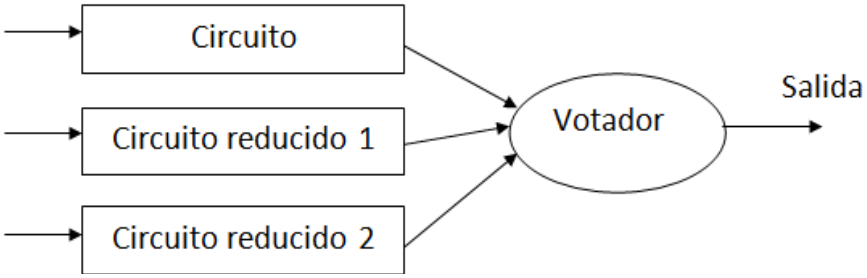

Figura 13: Esquema de implementación de TMR selectivo

En cuanto a las desventajas, esta técnica puede introducir retardos adicionales [23] y, además, quedan desprotegidas las señales de reloj y de reset, así como la lógica interna del dispositivo afectada por los SEFI [6].

2.1.5.2. Redundancia activa

Otra técnica de protección es la redundancia activa o dinámica, en la que se consigue la tolerancia a fallos detectando el error y corrigiéndolo, sustituyendo el módulo erróneo por otro. Esta técnica requiere que el sistema se reconfigure total o parcialmente en tiempo de ejecución para continuar funcionando correctamente. Sin embargo, la principal desventaja es que funcionamiento nominal del circuito tardará un tiempo en establecerse entre la detección del error y la reconfiguración del nuevo modulo, por lo que estos sistemas deben tolerar un funcionamiento erróneo durante un corto período de tiempo. Otro de los defectos de esta técnica es que no detecta los fallos ocurridos en el votador [11].

Un método de redundancia activa, que es capaz de detectar un error pero no de localizarlo, es **la duplicación con comparación** (o DWC por sus siglas en inglés). Un

esquema de esta técnica se muestra en la Figura 14. Su funcionamiento consiste en utilizar la copia del circuito duplicado para comparar las salidas (el de la copia y el del circuito original) para que en caso de que se produzca un error, detectarlo. En este caso se deja la responsabilidad al diseñador de cómo proceder en caso de detección de un error.

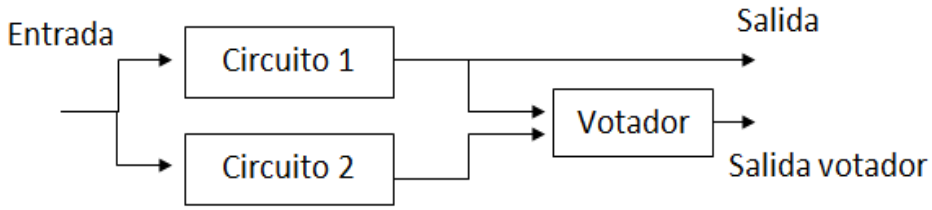

Figura 14: Esquema de implementación de duplicación con comparación

Otro método de redundancia activa son los **repuestos en espera**, que se muestra en la Figura 15. Esta técnica consiste en tener el módulo crítico replicado, aunque sólo uno está en funcionamiento en cada instante y el resto estarán inactivos esperando su activación. Cuando un módulo activo falla, delega la ejecución a un repuesto, por lo que un sistema con N módulos tolera N-1 errores. Los detectores de error seleccionan la entrada activa del multiplexor [14]

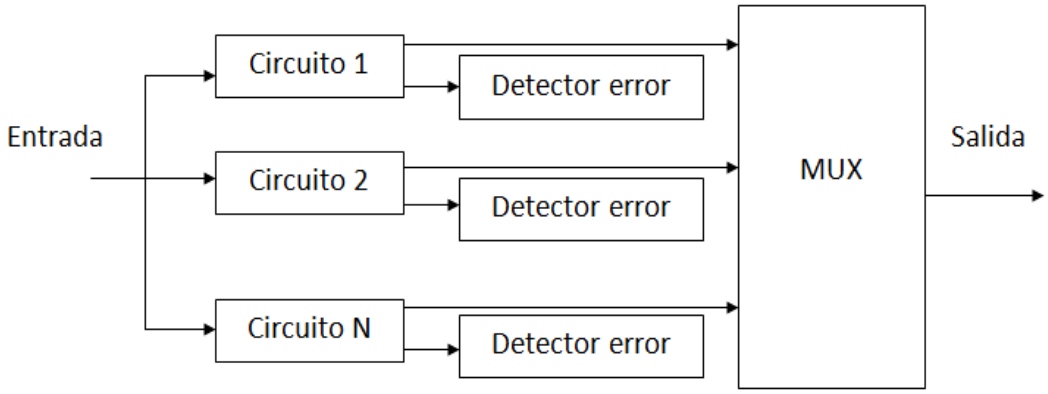

Figura 15: Esquema de implementación de repuestos en espera

La principal ventaja de este método es que un sistema con N módulos idénticos puede proporcionar mucha tolerancia a fallos con un número de repuestos bastante inferior a N, ya que no es necesario tener un repuesto por módulo.

Por otra parte, tiene las desventajas de que no es tolerante a fallos en el multiplexor y que es un sistema más complejo y costoso que las otras variantes presentadas [6].

Esta técnica puede aplicarse de dos maneras:

- En **frío**: los módulos de reserva no están operativos hasta que no son seleccionados por haberse producido algún error en el módulo activo. Esto explica que durante un tiempo, el sistema no está disponible a cambio de minimizar el consumo energético y conservar mejor los módulos en reserva. Para ello, esta técnica se utiliza comúnmente en satélites.
- En **caliente:** los módulos en reserva están continuamente en funcionamiento para así minimizar el tiempo de recuperación del error.

2.1.5.3. Métodos híbridos

Los últimos tipos de técnicas de redundancia que se mencionarán en esta tesis doctoral son los métodos híbridos. En ellos se emplea el enmascaramiento de errores. Sin embargo, el sistema también puede reconfigurarse en caso de fallo.

Existen numerosas implementaciones posibles para esta técnica entre las que se mencionarán la redundancia N-modular con repuestos, la redundancia con autoeliminación, y la arquitectura triple-dúplex [6].

La **redundancia N-modular con repuestos**, ilustrada en la Figura 16, consiste en reforzar la redundancia N-modular con módulos de repuesto adicionales. De este modo, se compara la salida de cada módulo con la salida del votador y, si no coinciden, se deduce que el módulo está fallando y se sustituye por un repuesto. La red de conmutación debe dar la posibilidad de reemplazar cada uno de los módulos por uno de los repuestos.

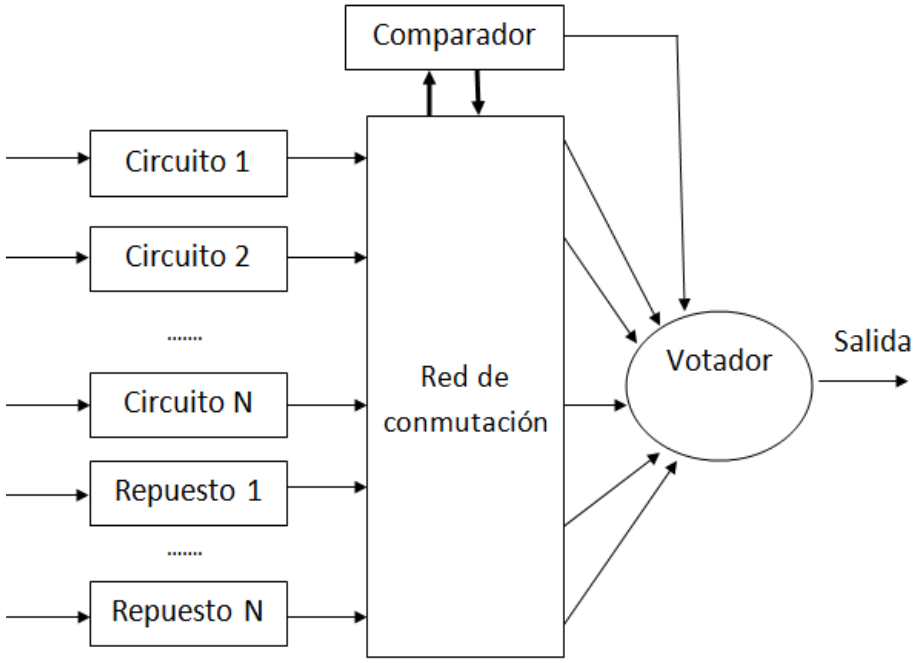

Figura 16: Esquema de implementación de redundancia N-modular con repuestos en espera

En cuanto a la **redundancia con autoeliminación**, consiste en la modificación de la redundancia N-modular para comparar la salida de cada módulo con la salida votada, como se puede observar en la Figura 17. En caso de que difieran, ese módulo se invalidará, por lo que su salida no entrará en el votador.

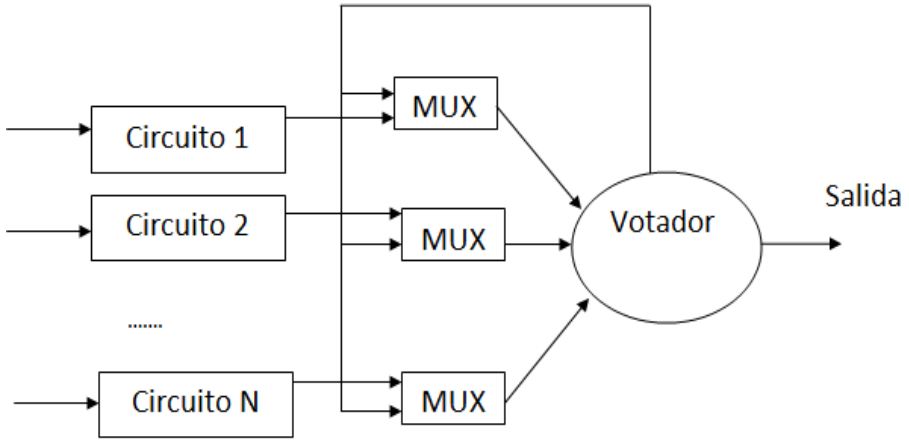

Figura 17: Esquema de implementación de redundancia con autoeliminación

Por último, **la arquitectura triple-dúplex**, mostrada en la Figura 18, consiste en

emplear el método de duplicación con comparación para detectar fallos en los módulos y, cuando hay diferencia en el resultado, se excluye la salida del comparador del votador de la redundancia triple modular.

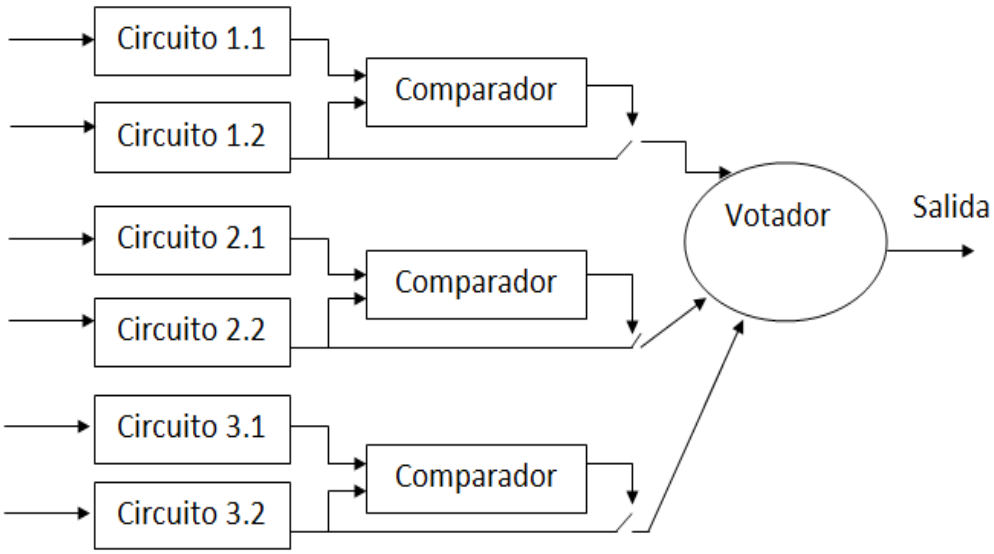

Figura 18: Esquema de implementación de redundancia híbrida triple-duplex

2.2. Técnicas de inyección de errores

La manera más inmediata de testear los efectos de los SEUs en celdas de memoria es testear los diseños bajo radiación real [7], [27], [28]. Sin embargo, debido a su alto coste y alto grado de aleatoriedad [40], este proceso no siempre es viable y por lo tanto, se han investigado una serie de alternativas basadas en la emulación de SEUs en flipflops, cada una con sus ventajas e inconvenientes. Principalmente, existen tres tipos de técnicas de emulación de SEUs:

- Emulación basada en instrumentación.
- Emulación basada en reconfiguración.
- Emulación basada en simulación.

Por otro lado, entre las técnicas de inyección de errores existentes en la literatura, habitualmente también se realiza una distinción entre las técnicas que se utilizan en circuitos cuya tecnología destino son las FPGAs y aquéllas que verifican circuitos que se implementan sobre ASICs. En la práctica totalidad de cualquier diseño hardware, es necesario hacer uso de biestables para registrar señales y mantener un sincronismo con la señal del reloj del sistema, característica necesaria en diseños según aumenta su complejidad. Un biestable es un elemento de memoria igual de vulnerable a los efectos de la radiación que la memoria de configuración. Así, si el circuito va a ser implementado en una FPGA, el sistema de inyección debe poder modificar el contenido tanto de los flipflops del propio diseño, como del de la memoria de configuración de la FPGA, mientras que si la tecnología objetivo es un circuito tipo ASIC, el sistema de inyección se limitará a inyectar errores únicamente en los flipflops. Esto da lugar a dos tipos de sistemas en función de cuál es la tecnología destino del circuito que se quiere evaluar:

- Sistemas implementados en una FPGA.
- Sistemas implementados como un ASIC.

Mientras que para inyectar errores en la memoria de configuración de los diseños implementados en FPGAs se debe hacer uso de la emulación basada en reconfiguración, para inyectar errores en los flipflops (ya sea la tecnología destino una FPGA o un circuito ASIC) se puede optar por cualquiera de las tres técnicas mencionadas.

2.2.1. Inyección de errores para sistemas implementados en FPGA

Si un sistema se implementa sobre una FPGA, como se ha explicado, se distinguen dos grupos de elementos de memoria según el tipo de elemento sobre el que se quiera inyectar el error. Esta distinción se debe a la sustancial diferencia que existe entre la tecnología utilizada para el diseño de la memoria de configuración del dispositivo y para las celdas de

memoria propias del diseño (flipflops). En las FPGAs basadas en SRAM de Xilinx, la totalidad de la memoria de configuración y de datos (BRAM) es accesible mediante varios mecanismos, como por ejemplo la interfaz SelectMap o JTAG, mientras que los flipflops, que son celdas de memoria internas a los CLBs, son de difícil acceso. Esta diferencia se traduce en que las inyecciones en la memoria de configuración, como norma general, serán más rápidas y menos intrusivas que las inyecciones en los flipflops.

Por otro lado, como se ha mostrado en la Tabla I del capítulo 1, la memoria de configuración es la que más elementos de memoria contiene en una FPGA. Por lo tanto, dicha memoria cobra gran importancia cuando se quiere evaluar la vulnerabilidad frente a SEUs de un circuito implementado en una FPGA. La emulación de un SEU en la memoria de configuración implica que se modificarán los bits asociados a la configuración de los diferentes recursos lógicos existentes en el dispositivo, afectando a su comportamiento final. La estructura interna de dicha memoria y la relación entre los recursos de las FPGAs Virtex y los bits de configuración, se explican en detalle en el Apéndice I.

Otros grupos de investigación han desarrollado herramientas capaces de inyectar errores en la memoria de configuración de las FPGAs. Los más relevantes se exponen a continuación, donde se destacan sus principales ventajas e inconvenientes y se explica su método de funcionamiento.

2.2.1.1. FLIPPER

La herramienta FLIPPER es una herramienta de inyección de errores para FPGAs basadas en SRAM desarrollada por el Instituto Nacional de Astrofísica de Milán [29] [3] [2]. Esta herramienta utiliza dos FPGAs: una de control y otra donde instanciar el diseño bajo test.

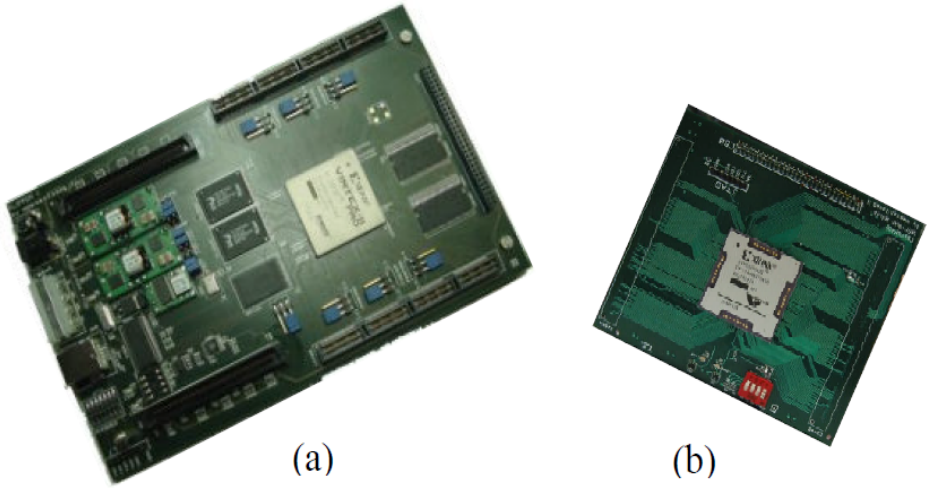

Figura 19: Plataforma FLIPPER

En la Figura 19, tomada de [3], se muestran las dos tarjetas que componen el sistema FLIPPER. La tarjeta (a) es la de control y la que gestiona la inyección de errores. Contiene una FPGA Virtex II, memoria SDRAM y un controlador USB para la comunicación con el PC. La tarjeta (b) contiene la FPGA donde irá el diseño bajo test y el rutado necesario para que el pinout de ésta sea accesible desde la tarjeta (a). La inyección de errores se realiza mediante la modificación del archivo de configuración de la FPGA destino en la tarjeta (b). Al tratarse de un sistema de inyección de errores para sistemas implementados en FPGAs, se basa en la reconfiguración de éstas. Por cada SEU a emular, la FPGA de control utiliza el fichero de configuración generado por las herramientas de Xilinx, para invertir el valor del bit deseado, recalcular el nuevo CRC del fichero y enviarlo a la FPGA destino en la tarjeta (b). La FPGA de control se encarga de enviar las entradas a la tarjeta (b), leer las salidas de ésta y así, cuando se detecta un error, la tarjeta de control enviará un paquete al PC con la información obtenida sobre dicho error. Explicada la arquitectura de la herramienta, el proceso de inyección de errores que realiza FLIPPER consiste en:

1. **Selección del archivo de configuración**. Este es el archivo de configuración de la FPGA destino que define el diseño que allí se configurará.
2. **Generación de patrones de test.** Estos patrones serán los que estimulen las entradas del circuito bajo test. Las salidas generadas por el circuito debido a estos estímulos, serán examinadas para la detección de errores.

3. **Selección del objetivo de la inyección**. En este punto existen dos opciones: inyectar errores sólo sobre los datos del archivo de configuración de la FPGA o inyectar errores en los metadatos de dicha configuración. Los metadatos contienen la información para que el controlador de la FPGA pueda reconfigurarse correctamente, pero no están relacionados con el funcionamiento del circuito que se quiere implementar.
4. **Selección del modo de inyección**. En esta plataforma existen dos formas de generar el error: modo aleatorio y modo secuencial. El modo aleatorio, como su nombre indica, inyecta un error en un bit aleatorio del mapa de bits, asemejándose más a como se comporta la radiación real, mientras que el modo secuencial lo hace siguiendo un orden para facilitar el posterior análisis de los resultados y permitir que los experimentos sean reproducibles.
5. **Adquisición de datos.** Estos datos son resultado de la inyección de errores.
6. **Verificación funcional del diseño**. Esto ocurre cada vez que la FPGA destino completa el proceso de configuración, para así detectar si ha habido errores de configuración debido a errores inyectados en los metadatos del archivo de configuración o no, en el caso de que se haya elegido inyectar errores sobre éstos.
7. **Almacenamiento de datos**. Los datos generados a raíz de la inyección de errores se escriben en el disco duro del PC para un futuro análisis.

Esta plataforma ha sido validada bajo experimentación con un acelerador de protones. Sin embargo, tiene algunas limitaciones:

- **Alto coste**: requiere dos FPGAs además de dos tarjetas ad-hoc.
- **Es intrusiva**: aún estando el diseño bajo test en una FPGA independiente, la comunicación entre ambas se realiza mediante unos pines fijos obligando al diseño bajo test a usar los restantes. Esto ocasiona cambios en el rutado y posicionamiento con respecto a la implementación real del circuito bajo test.
- **Portabilidad**: debido a estas conexiones fijas, no todas las tarjetas con la misma FPGA son compatibles con este sistema.

2.2.1.2. FT-UNSHADES

FT-UNSHADES fue inicialmente desarrollado para la simulación de SEUs en diseños VLSI y presenta dos modos de inyección de errores: en flipflops y en la memoria de datos. En [19], los autores de esta herramienta presentaron una extensión de la plataforma, FT-UNSHADES-2, la cual también emula SEUs en la memoria de configuración para FPGAs Virtex II.

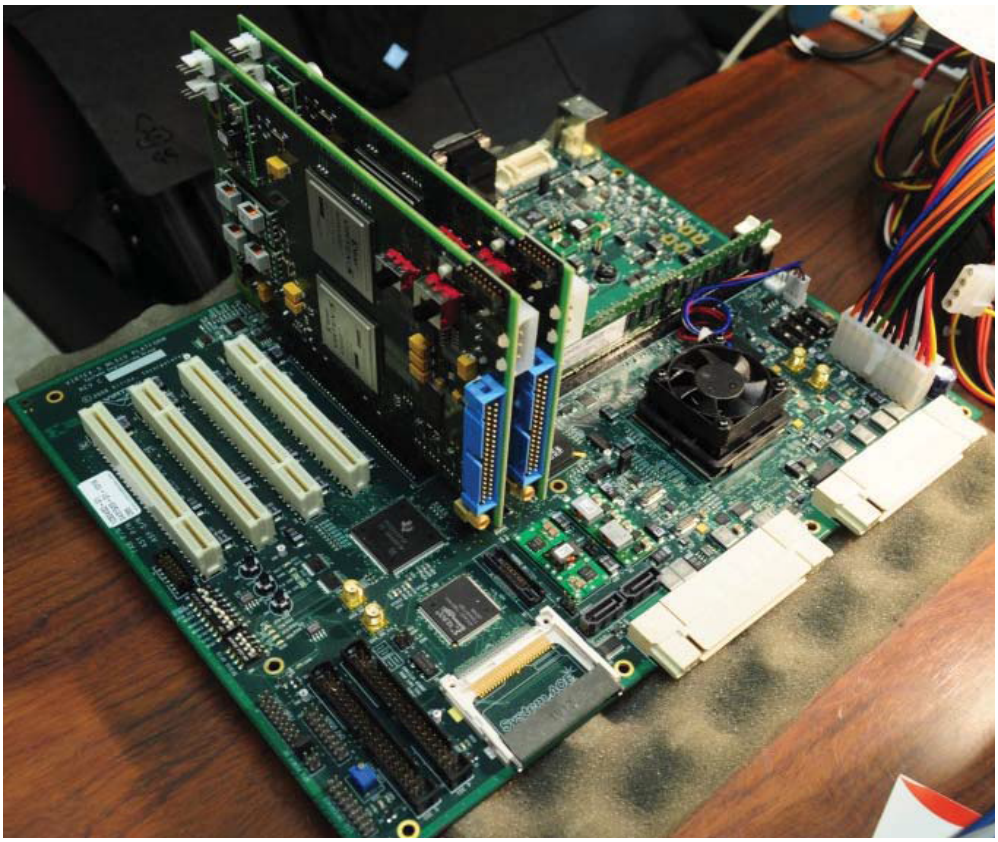

Figura 20: Plataforma FT-UNSHADES-2

La arquitectura de este sistema contiene un total de cinco FPGAs: por un lado existe una tarjeta ad-hoc principal que contiene a la FPGA de control y es la encargada de gestionar el funcionamiento de toda la plataforma. Por otro lado, existen otras dos tarjetas satélite ad-hoc, y cada una de ellas tiene dos FPGAs. En cada tarjeta satélite, una FPGA (FPGA objetivo) es la encargada de configurar el diseño bajo test mientras que la otra (FPGA de servicio) hace de interfaz con la FPGA principal mediante un bus PCI express. En la Figura

20, tomada de [19], se aprecia la tarjeta principal así como las dos tarjetas satélites.

La FPGA de servicio hace de serializador y deserializador de los datos que le llegan en serie desde la FPGA de control y los introduce como estímulos a la FPGA objetivo. La FPGA objetivo es la encargada de configurar el circuito que le indique la FPGA de control. Existen dos tarjetas satélite para poder inyectar errores en ambas FPGA objetivo de la tarjeta 1 y comparar las salidas de ésta con las salidas de la FPGA objetivo de la tarjeta 2.

La herramienta realiza campañas de inyección de errores de la siguiente manera:

1. **Configuración de la campaña de inyección de errores**. Esto lo realiza el usuario a través de la interfaz mediante comandos que le ofrece el PC. Se indican, entre otros, el diseño a testear, el vector de test que se pondrá a las entrada del diseño bajo test y parámetros como el número de errores a inyectar o en qué ciclo de reloj se inyectarán dichos errores.
2. **Configuración inicial del sistema**. La FPGA de control configura las dos FPGAs objetivo (valiéndose de las FPGAs de servicio) a través de la interfaz SelectMAP. A partir de este momento, la campaña de inyección puede comenzar.
3. **Inyección de errores**. Para cada bit sobre el que se inyecta un error, el reloj generará tantos pulsos como se haya indicado en el paso 1. A continuación, el reloj se detiene y la FPGA de control, mediante reconfiguración parcial, modifica el archivo de configuración de una de las dos FPGAs destino. Finalmente se restaura la generación del reloj y la ejecución. Para la detección de errores se comparan las salidas de ambas FPGAs objetivo.

Una vez la inyección de errores ha finalizado, la herramienta ofrece una serie de modos de análisis para estudiar los resultados. En este análisis se puede observar la propagación de los errores inyectados a nivel de registro ciclo a ciclo, e incluso detectar errores que no se han propagado a las salidas del diseño pero que han modificado el estado interno del diseño bajo test. Sin embargo, esta plataforma presenta las siguientes limitaciones:

- **Cobertura parcial**: la inyección de errores se hace sólo en algunos bits de la memoria de configuración; esto es, en los que afectan el contenido de elementos secuenciales (BRAM, flipflops, registros etc). Sin embargo, no se inyecta sobre bits

asociados al rutado o a las LUTs.

- **Alto coste**: requiere 5 FPGAs y la fabricación y montaje de 3 PCBs.

2.2.1.3. Sistema del grupo CAD de la Politécnica de Torino

El grupo CAD de la Universidad Politécnica de Torino también ha desarrollado una herramienta de inyección de errores en circuitos implementados en FPGA. Dicha herramienta ([5] [42]) está basada en FPGAs Xilinx Virtex II e inyecta errores únicamente en la memoria de configuración.

Por un lado, en [42] se presenta una primera versión de la arquitectura del sistema, el cual está basado en microprocesador. Se compone de un Power PC, memoria embebida y un controlador del puerto ICAP para realizar las inyecciones de errores. De esta manera, este sistema alcanza un rendimiento muy alto, ya que para la inyección de un error, se requiere modificar la menor cantidad de información direccionable por la arquitectura Virtex II y esta inyección se realiza internamente en la FPGA mediante el manejo del puerto ICAP, lo que optimiza el tiempo necesario para realizar la inyección del error. Sin embargo, esta primera versión es muy intrusiva puesto que el rutado y posicionamiento del hardware de control y el dispositivo bajo test se realiza sin ninguna restricción.

La otra propuesta [5], mejora la anterior plataforma en términos de intrusividad, al aislar el diseño bajo test en una FPGA, mientras la inyección de errores se realiza desde un PC a través del puerto JTAG. La campaña de inyección de errores para esta nueva herramienta, comprende los siguientes pasos:

1. **Inyección de error en la memoria de configuración.** El PC, que contiene el archivo de configuración de la FPGA, debe modificar una copia de éste donde invierte el bit donde se quiere forzar el error. Una vez hecho esto, a través del puerto JTAG se reconfigura la mínima cantidad de información direccionable por el sistema de configuración de la FPGA, inyectando el error en un tiempo óptimo.
2. **Aplicación de patrones de test a las entradas.** Esto se realiza mediante el uso de instrucciones de la interfaz JTAG. De la misma forma, las salidas del circuito se leen

a través de esta misma interfaz y se almacenadan en el PC para su posterior análisis.

3. **Corrección de fallo.** En este paso, la herramienta ofrece varias alternativas: corregir el fallo inmediatamente después de cada evaluación de error (para así emular SEUs), restaurar el error después de un determinado número de errores (para así emular un multiple bit upset o MBU) o restaurar el estado del sistema cada cierto periodo de tiempo para emular una técnica llamada scrubbing.

Estas dos arquitecturas tienen una serie de ventajas y desventajas entre sí, así como limitaciones a la hora de inyectar errores. La propuesta de [42], tiene como desventaja que el rendimiento se ve disminuido con respecto a la primera versión [5] ya que se realiza la inyección del error a través del puerto JTAG, externo a la FPGA y por lo tanto más lento que cuando el error se gestiona internamente en la FPGA. Por otro lado, la primera versión es una herramienta intrusiva que afecta al rutado del circuito bajo test, mientras que la segunda aproximación no lo es. En ambas versiones de la herramienta, la inyección del error es solo posible en la memoria de configuración, y no en los flipflops del propio circuito.

2.2.2. Inyección de errores en sistemas implementados como un ASIC

Los circuitos que se implementan como un ASIC, tienen como únicos elementos de memoria los flipflops del propio diseño y la memoria de datos. Esto implica que las alternativas para emular un SEU sean mayores, ya que, al no existir memoria de configuración, no es necesario utilizar una herramienta de inyección de errores basada en reconfiguración. Esto permite utilizar las tres formas de emulación mencionadas al principio de la sección 2.2. :

- La **emulación basada en instrumentación**, que consiste en añadir, para cada elemento de memoria del circuito, un hardware que ofrezca un mecanismo adicional de lectura y de escritura sobre su contenido, sin afectar al funcionamiento nominal del circuito.

- La **emulación basada en reconfiguración** se apoya en la flexibilidad que ofrecen los dispositivos reconfigurables para modificar su memoria de configuración.
- Por último, la **emulación basada en simulación** obtiene un modelo equivalente al diseño bajo test y mediante simulación, se introducen los errores en las celdas de memoria en los instantes de tiempo deseados.

A continuación se introducen las distintas herramientas de inyección de errores sobre sistemas implementados en ASIC existentes y que son más relevantes en el estado del arte. Se agrupan según el tipo de emulación en la que se basan.

2.2.2.1. Emulación basada en instrumentación

Las herramientas que utilizan técnicas basadas en instrumentación requieren añadir bloques lógicos adicionales, conocidos como "saboteadores", por cada recurso donde haya que inyectar un error. Una vez estos saboteadores han sido añadidos al circuito bajo test, no suponen una penalización temporal para la ejecución del mismo. Sin embargo, sí que suponen una penalización en área debido a los recursos adicionales a añadir, lo cual puede ser un factor crítico en circuitos de gran tamaño.

Cualquier diseño HDL (tanto si se quiere utilizar para simulación o para implementación en una FPGA) se debe traducir a una netlist. Dichas netlist son ficheros que contienen la información que define cómo se conectan los nodos de un circuito. Existen distintos formatos de netlist según vayan a ser utilizadas por las distintas herramientas de simulación o síntesis de archivos de configuración de FPGAs. Así, un circuito queda definido mediante su netlist al describir todos sus componentes internos y cómo se conectan entre sí. La implementación de los saboteadores se puede realizar modificando la netlist del diseño o directamente utilizando herramientas comerciales desarrolladas para tal fin. A continuación se detallan algunos de los métodos de emulación basada en instrumentación más destacados en la literatura:

- El método presentado en [30] utiliza herramientas comerciales y la interfaz JTAG

para la inyección de un bitflip, lo cual hace que se necesite entre 1,79s y 3,73s por inyección de bitflip para los experimentos realizados por los autores.

- Los autores de [43] presentan otra alternativa llamada NETFI (NETlist Fault Injection). Esta herramienta modifica el código RTL de un procesador añadiendo hardware adicional sobre los registros sensibles a errores. Esta alternativa se basa en la modificación de librerías proporcionadas por Xilinx [15]. Añade hardware adicional para cada flipflop, además de un controlador que gestiona la inyección de errores en todo el circuito. Dicho controlador necesita *m*n* bits de memoria, donde *m* es el número de vectores que compone el testbench y *n* es el número de flipflops que tiene el diseño. Por lo tanto, esta alternativa escala mal con respecto al tamaño del diseño y la complejidad de éste.

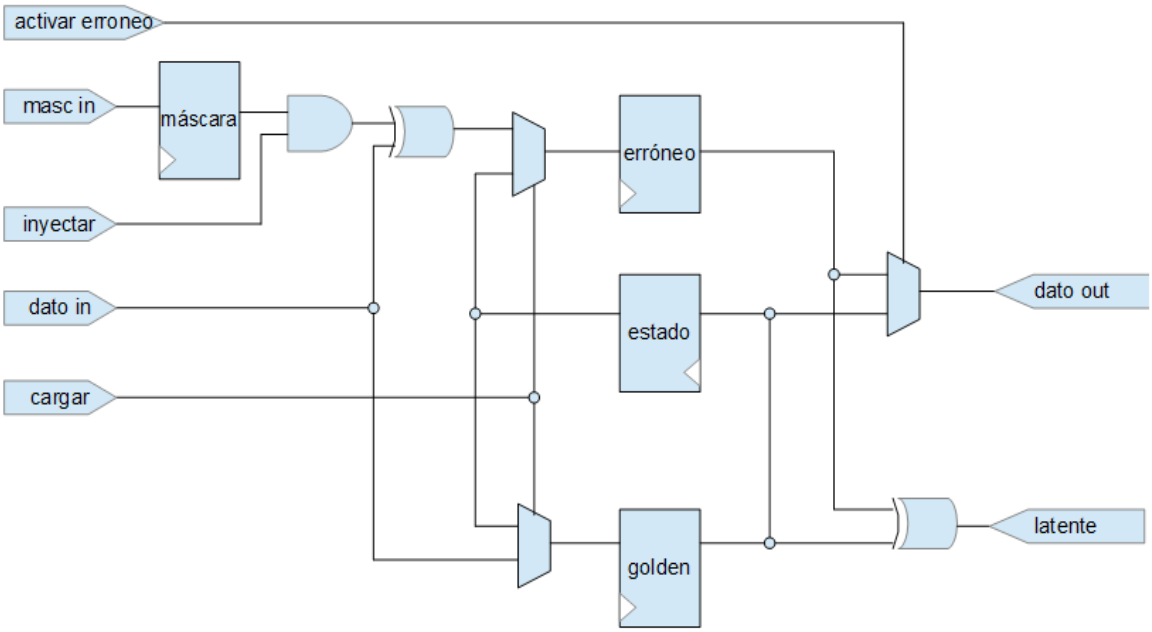

Figura 21: Estructura de multiplexación en tiempo

- Finalmente en [8] se presentan tres métodos diferentes de emulación de errores llamados: multiplexado en tiempo (*time-multiplexed*), escaneo de estados (*state-scan*) y escaneo por máscara (*mask-scan*), que ofrecen diferentes compromisos entre consumo de recursos y rendimiento. En el mejor de los casos, estas técnicas

consiguen inyectar un error en un tiempo del orden de unos pocos µs e introducen una sobrecarga de recursos que no es despreciable.

1. La técnica de multiplexación en tiempo se basa en la sustitución de cada flipflop por el hardware de instrumentación de la Figura 21. En primer lugar, este hardware duplica el flipflop original dando lugar a dos nuevos caminos para los datos (erróneo y golden). El flipflop golden se comportará como en el circuito original, mientras que el erróneo será el del circuito donde se inyectarán errores. Por otro lado, existe otro flipflop de máscara, que cuando se quiera inyectar un error sobre un flipflop determinado, su valor deberá ser '1' y esto permitará inyectar un fallo cuando se desee mediante la señal de control "inyectar". Estos flipflops de máscara forman una cadena entre ellos para ir propagando el flag de qué flipflop es el seleccionado e inyectarle un error. Por último, existe otro flipflop de estado que guarda el estado correcto del circuito antes de que éste sea manipulado. Este flipflop será utilizado por los dos flipflops del circuito (erróneo y golden) para restaurar su valor mediante la señal de control "cargar", una vez el tratamiento del error haya terminado. Los dos flipflops del circuito comparten la lógica combinacional para ahorrar recursos y trabajan en ciclos de reloj alternativos, realizando la multiplexación en tiempo de donde la técnica toma su nombre. Al dato de la salida se le pueden conectar ambos flipflops, tanto el erróneo como el golden, mediante la señal de control "activar erróneo". El error es detectado en la señal "latente" mediante la comparación del contenido de los flipflops erróneo y de golden.

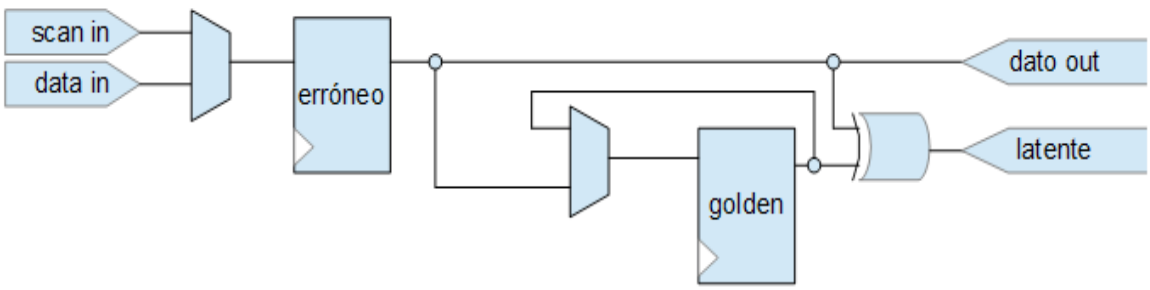

Figura 22: Estructura escaneo de estados

2. La técnica de escaneo de estados (Figura 22) es una simplificación de la anterior, donde se reduce el tamaño del hardware de instrumentación. En este caso los flipflops del circuito se conectan en paralelo tanto a la lógica del resto del diseño como a unos bits de máscara que están almacenados en memoria RAM. Con esto se consigue que se pueda modificar en cualquier momento el estado completo del circuito a través de una señal externa al circuito que controla el multiplexor conectado a la entrada del flipflop "erróneo". Existe también otro flipflop donde se almacena el estado del circuito previo a la inyección. De esta forma, después de cada inyección de error, se compara el contenido de los dos flipflops (erróneo y golden) para verificar si la inyección tiene efecto en la salida o no. La técnica restaura el estado del diseño mediante un escaneo en cadena de los flipflopos por lo que se necesitan *n* ciclos para cambiar el estado del diseño, donde *n* es el número de flipflops que hay en el circuito.
3. La última técnica, escaneo por máscara (Figura 23), es otra simplificación de la multiplexación por tiempo. En esta versión se añade un flipflop adicional (máscara) por cada flipflop del diseño (erróneo), más dos puertas lógicas. Los flipflops de máscara de todo el diseño forman una cadena de manera similar a como se hace en la técnica de multiplexación en tiempo. En esta técnica, el testbench se ejecuta siempre desde el principio por cada inyección de error, ya que no hay lógica que almacene el estado original del diseño (esto es, previamente a inyectar el error) y del que partir después de la inyección de un error, ya que no se puede de restaurar el estado anterior.

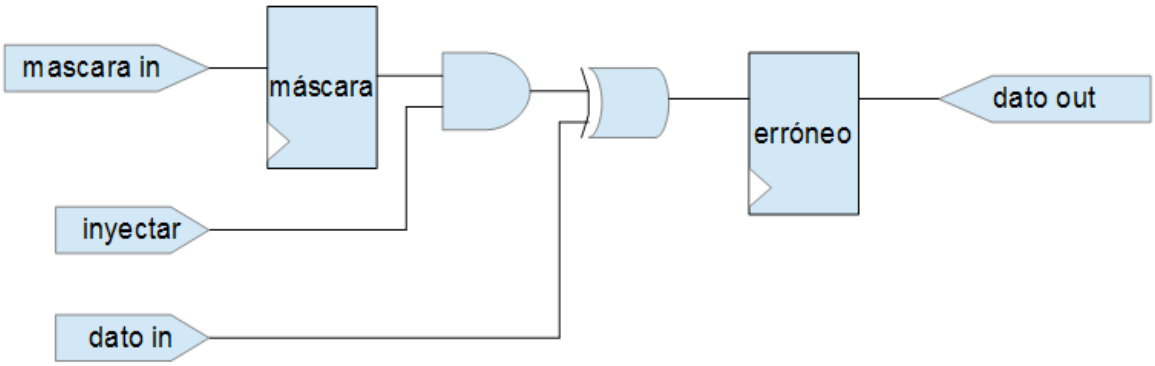

Figura 23: Estructura escaneo por máscara

2.2.2.2. Emulación basada en reconfiguración

Las inyecciones de errores basadas en reconfiguración utilizan la característica de las FPGAs de poder ser reconfiguradas, para modificar su memoria de configuración con el fin de alterar a su comportamiento. En estas técnicas, el cuello de botella viene dado por la velocidad de inyección del SEU, mientras que en otras, es la sobrecarga de área requerida, como ocurre en las técnicas basadas en instrumentación. En las técnicas basadas en reconfiguración, se puede utilizar tanto la reconfiguración global del dispositivo como la reconfiguración parcial.

Las técnicas basadas en reconfiguración global hacen uso de la señal global del dispositivo reset/set (GSR) para lograr su cometido. Esta señal resetea o setea todos los flipflops del dispositivo según su polaridad. El comportamiento de dicha polaridad viene dado por unos bits de configuración llamados INT0/INT1. De este modo, una inyección de errores consta de los siguientes pasos:

1. La lectura de todos los flipflops de la FPGA [11]. En el caso de las Xilinx Virtex-5, esta operación se realiza mediante el método conocido como readback&capture [47]
2. Modificar los bits INT0/INT1 en los flipflops. El valor a escribir dependerá del valor de su estado actual y si se quiere inyectar un error o no en ese flipflop.

3. Activación de la señal GSR.
4. Restauración de la memoria de configuración a su estado original.

Este método es muy lento ya que para realizar la emulación de un SEU, se requiere recorrer todos los flipflops de la FPGA, al ser necesario conocer previamente el estado de todos los flipflops y realizar las modificaciones necesarias a los bits INT0/INT1 de cada flipflop antes de activar la señal de GSR. En el trabajo presentado en [38], investigado por el laboratorio TIMA (Grenoble, Francia), se implementa esta técnica para FPGAs de la familia Virtex de Xilinx. Por otro lado, FTUNSHADES2, la herramienta de inyección ya introducida en la sección anterior, tiene un modo de funcionamiento, "ASIC mode", que también permite realizar inyección de errores sobre elementos de memoria del diseño mediante reconfiguración. Para ello, la netlist equivalente del circuito bajo test, se implementa en una FPGA y la herramienta realiza modificaciones a su memoria de configuración para alterar el valor de los flipflops del diseño.

Este algoritmo tiene una complejidad en el tiempo cuadrática, $\Theta(n^2)$, en función del tamaño del circuito que se debe examinar, ya que, para cada flipflop sobre el que se quiera inyectar un error, se debe leer el estado de todos los flipflops del circuito y este proceso a su vez, debe repetirse tantas veces como flipflops tenga el circuito .

2.2.2.3. Emulación basada en simulación

La emulación basada en simulación ofrece una gran flexibilidad a la hora de modelar un SEU, ya que permite implementar modelos de SEUs más fieles a la realidad que los otros dos métodos de emulación. Las emulaciones basadas en simulación encontradas en la literatura [26] [24] [17] [25] se centran en un nivel de abstracción bajo y estudian los efectos de los SEUs a nivel de la capa física o de dispositivo. Esto supone que la simulación de un SEU requiere mayor carga computacional que las otras dos alternativas, por lo que no se recomienda su uso en la inyección de errores en circuitos algo más complejos al existir otros

métodos de emulación más rápidos. Por lo tanto, esta metodología se centra en circuitos pequeños, típicamente unas pocas puertas lógicas o flipflops.

El objetivo de las herramientas utilizadas en estos experimentos (simuladores basados en lenguaje SPICE), es el de representar el comportamiento físico del circuito emulado en lugar del comportamiento lógico. Por este motivo requieren más memoria y potencia de cálculo. En cualquier caso, este tipo de emulación se aleja del objetivo de esta tesis al pertenecer a un ámbito de aplicación completamente diferente.

Capítulo 3: Metodologías desarrolladas para inyección de errores en FPGAs

Como ya se ha mencionado en la sección 1.3, la contribución de esta tesis doctoral ha sido el desarrollo de una herramienta que inyecta errores en circuitos de manera selectiva y determinista. Dicha herramienta, a la que hemos llamado NESSY, tiene dos modos de funcionamiento que determinan en qué elementos se van a inyectar errores:

1. El primer modo es la inyección en la memoria de configuración y es aplicable cuando el diseño bajo test se implementará en una FPGA basada en SRAM. La inyección de errores en este caso cubre la memoria de configuración de todo el diseño. Para las inyecciones de errores en este modo, NESSY permite clasificar el error según qué tipo de bit ha sido afectado. Esto es posible gracias a un profundo conocimiento de la arquitectura de la memoria de configuración de la familia Virtex, distinguiendo el error según el bit pertenezca a:
    - Bit asociado a la matriz de conexionado.
    - Bit asociado a la lógica.
    - Bit no usado por el diseño pero que está dentro de la región de la FPGA reservada para éste.
2. El segundo modo inyecta errores en los flipflops del circuito y éste se puede aplicar a todo tipo de diseño, tanto a cuando el diseño se implementa en FPGA (y se complementan los resultados con la inyección de errores del modo 1.), como cuando el diseño se implementa en un ASIC. En este último caso, los resultados se limitan a los elementos de memoria del circuito en sí (modo 2.) y no son complementables con los del modo 1. al no existir memoria de configuración.

Para ambos métodos, es necesario un hardware de control adicional al del circuito que se

quiera testear. NESSY implementa este hardware en la misma FPGA que el circuito bajo test y para garantizar la no intrusividad de la herramienta, dicho hardware se instanciará en una región aislada e independiente, haciendo que el rutado del circuito bajo test no se vea afectado por el resto del hardware de control de NESSY y, de esta manera, no afectar a los resultados de la inyección. El motivo por el que NESSY implementa estos componentes en regiones independientes, es que se asume que las tareas se ejecutan en una en una arquitectura multitarea como la que se muestra en Figura 24.

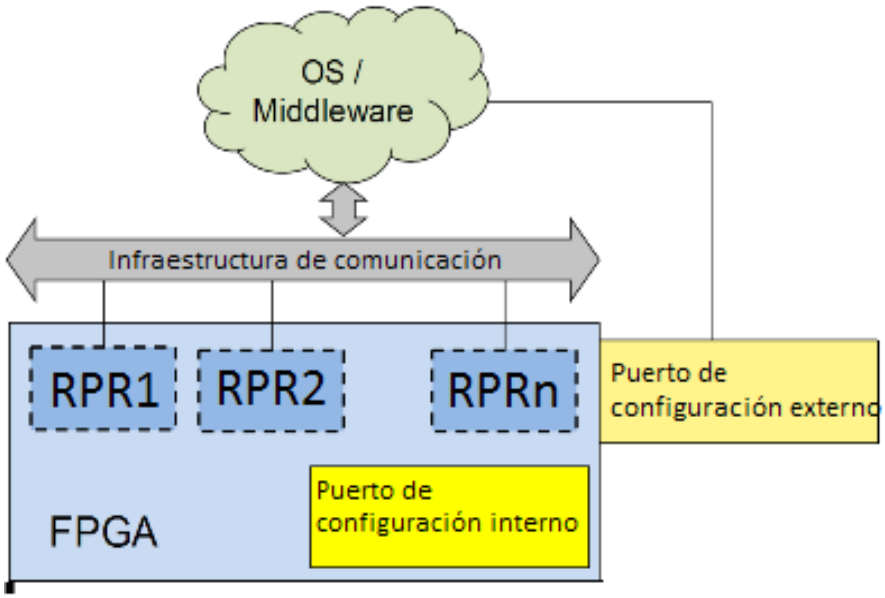

Figura 24: Arquitectura multitarea en FPGAs en NESSY

De este modo, la FPGA destino contiene un conjunto de Regiones Parcialmente Reconfigurables (RPRs). Estas RPRs se conectan a una infraestructura de comunicaciones que puede implementarse como un bus o una Network-on-Chip (NoC). Cada tarea que se ejecuta se coloca en una de estas RPRs, y se comunica con el resto del sistema a través de dicha infraestructura de comunicaciones. El control de las distintas tareas (configuración, entrada/salida, etc) se lleva a cabo mediante un sistema operativo o Middleware, que puede ejecutarse en un procesador situado dentro o fuera de la propia FPGA, y que también está conectado a la infraestructura de comunicaciones. La entrada/salida de tareas se realiza mediante reconfiguración parcial de las RPRs a través de uno de los puertos de

reconfiguración externos (JTAG, BPI, etc) o interno (ICAP).

Los dos modos de inyección desarrollados utilizan un fichero **testbench** y un fichero **golden** para la detección de errores. El testbench es un fichero que comprende un conjunto de vectores de valores que serán puestos en la entrada del circuito bajo test. La ejecución de estos vectores de valores genera unas determinadas salidas. Si el circuito bajo test no ha sufrido ninguna modificación, es decir, es el circuito **original**, a este conjunto de vectores de salida se les llama **golden,** los cuales también son almacenados en forma de fichero. Para conocer si la inyección de un error tiene consecuencias en el funcionamiento del circuito, después de cada inyección se ejecuta el testbench y se comparan las salidas del circuito con las del golden.

Para modificar el comportamiento del circuito, o lo que es lo mismo, inyectar un error, se modifica el fichero **bitstream**. Un bitstream es un archivo binario dependiente de la arquitectura y fabricante de la FPGA destino que sirve para programar dicho dispositivo. Este archivo contiene la información que será escrita en la memoria de configuración y que indica cómo se deben configurar los componentes internos de la FPGA para implementar la lógica del circuito que el usuario proporciona a NESSY a través de un código HDL de alto nivel. Cuando se habla de fichero bitstream, se deben distinguir dos niveles o ámbitos:

- El **bitstream global** es el binario generado por defecto por las herramientas comerciales y se utiliza para programar el comportamiento de toda la FPGA. Por este motivo, el contenido de este fichero se escribe en la memoria de configuración a través de un controlador externo a la FPGA.
- El **bitstream parcial** contiene información de sólo una parte de la FPGA como su propio nombre indica. Al contener menor información, es un fichero más pequeño y por lo tanto más rápido de procesar. Además, parte de la FPGA puede estar configurada y funcionando y ser ésta la que escriba el fichero en la memoria de configuración a través del puerto de configuración interno.

NESSY utiliza el bitstream global una única vez para configurar el hardware de control en el arranque del sistema y bitstreams parciales para las inyecciones de errores porque requieren menos tiempo de procesamiento.

Gracias a estas características, NESSY tiene tiene una serie de ventajas que lo hacen

interesante respecto a otras herramientas de emulación de SEUs del estado del arte:

• **Bajo coste en recursos**: utiliza una sola tarjeta comercial con una FPGA y un único PC para la interacción con el usuario. Esta es una diferencia significativa con respecto a otros sistemas propuestos en la literatura [3] [31]. Por ejemplo, FTUNSHADES2 necesita de hasta cinco FPGAs y tres tarjetas propias o, en el caso de FLIPPER, dos tarjetas propias y dos FPGAs.

• **Poca intrusividad de la plataforma**: en la inyección sobre memoria de configuración, la plataforma debe ser transparente para el diseño bajo test; es decir, el usuario puede decidir en qué región de la FPGA colocar y rutar el circuito bajo test. De esta manera, el diseño original será el implementado en una RPR y no es necesario adaptarlo añadiendo ningún hardware de control. Por tanto, esta herramienta sólo inyecta bitflips en esta región específica para evaluar el impacto de los bitflips sólo en el circuito que se va a testear de manera realista. Para el caso de la inyección en flipflops sí se utiliza instrumentación, pero hay que tener en cuenta que si la tecnología objetivo no es una FPGA, esto no afecta al comportamiento del circuito y por lo tanto, los resultados son también realistas.

• **Eficiencia**: es más eficiente que otros sistemas presentados en la literatura [5], [31]. De hecho, inyectar un bitflip con NESSY requiere 92 μs, lo que es al menos dos órdenes de magnitud más rápido que con cualquiera de estos otros sistemas.

• **Portabilidad y escalabilidad**: NESSY se ha implementado en una placa de prototipado XUPV505-LX110T, que contiene una Virtex-5. Sin embargo, este sistema es adaptable a cualquier FPGA de Xilinx, ya que sólo necesita el puerto ICAP (propio de Xilinx) y una interfaz serie RS232, la cual es un estándar ampliamente utilizado en la industria. Para cada campaña de inyección de errores, el hardware de NESSY en VHDL es sintetizado completamente por lo que se adapta a la arquitectura de la FPGA objetivo.

En las siguientes secciones se describen en detalle los dos modos de inyección de errores que se han implementado en NESSY: en la memoria de configuración de la FPGA y en los flipflops utilizados por el diseño a examinar.

3.1. Inyección de SEUs en la memoria de configuración

Para poder modificar la memoria de configuración de la FPGA, previamente es necesario añadir un hardware adicional al circuito bajo test, como ya se ha comentado en la sección anterior. Este hardware, implementado en la propia FPGA donde se configura el circuito bajo test, es controlado por el usuario a través de una interfaz gráfica intuitiva que se ejecuta en un PC. La Figura 25 muestra la arquitectura a nivel de sistema. Al programa que hace de interfaz con el usuario y que se ejecuta en el PC, se le ha llamado **sistema software** y es el maestro en este enlace de comunicaciones, enviando comandos y solicitando información a la FPGA. A la parte implementada en la FPGA se le ha llamado **sistema hardware** y está basado en un microprocesador (softcore *MicroBlaze*), que es el encargado de procesar los mensajes provenientes del PC y de controlar el resto de componentes del hardware de control.

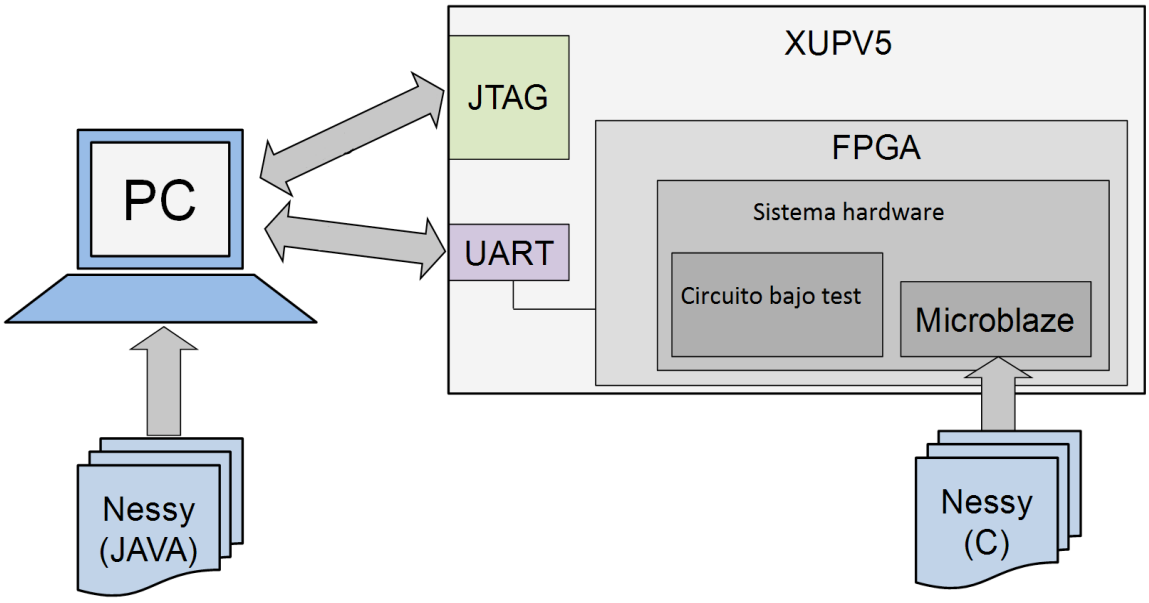

Figura 25: Arquitectura de NESSY

Las comunicaciones entre el PC y la FPGA se realizan mediante una interfaz RS232 y una interfaz JTAG. El enlace RS232 se utiliza para el envío de comandos al sistema

hardware o para la recepción de información de la inyección, mientras que la interfaz JTAG solo se utiliza una vez al comienzo de la inyección para la programación del bitstream global que incluye el sistema hardware.

Una vez el PC envía un comando al procesador, éste se encarga de ejecutarlo en el componente hardware que corresponda. Los comandos comprenden desde recibir o enviar un fichero de la memoria DDR de la FPGA, hasta modificar el contenido de la memoria de configuración, ejecutar el circuito bajo test, poner una serie de entradas al diseño bajo test que están definidas en el fichero testbench o leer sus salidas y compararlas.

Para la inyección de un error, se altera el contenido el bitstream parcial del circuito bajo test desde el *MicroBlaze,* modificando un único bit. Una vez se introduce el cambio deseado en el bitstream del diseño bajo test, se escribe este bitstream parcial en la memoria de configuración a través del puerto ICAP y se vuelve a ejecutar el testbench en el circuito. Las salidas generadas se comparan con el golden para detectar diferencias y por lo tanto fallos en el comportamiento deseado. El golden puede ser generado automáticamente a partir de un testbench o utilizar uno ya existente, enviado por el usuario a través del puerto RS232. Todo este proceso que supone la inyección de un error, está controlado mediante acciones ejecutables desde la interfaz de usuario y se explican más en detalle en la sección 3.1.2.

3.1.1. Sistema hardware

El sistema hardware está a su vez dividido entre la parte llamada hardware estático y otra llamada hardware dinámico.

La parte estática consiste en el hardware que se implementa al inicio en la FPGA y permanece configurada durante toda la ejecución. Comprende los siguientes componentes: el softcore del *MicroBlaze*, un controlador de memoria DDR, un controlador UART, un controlador de memoria BRAM, un controlador del puerto ICAP, el adaptador de interfaz del circuito bajo test y el bus del sistema. En esta arquitectura, todos los componentes están conectados al bus PLB del sistema y el *MicroBlaze* hace de maestro mientras que el resto de componentes hacen de esclavos.

La parte dinámica consiste únicamente en el diseño bajo test, aislado en su propia RPR para que la parte estática no pueda interferir en el rutado y posicionamiento de éste.

La Figura 26 muestra un esquema a nivel de bloque del sistema hardware.

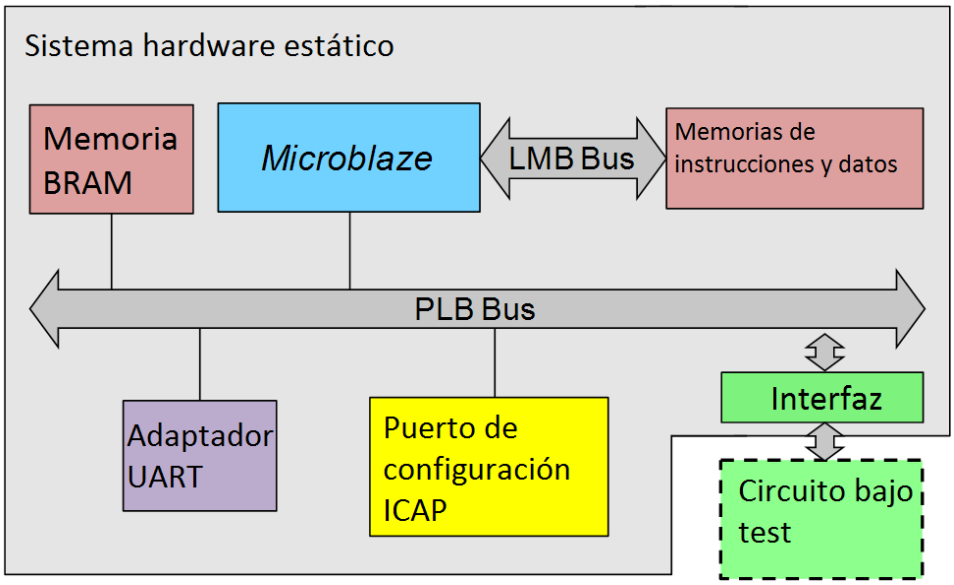

Figura 26: Sistema hardware

La interfaz de comunicación que existe entre la parte estática y la dinámica es un periférico HDL generado automáticamente por NESSY una vez el usuario ha proporcionado los ficheros fuente. La limitación actual en el número de entradas y salidas es de 128, pero es fácilmente ampliable a un número mayor en caso de ser necesario.

El procesador es el único módulo maestro del sistema, por lo que espera por el método de "polling" la llegada de algún comando a través del enlace serie para realizar las acciones oportunas. Las razones de incluir parte del procesado en el *MicroBlaze* de la FPGA son las siguientes:

- Permite el manejo del puerto ICAP de manera sencilla, cuya velocidad de funcionamiento es de 4 órdenes de magnitud más rápido que el JTAG. Puesto que la configuración parcial de la FPGA es algo que se hará muy a menudo durante la inyección de SEUs, esto supone una mejora significativa en rendimiento respecto al uso de JTAG.
- Reduce en la medida de lo posible las comunicaciones entre PC y la FPGA, al ser

éste uno de los cuellos de botella del sistema. Los bitstream parciales emulando SEUs, el testbench, los golden y resultados son gestionados por el *MicroBlaze* y almacenados en las BRAM de la FPGA.

3.1.2. Sistema software

El sistema software de NESSY es el programa que se ejecuta en el PC. Este software es el encargado de realizar la síntesis e implementación de un sistema hardware completo que contiene al diseño bajo test (parte dinámica) junto a su interfaz adaptada al bus del sistema y el resto del sistema hardware (parte estática). Los ficheros fuente que espera recibir NESSY por parte del usuario, deben ser sintetizables por cualquier herramienta de síntesis de Xilinx y, para cada prueba, se sintetiza e implementa un sistema hardware completo desde cero. Esto permite que el sistema pueda ser portable a otras FPGAs de Xilinx.

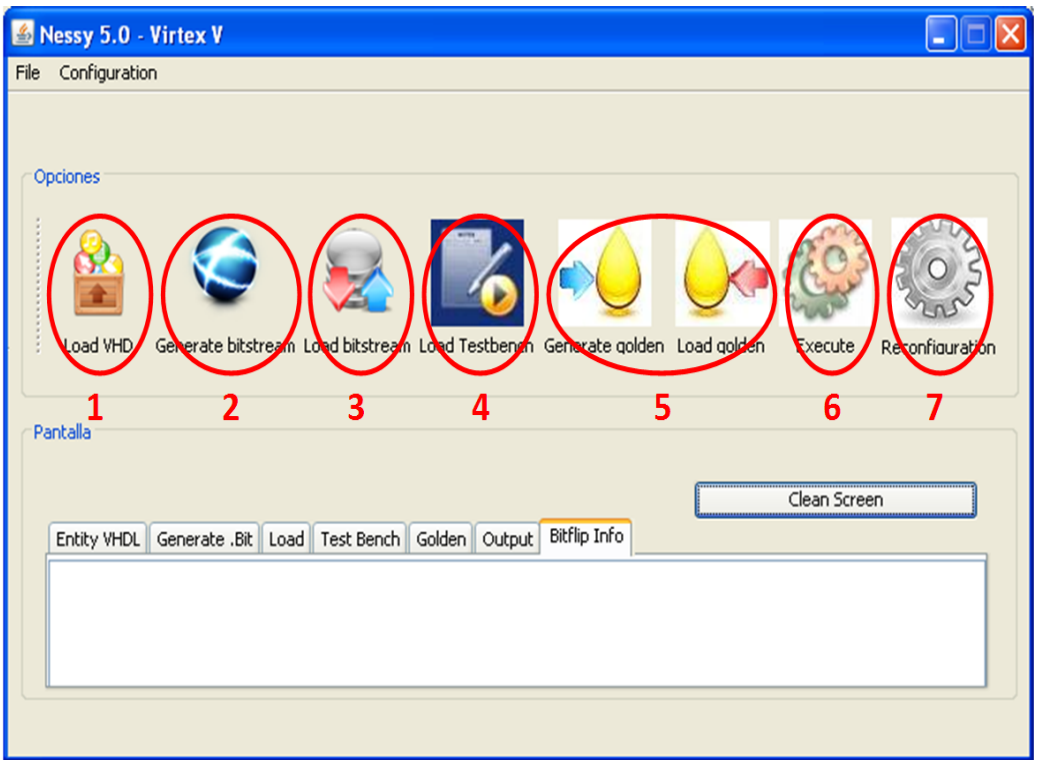

Figura 27: GUI de NESSY

Capítulo 3

La inyección de SEUs en NESSY consta de una serie de órdenes secuenciales que se identifican uno a uno con botones de la interfaz de usuario, como se puede apreciar en la Figura 27. Los pasos para realizar una campaña de inyección de errores son los siguientes:

1. Load VHD
2. Generate bitstream
3. Load bitstream
4. Load testbench
5. Load/generate golden
6. Execute
7. Reconfiguration

Load VHD se utiliza para que el usuario indique los ficheros fuentes del diseño al que quiere inyectar errores. En este paso, NESSY también adapta la entrada/salida del circuito bajo test con el controlador del bus PLB añadiendo para ello una interfaz (ver Figura 26).

Generate bitstream utiliza las herramientas ISE 12.1 y EDK 12.1 de Xilinx para generar el bitstream global que contiene todo el sistema hardware y el bitstream parcial del circuito bajo test. El bitstream global generado aquí es el que permite arrancar la FPGA y llevarla a un estado inicial. El bitstream parcial se usará en pasos posteriores.

Load bitstream programa la FPGA utilizando el bitstream global y comprueba que existe una comunicación entre el PC y la FPGA por el puerto serie mediante un comando de sincronización.

Load testbench permite al usuario enviar un archivo a la FPGA con el vector de entradas utilizadas para comprobar el correcto funcionamiento del diseño. Los ficheros de testbench son ficheros de texto plano que deben tener la estructura de una matriz de “0”s y “1”s. Cada fila determina un vector de entrada al circuito en un ciclo de reloj y la asignación de estos valores a las señales de entrada del circuito se realiza según el orden de aparición de éstas en la declaración de la entidad de más alto nivel del diseño bajo test.

Generate golden y **Load golden** son procesos complementarios: el primero permite generar un fichero golden desde cero ejecutando un testbench sobre el diseño bajo test y

guardando las salidas que se generen como fichero golden. El segundo método permite enviar desde el PC un fichero golden previamente generado. La estructura del fichero golden tiene el mismo formato que el fichero de testbench, con la excepción de que el ancho de cada fila de datos debe coincidir con el número de salidas, en vez de con el número de entradas.

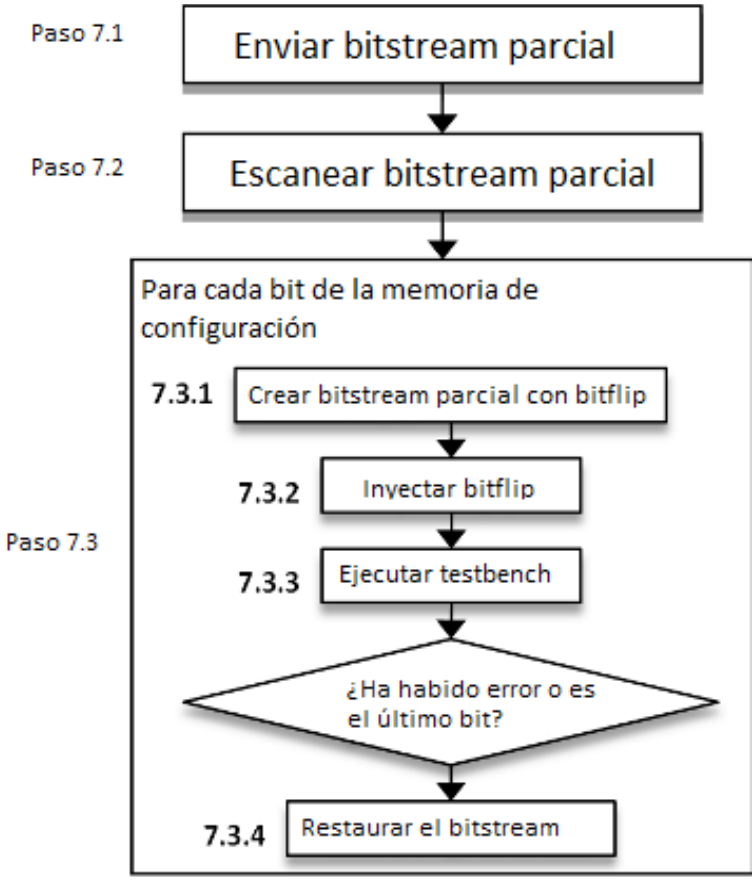

Figura 28: Diagrama de flujo para la inyección de errores en memoria de configuración con NESSY

Execute es una acción utilizada especialmente con propósitos de depuración. Lo que hace esta acción es ejecutar un determinado testbench actual y mostrar por pantalla las salidas que se han generado.

Reconfiguration es el proceso encargado de realizar la inyección de errores. El algoritmo de esta acción se puede dividir en pasos tal y como se muestra en la Figura 28.

1. Paso 7.1: el PC envía a la FPGA el bitstream parcial del diseño bajo test.
2. Paso 7.2: Una vez en la FPGA, este bitstream parcial se analiza y parsea para

identificar los mapas de configuración que contiene. En particular, habrá tantos mapas de configuración como zonas tenga el circuito. Cada mapa de configuración contiene una serie de instrucciones para realizar una escritura al registro de dirección de frame (frame Address Register o FAR por sus siglas en inglés), donde dicha dirección coincidirá con la del bloque del circuito, seguido de escrituras al registro de datos de entrada del frame (frame Data Register Input o FDRI). En este último es a donde se enviarán los datos a escribir en la memoria de configuración de la FPGA.

3. Paso 7.3: Finalmente, se realiza la inyección de un SEU en cada bit de la zona de la memoria de configuración asociada al circuito bajo test.
    1. Subpaso 7.3.1: Para cada inyección de un SEU, se debe generar un bitstream parcial modificado, donde se invierte el valor del bit correspondiente (es decir, se inyecta un bitflip). Ya que el segmento más pequeño posible direccionable es un frame (Apéndice I), el bitstream parcial modificado consiste en una instrucción de escritura en el registro FAR para escribir la dirección del frame y las correspondientes escrituras al registro FDRI con el contenido del frame modificado. Los datos escritos mantienen el valor original del frame excepto por el bit donde se quiere emular el SEU.
    2. Subpaso 7.3.2: A continuación se programa la FPGA enviando este bitstream parcial modificado al puerto ICAP. Puesto que este bitstream parcial modificado tiene el tamaño mínimo aceptado por dicho puerto, el tiempo de configuración asociado a este subpaso es el menor posible.
    3. Subpaso 7.3.3: Una vez el SEU ha sido inyectado, el MicroBlaze se encarga de ejecutar el testbench poniendo a la entrada del circuito un vector de entradas por cada ciclo de reloj. A continuación se leen las salidas que éste produce y se comparan con el golden para ver si tras inyectar el SEU se producen cambios en el comportamiento del diseño bajo test.
    4. Subpaso 7.3.4: Finalmente, se debe restaurar el error introducido antes de proceder a tratar el siguiente bit. Para ello, si el bit tratado *i* pertenece al mismo frame que el bit anterior *i-1*, no se ejecuta ninguna acción puesto que el bitstream parcial modificado del bit *i* sobrescribirá el error introducido por el bit *i-1*. En

caso contrario, primero se debe restaurar el bitstream parcial original del bit *i-1* y después proceder con el tratamiento del bit *i* volviendo al subpaso 7.3.1. Esta optimización reduce el tiempo requerido en el proceso de inyección, ahorrando reconfiguraciones innecesarias por parte del puerto ICAP.

3.2. Inyección de SEUs en flipflops

Los flipflops son un tipo particular de celdas de memoria que retienen internamente un estado o valor lógico refrescado en función de otra señal, típicamente un reloj. Los flipflops que utiliza la familia de FPGAs Virtex son biestables tipo D (ver Figura 29) disparados por flanco, por lo que en cada flanco de reloj el valor de la entrada D se lee y se copia a la salida Q. El estado de un flipflop consiste en un bit lógico que no pertenece a la memoria de configuración de la FPGA. En las FPGAs Xilinx Virtex también es posible emular SEUs en dichos bits. Para ello, se deben modificar ciertos bits de la memoria de configuración de la FPGA utilizando una metodología similar a la explicada en la sección 3.1. . Para ello y puesto que el estado de los flipflops no se almacena en la memoria de configuración, por cada flipflop es necesario modificar el contenido de un par de bits de la memoria de configuración (llamados INIT0/INIT1) asociados a cada flipflop y que definen su estado cuando se produce un reset y, junto a esto, utilizar la señal de reset global de la FPGA (GSR). Los bits INIT0/INIT1 son complementarios, esto significa que solo es posible escribir los valores “01” (para el estado '1' como estado por defecto o estado de reset) o “10” (para el estado '0') e indican el estado al que irá el flipflop una vez se active la señal GSR de la FPGA. Esto supone, que para la inyección de un SEU, es necesario primero leer y modificar el contenido de los bits INIT0/INIT1 de todos los flipflops y después generar un pulso de la señal GSR. La modificación de los bits INIT0/INIT1 se debe realizar de manera que se mantenga el estado que tuvieran todos los flipflops antes de la activación de la señal de reset, excepto aquél que se quiera inyectar el error. En este último caso se deberá invertir su valor lógico.

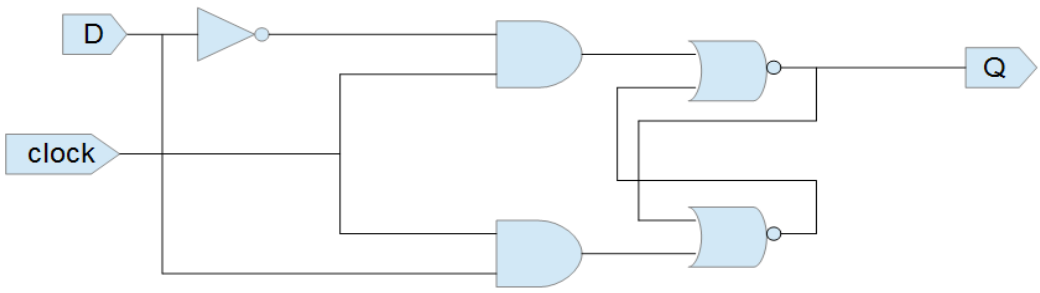

Figura 29: Flipflop D

Este algoritmo tiene una complejidad lineal en el tiempo con el número de flipflops del diseño, ya que para una única inyección se debe iterar sobre los N flipflops. Esto hace que realizar una campaña de errores en circuitos complejos, se requiera una gran cantidad de tiempo. Es por esto que NESSY utiliza otro método más eficiente en términos de tiempo, ya que tiene una complejidad en tiempo constante con el número de flipflops en el diseño y, por lo tanto, el algoritmo escala mejor para diseños complejos. Para llevar a cabo este método, es necesario añadir un hardware de instrumentación para cada flipflop sobre el que se quiera inyectar errores. Esta diferencia en la metodología, supone que para la inyección de errores sobre flipflops, NESSY pierde su característica de no intrusividad.

Para entender el hardware de instrumentación, es necesario primero entender el funcionamiento de los flipflops de las FPGAs de Xilinx. Además de la arquitectura del flipflop descrita en la Figura 29, los flipflops de la familia Virtex tienen señales de control adicionales: CE, SR y REV. Mientras que la primera es un chip enable y se encarga de habilitar el flipflop, las otras dos hacen la función de reset y de set, forzando el contenido de éstos al valor lógico '0' o '1', respectivamente.

El método para la emulación de SEUs en estos componentes se apoya en la técnica utilizada para la inyección de errores en la memoria de configuración y adicionalmente, utiliza para cada flipflop sus señales de SR y REV a través de unos bits complementarios llamados SRHIGH (SR activo a '1') y SRLOW (SR activo a '0'), que son accesibles desde la memoria de configuración. Al ser complementarios, sólo existen 2 combinaciones válidas, al igual que ocurre en los bits INIT0/INIT1:

1. SRHIGH=0, SRLOW=1.

2. SRHIGH=1, SRLOW=0.

La función de estos bits es la de configurar el comportamiento de las señales de SR y REV del flipflop al que están asociado. Dicho comportamiento está definido según muestra la Tabla II.

SRHIGH=0, SRLOW=1			SRHIGH=1, SRLOW=0		
SR	REV	Q(t+1)	SR	REV	Q(t+1)
0	0	Q(t)	0	0	Q(t)
0	1	1	0	1	0
1	0	0	1	0	1
1	1	0	1	1	0

Tabla II: Tabla de verdad de los flipflops de las FPGAs Xilinx Virtex-5

La conclusión que se puede obtener es que si el flipflop funciona de acuerdo a la configuración 1 (SRHIGH=0, SRLOW=1), SR funciona como un reset y REV como un set. En cambio, en la configuración 2 (SRHIGH=1, SRLOW=0), SR hace de set y REV de reset. La herramienta de síntesis de Xilinx considera por defecto la configuración 1 y, para cada flipflop del diseño a implementar, su señal de SR irá conectada a la señal de reset, mientras que su señal de REV irá conectada a un set y el CE a un enable.

Descrito el funcionamiento de las señales SR y REV, se puede emular un SEU en el contenido de un flipflop mediante la modificación del contenido de los bits SRHIGH/SRLOW en conjunto con el hardware de instrumentación adicional (y que se describirá a continuación). Dicho hardware de instrumentación consiste en una serie de funciones booleanas que tienen como entradas una señal de control "*inyección*", el estado actual del flipflop y las señales de reset, set y chip enable del circuito para generar las señales de los flipflops: SR, REV y CE. De esta manera, se puede invertir o mantener el estado de los flipflops según el estado de sus bits SRHIGH/SRLOW. Las funciones a implementar son tres:

$$(1)\ FF_{CE} = \overline{inyección} \cdot original\_ce$$

$$(2)\ FF_{SR} = inyección \cdot \overline{Q(t)} + \overline{inyección} \cdot original\_sr$$

$$(3)\ FF_{REV} = inyección \cdot Q(t) + \overline{inyección} \cdot original\_rev$$

En las ecuaciones (1) - (3), se asume que por cada flipflop existente en el diseño, aparecen sus tres señales de control. Si en el HDL no aparecieran explícitamente alguna de estas tres señales de control, como se ha explicado, la herramienta de síntesis las inferirá por defecto como original_sr = '0', original_ce = '1' y original_rev = '0' según corresponda.

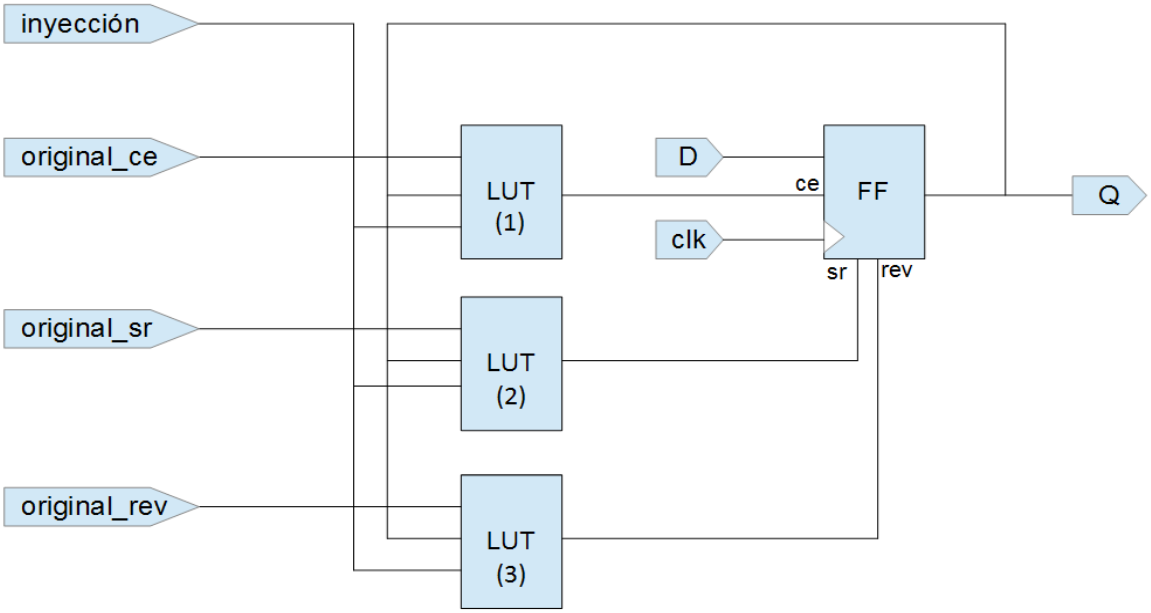

Figura 30: Instrumentación para un flipflop

La implementación de esta lógica necesita una LUT para cada función, quedando implementado cada flipflop junto a su hardware de instrumentación como se muestra en la Figura 30.

Según (1), (2) y (3) si no se inyecta un SEU (inyección='0'), el flipflop trabajará nominalmente:

$$FF_{CE} = original\_ce$$

$$FF_{SR} = original\_sr$$

$$FF_{REV} = original\_rev$$

De lo contrario (inyección='1'), las señales de FF_{REV} y FF_{SR} dependerán únicamente de Q(t):

$$FF_{CE} = 0$$
$$FF_{SR} = \overline{Q(t)}$$
$$FF_{REV} = Q(t)$$

O lo que es lo mismo, Q(t) determina el valor de las señales FF_{SR} y FF_{REV}:

$$Q(t) = 0 \rightarrow FF_{REV} = 0,\ FF_{SR} = 1$$
$$Q(t) = 1 \rightarrow FF_{REV} = 1,\ FF_{SR} = 0$$

Esta dependencia entre señales, reduce el comportamiento booleano descrito en la Tabla II a las filas 2 y 4, permitiendo que al modificar los valores de los bits SRHIGH/SRLOW, se pueda invertir o mantener el estado actual del flipflop como se aprecia en la Tabla III.

SRHIGH=0, SRLOW=1		SRHIGH=1, SRLOW=0	
Q(t)	Q(t+1)	Q(t)	Q(t+1)
0	0	0	1
1	1	1	0

Tabla III: Tablas de verdad con instrumentación

En esta situación el comportamiento de Q(t+1) es el de $\overline{Q(t)}$ cuando SRHIGH=1, SRLOW=0 o lo que es lo mismo, se invierte su estado. En cambio, si SRHIGH=0, SRLOW=1, Q(t+1) es el mismo que Q(t).

En resumen, junto al hardware de instrumentación, si se escribe "10" en los bits SRHIGH/SRLOW de la memoria de configuración de un flipflop, cuando se genere un pulso de la señal "inyección", se producirá un bitflip:

$$Q(t+1) = \overline{Q(t)}$$

Mientras que si se escribe un "01" en dichos bits (valor por defecto), ante un pulso de la

señal “inyección” el estado del flipflop se conservará:

$$Q(t+1) = Q(t)$$

Esta forma de inyectar un error afecta al algoritmo utilizado para la inyección en la memoria de configuración (mostrado en la Figura 28), por lo tanto, NESSY debe distinguir el tipo de inyección que se quiera realizar. Dentro del proceso de reconfiguración, cuando se quiera realizar la inyección en flipflops, se deberán realizar una serie de pasos de manera distinta a como se hace en la inyección en la memoria de configuración. Este nuevo algoritmo para inyección de errores en flipflops se muestra en la Figura 31.

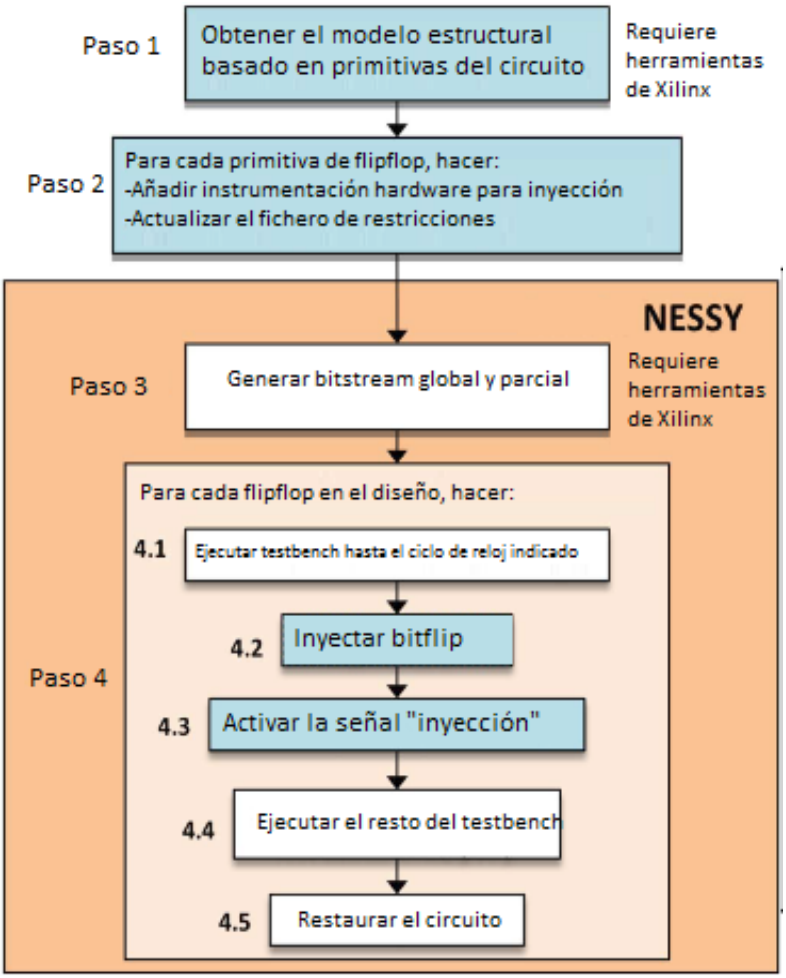

Figura 31: Flujo de proceso para inyección sobre flipflops

- Paso 1: Como se ha mencionado, en este caso es necesario añadir instrumentación para realizar la inyección de errores en flipflops. Para ello, es imprescindible obtener el modelo RTL (o estructural) del circuito bajo test en el que los flipflops aparecen instanciados directamente a través de primitivas de Xilinx en código HDL [45]. Dicho modelo se obtiene utilizando la herramienta Xilinx ISE y debido a la complejidad de introducir dicha instrumentación de manera automática, este proceso se explica más adelante en su propia sección, "Adaptador VHDL" (3.2.1.).
- Paso 2: A continuación, dicho modelo estructural se analiza para detectar toda primitiva que instancie un flipflop y se envuelve con el hardware adicional necesario.
- Paso 3: Este paso consiste en la generación del bitstream global y el parcial de la misma forma que se hace para la inyección de errores en la memoria de configuración.
- Paso 4.1: El testbench en este tipo de inyección de errores se ejecuta por partes. Primero NESSY coloca en las entradas del circuito, ciclo tras ciclo, tantos vectores de entrada como haya indicado el usuario hasta el ciclo anterior a donde se quiere inyectar el bitflip.
- Paso 4.2: Con el reloj detenido, NESSY puede modificar la memoria de configuración de la FPGA cambiando los valores de los bits SRHIGH/SRLOW del flipflop deseado al valor "10". Una vez modificada la memoria, la señal de reloj se restaura.
- Paso 4.3: Activación de la señal de inyección. Esta activación se realiza mediante la modificación del testbench ejecutado por el circuito. Esto significa que sobre el testbench original, se debe añadir a sus vectores de entradas una señal adicional que es la que indicará en qué ciclo de reloj se inyecta el error. La herramienta realizará la emulación del SEU activando la señal de inyección en el ciclo de reloj *n*-ésimo, donde *n* es el número de fila dentro del testbench donde la señal de inyección valga 1.
- Paso 4.4: Se termina de ejecutar la parte restante del testbench y se comprueba si ha habido errores en las salidas.
- Paso 4.5: Se restaura el estado original del circuito modificando los bits

SRHIGH/SRLOW a “01” mediante la escritura del frame correspondiente y así dejar el circuito preparado para la siguiente inyección.

La modificación de los bits SRHIGH/SRLOW se realiza de la misma forma que cuando se modifica cualquier bit de la memoria de configuración, esto es, mediante la modificación de un bitstream parcial del tamaño de un frame. Por lo tanto, por lo que respecta a este método de inyección, el rendimiento no se ve afectado con respecto al método de inyección en la memoria de configuración.

3.2.1. Adaptador VHDL

Como se ha comentado en el apartado anterior, con el objetivo de añadir la instrumentación necesaria para poder inyectar errores en los flipflops, es necesario modificar el código VHDL del diseño original. La interfaz accesible para el usuario de un flipflop en VHDL, puede variar dependiendo del tipo de instanciación que se realice en el código. Existen varias formas de instanciar un flipflop, como se explica en [45], y las diferencias vienen dadas por el número de entradas/salidas que se utilicen, ya que no siempre es necesario utilizar todas las señales de control disponibles. La herramienta de síntesis inferirá a partir del código la utilización de un flipflop y lo sustituirá durante el proceso de implementación por su primitiva correspondiente, conectando las señales no utilizadas a su valor por defecto (SR='0', REV='0' y CE='1'). Existe también la posibilidad de que el diseñador utilice directamente estas primitivas en el código. Puesto que NESSY necesita acceder a las tres señales de control: CE, SR y REV de los flipflops, se debe modificar el código VHDL para garantizar que en todas las instanciaciones de flipflops del diseño, aparezcan dichas señales.

Para ello, primero se utiliza la herramienta de síntesis de Xilinx, para obtener el modelo estructural post-síntesis del diseño. Este código, que describe el modelo estructural del diseño, es lógicamente equivalente al diseño original y utiliza las primitivas soportadas por la tecnología de la FPGA destino, las cuales incluyen a los flipflops. En el caso de NESSY,

hace una traducción del código VHDL original a otro con primitivas de componentes compatibles con la FPGA Virtex-5.

Este nuevo fichero VHDL estructural será la entrada al proceso de NESSY llamado "adaptador VHDL", que se encargará de parsear el código del modelo estructural buscando primitivas de flipflops en todas sus variantes. A continuación se enumeran todas las primitivas asociadas a flipflops que se pueden encontrar [45], así como la lista de sus señales de entrada y salida. Aclarar que en estas primitivas de flipflops, definidas por Xilinx, la señal de SR se llama CLR mientras que la señal de REV se llama PRE:

- FDCE: D, CE, C, CLR, Q
- FDE: PRE, D, CE, C, CLR, Q
- FDCPE: PRE, D, CE, C, CLR, Q
- FDRSE: S, D, CE, C, R, Q
- FDCE_1: D, CE, C, CLR, Q
- FDCPE_1: PRE, D, CE, C, CLR, Q
- FDRSE_1: S, D, CE,C, R, Q
- FDP: PRE, DC, Q
- FDR: S, D, CE, C, R, Q
- FDS: Q, D, C, S

Donde las definiciones de las distintas señales de cada primitiva son:

- D: data input
- CE: clock enable
- C: clock
- CLR: asynchronous clear
- Q: data output
- PRE: asynchronous preset
- S: set
- R: synchronous reset

Por cada primitiva de flipflop encontrada, NESSY la sustituirá por la primitiva FDRSE

si es síncrono o por FDCPE si es asíncrono. Además, a cada flipflop se le añadirá el hardware de instrumentación para que implemente las ecuaciones (1), (2) y (3), quedando el diseño mostrado en la Figura 30.

En dicha figura, se aprecia que cada ecuación se corresponde a una LUT, donde cada una de éstas deberá implementar una función booleana:

(1) Señal de CE:

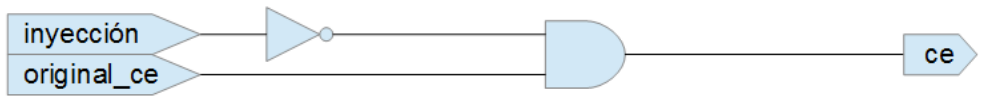

Figura 32: Implementación de CE

(2) Señal de SR:

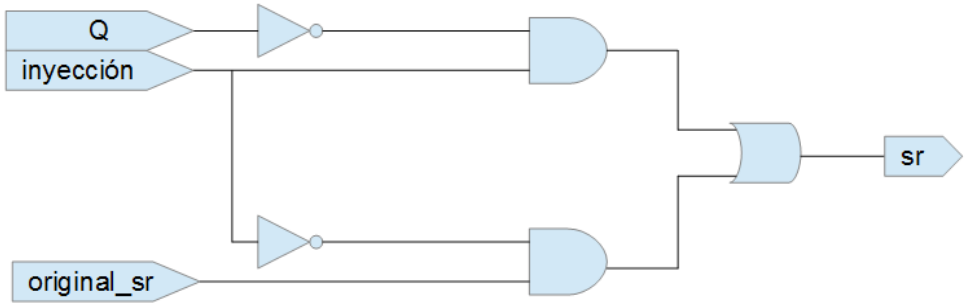

Figura 33: Implementación de RST

(3) Señal de REV:

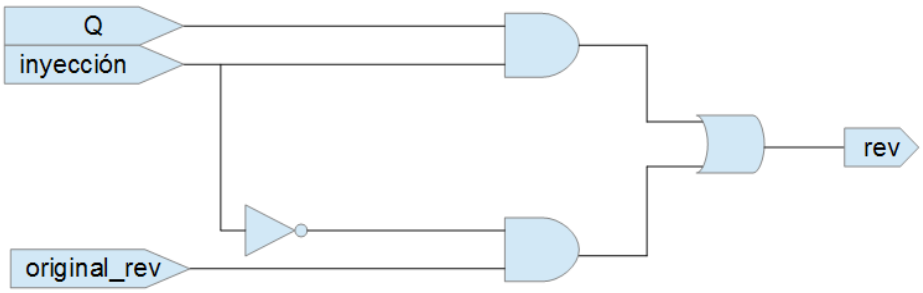

Figura 34: Implementación de SET

En caso de no existir algunas de las tres señales de control en la primitiva del flipflop

original, como se ha explicado anteriormente, la señal en cuestión, se conectará por defecto a un valor fijo (original_sr = '0', original_rev = '1', original_ce = '1'). En dichos casos, donde alguna o ninguna de estas señales aparezcan, las funciones a implementar se simplifican y serán función únicamente de las señales de inyección, "inyección" y del estado del flipflop en el ciclo actual, Q(t):

- original_sr = '0' y original_set = '0' al ser parte de una función OR.
- original_ce = '1' al ser parte de una puerta AND.

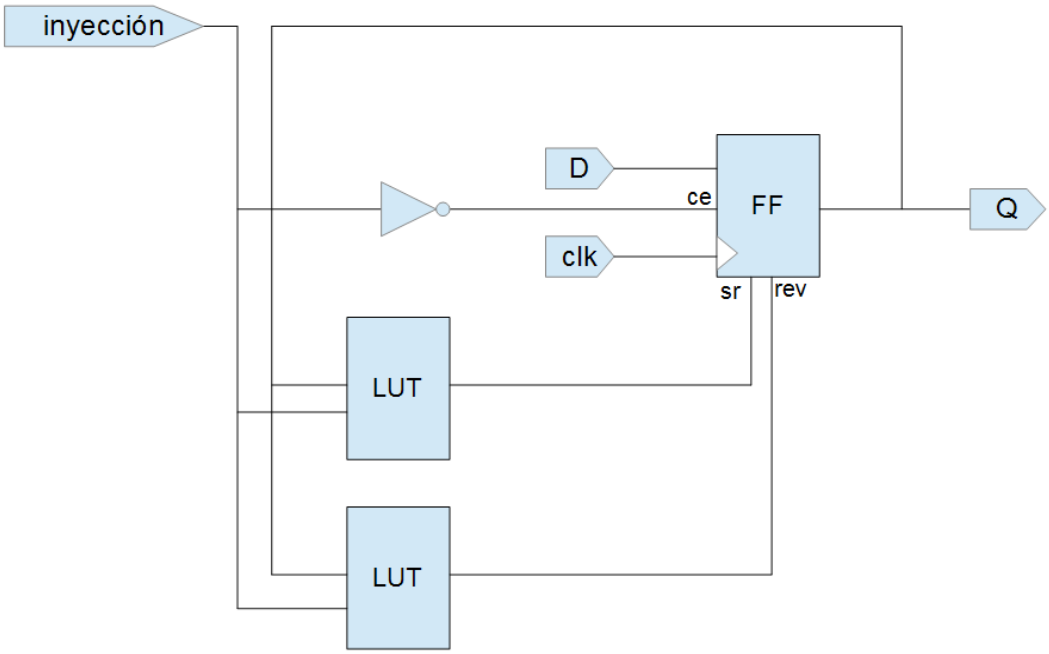

Figura 35: Adaptación FF sin señales de SR, REV y CE

En el caso límite de esta situación, tendríamos un flipflop donde no aparecen ninguna de las tres señales de control. El hardware de instrumentación en este caso quedará como el que se muestra en la Figura 35.

Capítulo 4: Técnicas de protección de circuitos

En este capítulo se van a estudiar dos familias de técnicas de protección de circuitos bien diferenciadas. En primer lugar, se ha partido de métodos existentes (TMR) donde su fiabilidad ha sido ya demostrada y que se utilizan actualmente en diseños del sector aeronáutico y del sector espacio para estudiar variantes de éstos que mejoren los resultados ya existentes. En segundo lugar, se han desarrollado métodos novedosos, por un lado basados en el conocimiento avanzado de la estructura de una familia muy específica de circuitos, como son las redes neuronales, para hacerlas más robustas y, por otro lado, en diseños basados en la utilización de recursos específicos de la FPGA (DSPs) en lugar de los CLBs de propósito general.

4.1. TMR

La primera técnica de protección de circuitos que se ha desarrollado en esta tesis doctoral ha sido la ya comentada TMR, con algunas variantes en su implementación. Estas son:

1. TMR1: TMR simple
2. TMR2: TMR + aislamiento
3. TMR3: TMR + aislamiento + votador blindado

Para la implementación y validación del TMR simple, la arquitectura y metodología de inyección de errores de NESSY explicada es compatible con este método. Sin embargo, en el caso de las variantes de TMR 2 y 3, el diseño bajo test presenta una serie de particularidades que obliga a que NESSY sea capaz de inyectar este tipo de errores

especiales. Esto implica que para poder dar soporte a estas variantes de TMR en la herramienta, ha sido necesario extender la funcionalidad de la inyección de errores.

En el TMR simple, el diseño bajo test sobre el que se inyectará errores acabará siguiendo la arquitectura mostrada por la Figura 36.

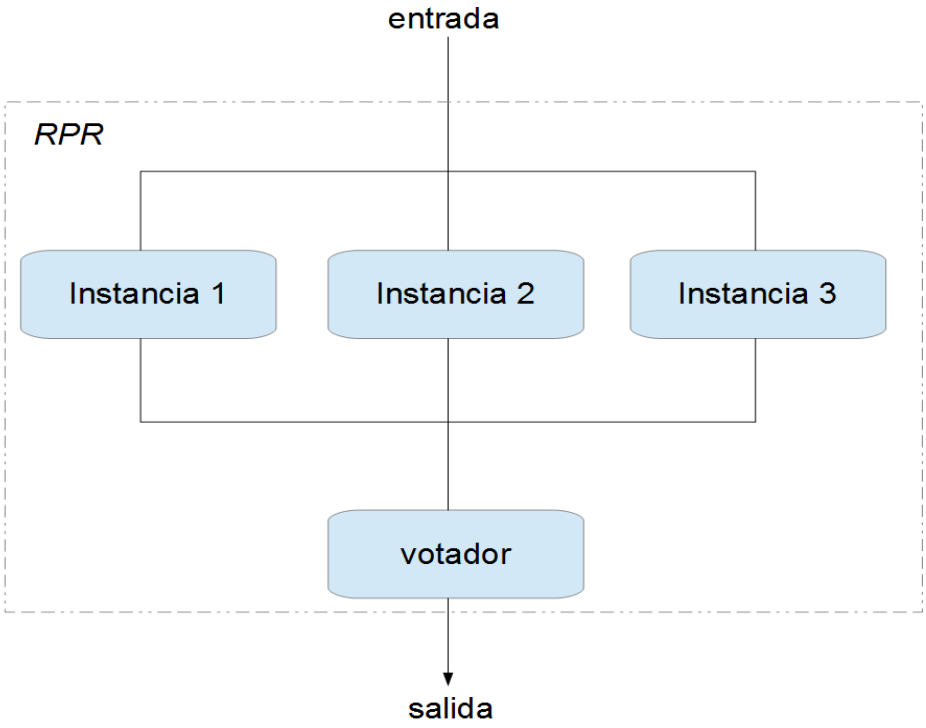

Figura 36: TMR simple

La aplicación del TMR simple, consiste en triplicar el diseño a proteger y añadir un votador combinacional, que decidirá por mayoría la salida correcta. Las tres instancias del circuito junto con el votador serán parte del mismo circuito bajo test donde inyectar errores y pertenecerán a una misma RPR. La implementación de esta técnica se puede realizar manualmente por el diseñador modificando el diseño que quiere proteger. Sin embargo, dado un diseño en VHDL cualquiera, NESSY puede implementar su variante de TMR simple de manera automática realizando la triplicación del circuito bajo test, generar el votador y la conexión entre ambos. Por anto, es posible realizar una inyección de errores tanto a un diseño cualquiera como a su variante de TMR simple equivalente y comparar los resultados de manera automática.

El problema que presenta este método, es que la herramientas de síntesis pueden

optimizar partes del diseño al comprobar que son redundantes y optar por unirlas, dando lugar a un TMR selectivo indeseado. El caso más habitual donde esto puede suceder es en las constantes, muy comunes en los filtros digitales, donde un SEU en alguna de estas constantes, supondría la propagación de este fallo a las tres copias del diseño y por lo tanto el error no sería enmascarado ya que el votador sería incapaz de distinguir la salida errónea.

Para solucionar este problema, se han propuesto alternativas de implementación al TMR simple más robustas, donde se asegura un aislamiento real entre las distintas instancias del circuito. El **TMR + aislamiento** evita esto asegurándose que las tres copias del circuito son independientes entre sí. Para que esto sea posible en NESSY, se debe realizar una serie de cambios en la forma de inyectar errores. En este caso, puesto que la triplicación debe mantenerse durante todo el proceso de síntesis, es necesario separar las distintas instancias entre sí y para ello, la herramienta de síntesis debe considerar cada instancia del diseño como una RPR independiente, de manera que no haya ninguna relación entre las tres copias del diseño. Al votador también se le considera como otra RPR independiente.

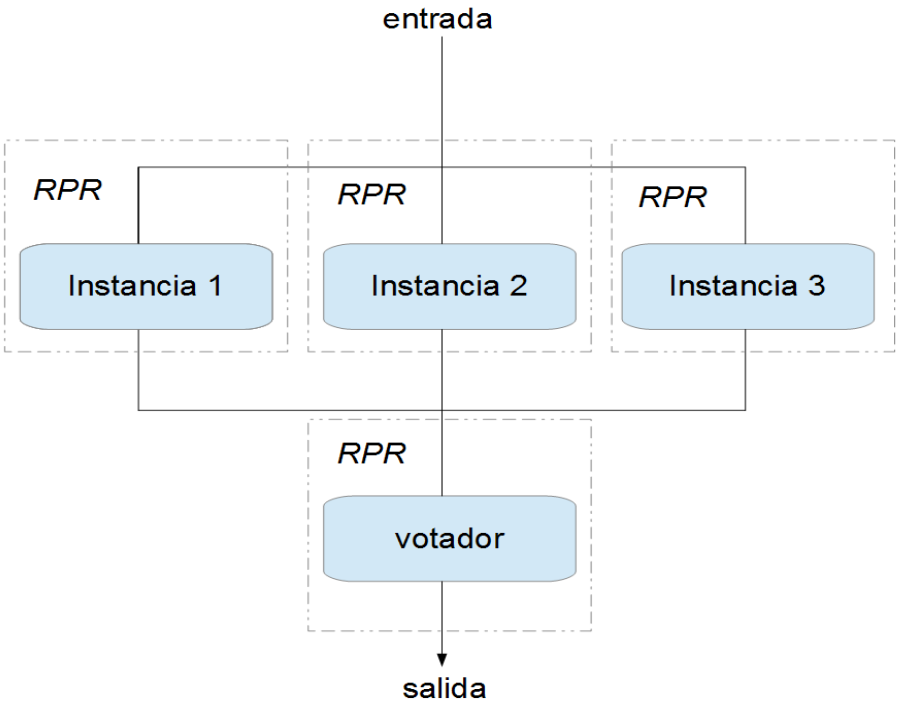

Figura 37: TMR con aislamiento

En esta implementación NESSY utiliza como entrada la descripción VHDL original del

diseño bajo test y la plataforma se encarga de generar automáticamente las tres RPRs necesarias con una copia del diseño en cada una, generar el fichero VHDL del votador con su interfaz adaptada en su propia RPR, unir las tres instancias con el votador y al resto del hardware de control. En estos tipos de inyección, la arquitectura que tiene NESSY se muestra en la Figura 37.

Para que la distinción de las tres regiones en las que se colocará cada una de las tres instancias del diseño triplicado se mantenga después de pasar por las herramientas de procesado de Xilinx, es necesario seguir una serie de pasos.

1. Las tres copias del diseño deben ser completamente independientes entre sí. Para ello, lo primero es declararlas como entidades independientes. Puesto que el sistema hardware de NESSY es un proyecto de la herramienta Xilinx EDK, cada instancia del circuito debe ser considerada por el proyecto como un core independiente. Esto se puede observar en la Figura 38.

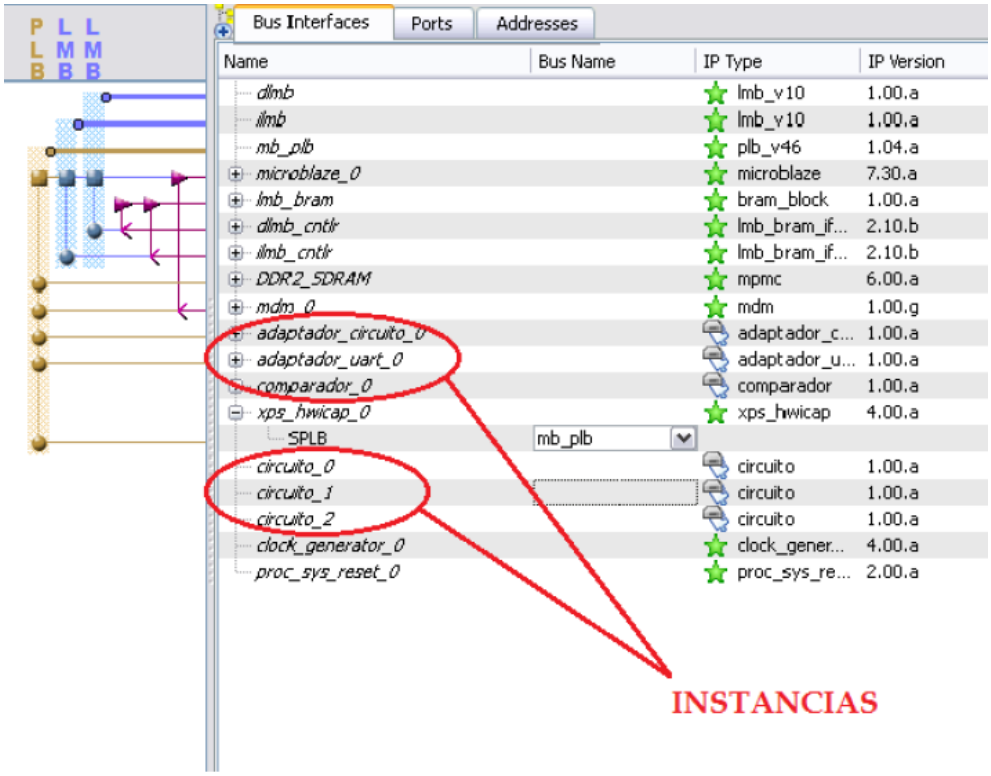

Figura 38: Arquitectura de cores para TMR + aislamiento con Xilinx EDK

2. Gracias al paso anterior, ahora es posible indicar a la herramienta de síntesis que el

posicionamiento de las tres copias del diseño debe realizarse en regiones independientes. NESSY solicita al usuario las coordenadas de la FPGA donde quiere situarlos Esto implica que se modificarán las restricciones utilizadas en el proceso de place&route de las herramientas de Xilinx, añadiendo una RPR para cada instancia del circuito triplicado. De este modo se asegura que la triplicación se realiza para todos los recursos del diseño.

Una inyección de errores sobre un circuito que implemente la variante de TMR 2, implica que existirán errores que afecten al diseño del votador y al ser éste único para las tres copias del diseño, un error aquí se traducirá en un error en los resultados. La variante 3 se diferencia con respecto a la 2 únicamente en que NESSY no inyectará errores sobre la RPR del votador, emulando que éste sea inmune a SEUs. Por lo que en este caso, tenemos un esquema idéntico al TMR con aislamiento con la peculiaridad de que NESSY no inyectará errores en la RPR del votador (Figura 39).

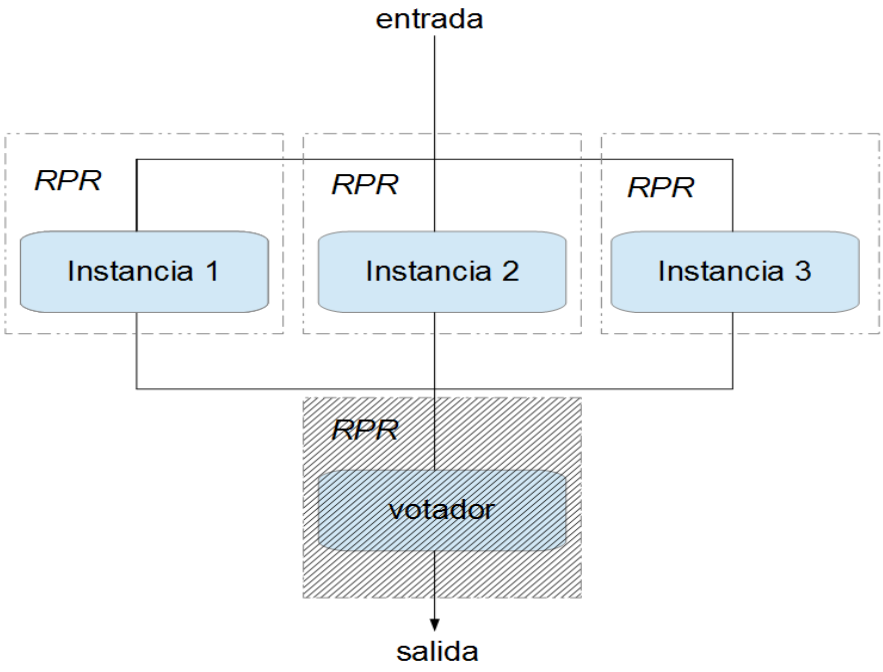

Figura 39: TMR con aislamiento y votador blindado

Esta es una suposición realista puesto que en este caso, se asume que el votador estará implementado como un ASIC con lógica combinacional y como se ha explicado

anteriormente, de esta manera no se podría producir un SEUs en él al no contener elementos de memoria.

La triplicación en estas 2 últimas variantes de TMR, no es total, puesto que hay partes del diseño donde no se puede aplicar. Estas son:

1. Las entradas de datos. Los datos introducidos a las tres instancias del diseño son comunes. Por tanto, un SEU en la memoria de configuración del rutado de esta parte afectaría por igual a las tres instancias, siendo así imposible de detectar.
2. Las salidas de datos. El motivo es similar al punto anterior, pero en este caso ocurre a la salida del votador. El rutado de sus salidas está almacenado en la memoria de configuración de la FPGA, por lo que si se da un SEU en esa zona de memoria, es imposible de detectar.
3. Las señales críticas asíncronas. Por ejemplo, la señal de reset. Si el diseño tiene un reset asíncrono, dicha señal no se puede registrar ni aislar de las distintas instancias del diseño triplicado. Esto implica que, si un SEU afecta a la memoria de configuración de un reset o set asíncronos, el error se podría propagar al resto del diseño redundante de manera catastrófica.

Después del estudio y análisis de esta técnica, una posible mejora sencilla y compatible con la tecnología actual de las FPGAs, consistiría en un cambio en la arquitectura de las FPGAs donde se incluyan votadores inmunes a SEUs por diseño. De esta manera, se deja al ingeniero la posibilidad de elegir qué partes del diseño son críticas y deberán ser redundantes, pudiendo así hacer una mejor gestión de los recursos disponibles. Actualmente, la alternativa comercial se reduce a un pequeño número de FPGAs perteneciente a una familia tecnológica calificada para espacio donde por diseño, se aplica TMR a todos sus recursos con un alto coste económico y muy poca lógica configurable en comparación a las familias orientadas a electrónica de consumo o industrial. La opción propuesta, de una arquitectura con sólo votadores inmunes a SEUs y no toda la lógica, sería complementaria al resto de técnicas aplicadas a las FPGAs comerciales “rad hard” actuales para proteger los circuitos frente a otros SEEs (como pueden ser los circuitos anti latch-up). En el capítulo 5.3 se aplica esta técnica con sus variantes a distintos diseños para ver su comportamiento y

comprobar la efectividad de la misma.

4.2. Redes Neuronales de Hopfield tolerantes a fallos (FT-HNN)

Las redes neuronales de Hopfield [18] (HNN por sus siglas en inglés) son un caso particular de redes neuronales y, por lo tanto, su estructura es la de una matriz de neuronas relacionadas entre sí mediante una matriz de pesos. Las neuronas que componen la red son binarias, es decir, que las posibles salidas de sus neuronas solo pueden tener dos valores en función de si superan un umbral θ o no. Además, para estas redes se define un parámetro llamado "energía de red":

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} s_j s_i + \sum_i \theta_i s_i$$

Donde:

- w_{ij} es el peso entre las neuronas i y j.
- s_i es el estado de la neurona i.
- θ_i es el valor umbral de la neurona i.

Estas redes son sistemas estables y convergen a mínimos locales de su función de energía, siempre y cuando sean redes simétricas, es decir, $w_{ij} = w_{ji}$ (en caso contrario no se garantiza la convergencia). Esta característica se puede utilizar para recuperar o reconocer patrones parciales. Para ello, el primer paso consiste en entrenar la red asignando pesos a sus neuronas, de tal forma, que el patrón buscado sea un mínimo en su función de energía. Así, cuando en las entradas se introduzca un patrón similar al utilizado para el entrenamiento, la red acabará convergiendo a ese mínimo local, dando como reconocido el patrón utilizado como entrada. En cambio, si la entrada introducida está muy distorsionada, la salida acabará convergiendo a otro mínimo local de la función permitiendo saber que el

patrón de entrada no es el esperado.

Este concepto puede ser utilizado en muchas aplicaciones, como por ejemplo el reconocimiento de imágenes, control, robótica, procesamiento de señal, clasificación de datos, eliminación de ruido y recuperación de información. Las HNNs han sido utilizadas en aplicaciones espaciales, satélites, y también en otras aplicaciones en entornos peligrosos para circuitos electrónicos [1].

Las FPGAs permiten una implementación de algoritmos basados en HNNs de alta velocidad [33] [44], por lo que estos dispositivos son buenos candidatos para implementar estos diseños, lo que implica que deban poder trabajar bajo los efectos de los SEUs, y de ahí que sea interesante analizar la vulnerabilidad a estos errores e investigar mejoras.

La arquitectura de una red neuronal, y en particular la de las HNN, [12] se considera intrínsecamente tolerante a fallos. Sin embargo, en implementaciones hardware pueden manifestarse fallos no destructivos (SEUs y SETs), provocados por la colisión de partículas de altas energías. Los detalles de la implementación de una HNN se explican en [44].

En esta tesis doctoral se ha examinado la robustez de la mencionada HNN, así como de una versión tolerante a fallos (FT-HNN por sus siglas en inglés). Este circuito es una red neuronal de tipo Hopfield, más robusta frente a SEUs por diseño. La FT-HNN es más robusta debido al reforzamiento en sumadores y multiplicadores de la red neuronal, se aplica redundancia temporal a los multiplicadores, mientras que se aplica redundancia espacial a los sumadores.

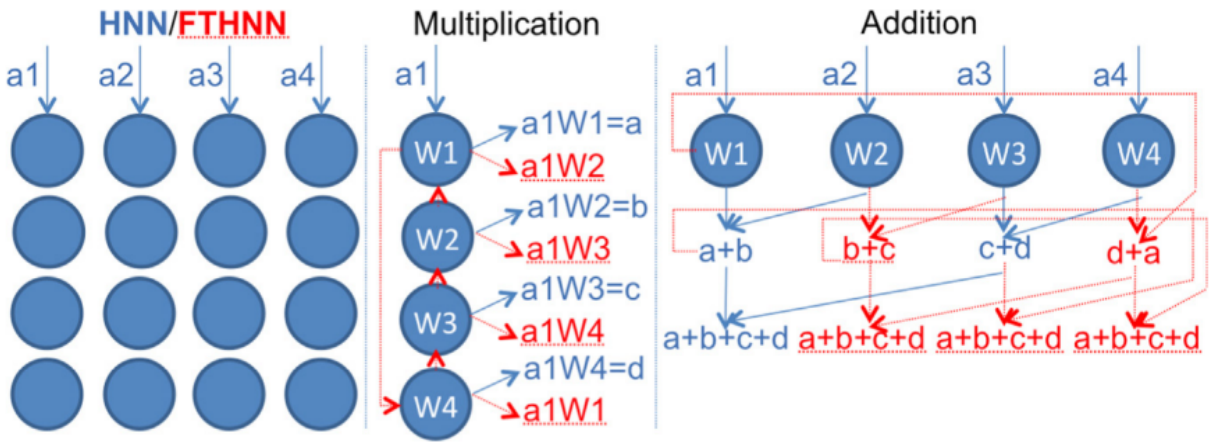

Figura 40: Comparativa entre HNN y FT-HNN

En la Figura 40 se presentan las diferencias de ambos diseños, donde en azul se muestran las partes propias de una HNN mientras que en rojo/subrayado se resalta lo particular de la FT-HNN. En el caso de la multiplicación del peso de una neurona por su valor de entrada, su resultado permanece en la red durante varios ciclos, debido a que es propagada a otras neuronas, como se muestra en la figura (redundancia temporal). Mientras que para la combinación de las salidas de neuronas mediante sumas, dichas salidas son propagadas a varias neuronas del siguiente nivel a la vez (redundancia espacial), en lugar de a solo a una.

Existen dos tipos de nodos en la FT-HNN (Figura 41):

- El primer tipo, es el **nodo serie tolerante a fallos (FTSN)**. Estos nodos son los que se sitúan en el primer nivel de la red, es decir, a la entrada. Dichos nodos multiplican su peso por el valor de su entrada y su salida es enviada a dos nodos del siguiente nivel. Estos nodos solo se utilizan en la primera etapa de la red.
- El segundo tipo de nodo es el llamado **nodo maestro (MN)**. Estos son utilizados en el resto de niveles y reciben dos entradas a la vez de otras neuronas, combinándolas mediante la suma, para luego ser multiplicados por su peso y enviar la salida a otras neuronas en los siguientes niveles.

Para una FT-HNN de $m*n$ nodos (donde m es el número de etapas y n el número de nodos por fila) se usarán $n*\log_2(n)$ nodos de tipo MN para una fila de n nodos. En la primera etapa, cada dos FTSN consecutivos alimentan un nodo MN de la siguiente etapa, mientras que para el resto de etapas, cada MN de la etapa m es alimentado por otros MN de la etapa anterior $m-1$, que están separados entre sí por $2*n$ nodos de distancia. Una máquina de estados es la encargada de controlar las iteraciones realizadas.

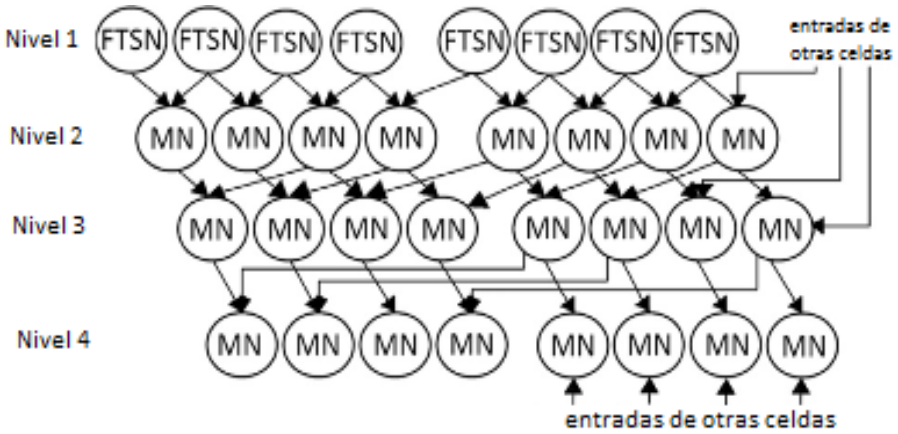

Figura 41: Arquitectura de una fila de 8 nodos de la FT-HNN

La técnica propuesta, como se verá en el siguiente capítulo, ofrece una protección frente a SEUs significativa mientras que solo incrementa el número de recursos adicionales en un 50,8% de media en lugar del >300% que implica el TMR.

4.3. DSPs tolerante a fallos (FT-DSPs)

En este apartado vamos a presentar el tercer método que se ha desarrollado para proteger circuitos. Se trata de un mecanismo inmediato para endurecer un diseño frente a SEUs y consiste en minimizar la parte vulnerable de dicho diseño mediante la utilización de DSPs. El uso de DSPs como alternativa a los CLBs supone una fiabilidad mayor del diseño debido a la estructura de los DSPs. Sin embargo, al ser hard cores implementados físicamente dentro de la FPGA, ofrecen menos flexibilidad en términos de configuración con respecto a los CLBs y, por lo tanto, no toda función lógica es apta para ser implementada con DSPs. Esto hace que solo ciertas familias de diseños o partes de éstos sean candidatos a ser implementadas en DSPs, es decir, exista una implementación funcionalmente equivalente a la de los CLBs.

El método de protección investigado explota esta característica y se basa en la utilización de DSPs infiriendo la parte del diseño implementable por éstos, además de centrarse en optimizar la utilización de recursos, ya que los DSPs son más escasos en comparación a los CLBs. La aplicación de este método de protección de circuitos ha sido

posible gracias a los profesores Marcos Sánchez-Elez e Inmaculada Pardines del departamento de Arquitectura de Computadores y Automática de la Universidad Complutense de Madrid [32]. La técnica ha sido estudiada en diseños que realizan operaciones aritméticas, ya que son buenos candidatos para ser implementados en DSPs. Esta técnica ha sido validada en los DSP48E de la familia de FPGAs Virtex-5 de Xilinx.

Para entender el algoritmo, primero es necesario introducir una serie de conceptos:

Primero, se define la clase de operación (o operator class en inglés, OC) como un tipo de operación aritmética, tal como la suma/resta, multiplicación o multiplicación acumulada y a cada operación se le asigna una plantilla genérica en código HDL para su implementación en un DSP.

A una implementación concreta de una OC en un DSP, se le conoce como instancia de clase (CI) y cada CI tiene asociado un código VHDL que utiliza bloques de DSP con un mapeo y configuración específico, esto es, la asignación de entradas/salidas, reloj, reset y configuración de registros internos. Esto supone que como mínimo, habrá tantas CIs como OCs existan. Sin embargo, una OC podrá tener varias implementaciones distintas en un DSP, o lo que es lo mismo, varias CIs asociadas. Al conjunto formado por estas CIs, se le llama conjunto de instancias de clase (CIS).

La metodología basada en DSPs maximiza el uso de este tipo de recursos a la hora de sintetizar un diseño, obteniendo una reducción en área y una mayor frecuencia de trabajo. La técnica se basa en [13], que trata de encontrar diferentes CIs que puedan ser implementadas en un mismo DSP. A raíz de esto, surge el concepto de ámbito de un CI, que define si distintos CIs pueden compartir el mismo recurso y ser ejecutados concurrentemente. El ámbito puede ser global, si un DSP se reserva para un único CI durante todo el proceso o local, permitiendo a un DSP ser usado por diferentes CIs.

Además, el CI puede ser de tipo protegido o no protegido. Esta clasificación depende del usuario ya que debe indicar el tipo de un CI. Diferentes CIs que no se ejecuten concurrentemente podrán ser de tipo no protegido y serán unidos en un CI para minimizar los espacios de tiempo libre entre instancias en un intervalo de tiempo.

Un SI (slot instance) es el código VHDL que implementa una operación en un DSP, por lo que minimizando el número de SIs se minimizarán los recursos utilizados, al compartir

los DSPs para distintas operaciones. Esto se consigue configurando de distintas maneras un mismo DSP en tiempo de ejecución. Sin embargo, si el ámbito es declarado como protegido, esa CI tendrá asociado un SI y por lo tanto, tendrá un DSP cuya configuración será fija. De esta manera dos CIs declarados como protegidos solo pueden compartir el mismo DSP si no son concurrentes y usan el DSP con la misma configuración.

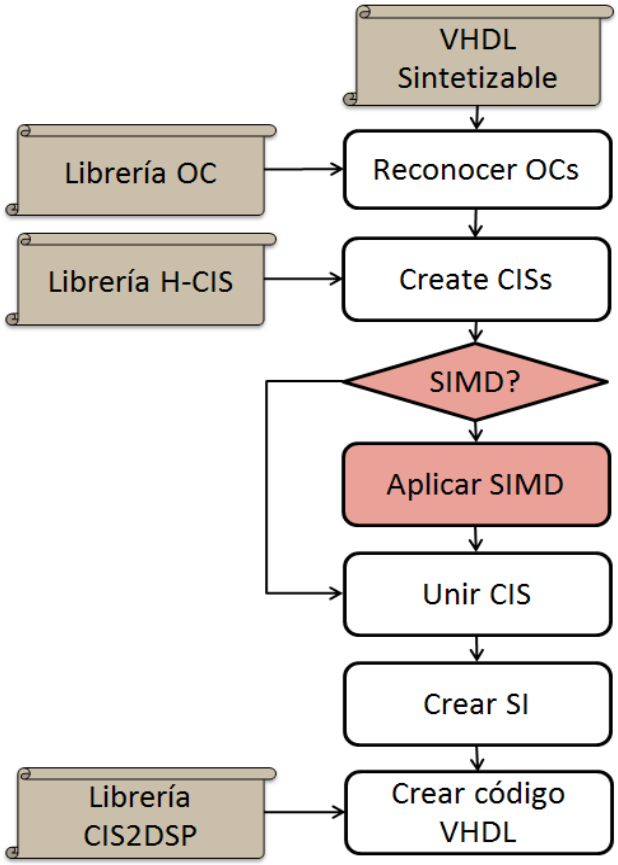

Figura 42: Grafo de flujo para la técnica FT-DSP

El flujo de diseño para esta metodología viene ilustrado en la Figura 42. A dicha metodología, le ha acompañado el desarrollo de una herramienta que automatiza estos pasos y facilita la generación del código reforzado. Con los conceptos necesarios ya introducidos, se explica paso a paso a continuación:

1. Reconocer OCs: primero se deben reconocer las operaciones consideradas como OCs en el código HDL original y utilizando una librería de OCs, sustituirlas por otra plantilla HDL equivalente de la operación optimizada. Al comenzar el proceso de

implementación, los diseñadores deben decidir qué operaciones se almacenarán en la librería de OCs y serán reconocidas en el código HDL original. Entonces se hará una clasificación de OCs dependiendo de su tamaño y número de señales involucradas.

2. Crear CISs: el siguiente paso consiste en traducir las OCs reconocidas en implementaciones concretas para la FPGA destino. Por lo tanto se generan las posibles CIs según los OCs detectados. El objetivo consiste en desarrollar y probar distintas CIs reforzadas para ver cuáles son más efectivas en la protección frente a SEUs. Probada la efectividad de una OC, se añade a la librería de OCs. Por último, la herramienta desarrollada, ofrece la posibilidad de usar los DSP48E en modo SIMD (single instruction multiple data por sus siglas en inglés) para algunas operaciones aritméticas. Esta opción permite al usuario indicar si se quiere usar este modo a la hora de crear los CIs.
3. Unir CISs: para cada OC se selecciona el mejor CI y se descarta el resto. El conjunto resultante de esta etapa es un mapeo uno a uno de todas las operaciones encontradas en DSP48E.
4. Crear SIs: el código HDL de cada CI generado por la herramienta se realiza en la etapa de "Crear SIs". Este código define la configuración del bloque DSP y sus salidas y entradas implicadas. Después de la inclusión de las SIs y la conexión de sus señales con el resto del sistema, el resultado es un HDL sintetizable.
5. Crear VHDL optimizado: las entradas o salidas de un SI pueden cambiar en tiempo de ejecución dependiendo de sus ámbitos, como resultado de unir varios CIs en un DSP. El código a generar tiene en cuenta esto y añade un controlador que gestiona esta multiplexación de señales, considerando el ámbito de los distintos CIs unidos en el mismo SI ahorrando de esta manera recursos.

Esta técnica, junto con la herramienta desarrollada para su implementación, ofrece una enorme flexibilidad a la hora de hacer más robusto un diseño que contenga operaciones aritméticas. Como se verá más adelante en el capítulo 5, permite alcanzar un buen compromiso entre utilización de DSPs y CLBs en función de los DSPs disponibles (mucho más escasos que los CLBs), protección del diseño y sobrecarga de recursos extra debido a la redundancia.

Capítulo 5: Resultados experimentales

En este capítulo se detallan los resultados experimentales obtenidos por las distintas metodologías de inyección de errores y protección de circuitos presentadas en esta tesis doctoral.

En primer lugar, en la sección 5.1. se presentan los resultados de inyección de errores en la memoria de configuración a un conjunto de circuitos patrón utilizados como referencia. Los circuitos seleccionados son los pertenecientes al benchmark ITC'99 y se explican cada uno en detalle en la siguiente sección. La inyección de errores sobre este conjunto de circuitos ha servido además como ejercicio para la depuración y mejora de la herramienta NESSY.

Para complementar los resultados sobre la memoria de configuración de los diseños del benchmark ICT'99, se ha realizado también una inyección de errores en los flipflops (sección 5.2.). Además, en este experimento se ha decidido añadir un filtro digital al conjunto de circuitos bajo test, en concreto un circuito de tipo "Fast Forward Equalization" (o FFE), que es un filtro particular de la categoría de filtro "Finite Impulse Response" (o FIR). Este tipo de filtros hacen uso de un gran número de flipflops, siendo un buen candidato para este tipo de inyección de errores.

A continuación, en la sección 5.3. , se presentan e interpretan los resultados obtenidos para la inyección de errores en la memoria de configuración al aplicar TMR en sus distintas variantes de estos circuitos y se realiza una comparativa de cómo de eficaces son cada una de ellas.

Las dos últimas secciones de este capítulo se centran en los dos métodos investigados en esta tesis centrados en las redes neuronales de tipo Hopfield y en el uso de DSPs como método de refuerzo. El filtro FFE, utilizado en la inyección de errores sobre flipflops es también buen candidato para ser implementado con DSPs por su estructura regular y con

abundantes operaciones aritméticas. Por ello, en la sección 5.5. , antes de abordar en la técnica de protección mediante el uso de DSPs, se hace un estudio comparativo sobre las diferencias entre este filtro cuando se implementa con CLBs y cuando se hace utilizando DSPs. Con ello se pretende analizar las ventajas estructurales de los DSPs frente a CLBs. Además, se hace uso de un filtro FIR genérico de una única etapa, para analizar la equivalencia en funcionalidad implementable por un DSP frente a CLBs, lo cual es un factor a considerar cuando se habla de robustez frente a SEUs.

5.1. Inyección de errores en la memoria de configuración: benchmark ITC99

Para la obtención de los resultados experimentales de esta sección, se ha utilizado el conjunto de diseños recogidos en el benchmark ITC'99. Este benchmark ha sido desarrollado por el grupo CAD de la universidad Politécnica de Torino [9]. Estos diseños cubren un amplio espectro de circuitos digitales típicos. Se destacan como principales características:

- Código HDL completamente sintetizable.
- Independencia de directivas de compilador.
- Utiliza librerías del estándar IEEE.
- Circuitos completamente síncronos.
- Reloj con una única fase conectado directamente a los elementos de memoria.
- Reset global para todos los diseños.
- Sin memorias internas (exceptuando los bancos de registros).
- No hay buferes tri-estado.
- No hay conexiones OR-cableadas.

Los resultados experimentales presentados en esta sección son resultado de una inyección de errores en la memoria de configuración cuando estos circuitos se implementan en una FPGA Virtex-5. La Tabla IV muestra el número de líneas y procesos del código

VHDL así como los recursos utilizados por cada diseño:

- CELDAS: número de puertas lógicas
- NI: número de entradas
- NO: número de salidas
- FF: número de flipflops

NOMBRE	VHDL		RECURSOS			
	LINEAS	PROC	CELDAS	NI	NO	FF
b01	110	1	49	2	2	5
b02	70	1	28	1	1	4
b03	141	1	160	4	4	30
b04	102	1	737	8	11	66
b05	332	3	998	1	36	34
b06	128	1	56	2	6	9
b07	92	1	441	1	8	49
b08	89	1	183	9	4	21
b09	103	1	170	1	1	28
b10	167	1	206	11	6	17
b11	118	1	770	7	6	31
b12	569	4	1,076	5	6	121
b13	296	5	362	10	10	53

Tabla IV: Recursos de los diseños ITC'99

Para poder realizar pruebas realistas, es necesario generar un testbench significativo. Un testbench se considera significativo si explota toda la funcionalidad del circuito, esto es, explora todos los estados posibles del código HDL. Para ello es necesario tener un conocimiento profundo del diseño que se quiere tratar. El grupo CAD ofrece una breve descripción de la funcionalidad de cada diseño, también mostrada en la Tabla V.

Nombre	Funcionalidad
b01	FSM que compara flujos serie
b02	FSM que reconoce números BCD
b03	Árbitro de recursos
b04	Calcula máximo y mínimo
b05	Genera contenido de una memoria
b06	Manejador de interrupciones
b07	Cuenta puntos en una línea recta
b08	Encuentra inclusiones en secuencias de números
b09	Convertidor serie-serie
b10	Votador
b11	Cifrador de cadenas variable
b12	Juego de adivinar la secuencia
b13	Interface a sensores “meteo”

Tabla V: Descripción funcional de los diseños ITC'99

Para la generación de los testbench ha sido necesario la Tabla V junto a un análisis profundo del código HDL para estudiar la estructura interna y funcionamiento de cada diseño y así poder diseñar un testbench que recorra todos los estados posibles del circuito, así como estimular todas sus señales.

Los resultados de las inyecciones se presentan en la Tabla VI, donde se muestra el tiempo requerido para la inyección, el total de errores inyectados y cuántos de esos errores tienen efecto en la salida. Los tiempos de inyección han sido especialmente importantes en este experimento, al servir como métrica para comparar NESSY con el resto de herramientas del estado del arte. Destacar que la inyección de un error con NESSY es de coste constante en el tiempo, con independencia del tamaño total del circuito. El tiempo por inyección es el tiempo necesitado únicamente para la modificación del bitstream, mientras que el tiempo por evaluación también incluye el tiempo requerido para el procesado de la salida del circuito. Este último tiene una dependencia en el tiempo lineal con el tamaño del testbench, aunque se puede apreciar que las diferencias entre los distintos diseños son despreciables si las comparamos con el tiempo que lleva en sí la evaluación (90 µs es la máxima desviación frente a una media de 10,38 ms por evaluación).

Circuito	Tiempo requerido			Inyección de errores		
	Total	Por evaluación	Por inyección	SEUs inyectados [#]	SEUs que afectan al diseño [#]	Errores [%]
b01	7m, 57s	10,35ms	98,13µs	46080	39168	1,35%
b02	7m, 57s	10,35ms	98,13µs	46080	39864	0,52%
b03	7m, 57s	10,35ms	98,13µs	46080	32256	3,73%
b04	16m, 8s	10,50ms	98,13µs	92160	78336	7,56%
b05	23m, 51s	10,35ms	98,13µs	138240	115200	0,05%
b06	15m, 54s	10,35ms	98,13µs	46080	23040	1,71%
b07	7m, 57s	10,35ms	98,13µs	46080	55296	0,66%
b08	7m, 57s	10,37ms	98,13µs	46080	27648	6,19%
b09	7m, 57s	10,35ms	98,13µs	46080	32256	4,04%
b10	7m, 57s	10,37ms	98,13µs	46080	29952	6,23%
b11	16m, 7s	10,49ms	98,13µs	92160	57600	7,62%
b12	24m, 4s	10,45ms	98,13µs	138240	124416	22,58%
b13	8m, 1s	10,44ms	98,13µs	46080	43776	6,63%

Tabla VI: Resultados de inyección de erorres en la memoria de configuración para los diseño de ITC'99

En los resultados referentes a la inyección de errores (columnas 5-7), se distingue entre el número de total de SEUs inyectados en la memoria de configuración, determinado por el área de la RPR indicada por el usuario para la instanciación del diseño (la unidad mínima de configuración distinguible es un bloque de recursos, bloque-CLB en este caso, con 46080 bits, para más detalles consultar el Apéndice I) y los SEUs efectivos que modifican bits de configuración de recursos de la RPR utilizada por el diseño. Si el usuario ha escogido un área de la RPR demasiado grande, se realizarán muchas inyecciones de errores no útiles, al no estar afectando la mayor parte de ellas al comportamiento del circuito y en consecuencia, la campaña de inyección de errores requerirá más tiempo. En la columna 7 se muestra el ratio de errores/SEUs donde aquí se tiene en cuenta el número de SEUs efectivos, es decir, dentro de los recursos usados para el circuito.

5.2. Inyección de errores en flipflops: benchmark ITC99 y FFE

La inyección de SEUs sobre flipflops se ha realizado sobre el mismo conjunto de circuitos utilizados en la inyección de errores en la memoria de configuración de la sección anterior. Además, como ya se ha mencionado, se ha realizado un experimento adicional sobre un circuito complejo de tipo "Fast Foward Equalization" (FFE).

Los resultados aquí expuestos son complementarios a los de dicha sección anterior, dando entre ambos una cobertura del 100% de los elementos de memoria de dichos diseños y, por lo tanto, quedando caracterizada por completo su vulnerabilidad frente a SEUs. Para este experimento se ha optado por una inyección estadística en vez de una inyección exhaustiva.

Cualquier test estadístico de inyección de fallos queda definido principalmente por las siguientes tres variables:

1) Número de celdas de memoria susceptibles a errores.
2) Número de vectores de test de entrada.
3) Número de ciclos durante los cuales el circuito está en funcionamiento.

Por lo tanto, se deberán de tener en cuenta estas variables para poder garantizar una cobertura completa con respecto a la funcionalidad del diseño testeado. Sin embargo, incluso para circuitos pequeños, un test exhaustivo puede suponer una explosión combinatoria de casos a probar al combinar estas 3 variables, requiriendo una capacidad de cómputo muy alta. Las inyecciones presentadas a continuación tienen como características que: a) las pruebas realizadas inyectan errores en todos los flipflops del diseño tratado, b) los vectores usados han sido los provistos por los desarrolladores del benchmark [48] y c) NESSY permite inyectar un SEU en cualquier ciclo de reloj.

De [37] obtenemos la ecuación de muestreo de error:

$$n = \frac{N}{1 + e^2 \cdot \left(\frac{N-1}{t^2 \cdot p \cdot (1-p)}\right)}$$

Dicha ecuación, resuelve el número de SEUs necesarios a inyectar en un sistema con N casos de pruebas para un nivel de confianza t, una estimación del porcentaje de SEUs que darán error p y un margen de error e determinados, donde N en este caso es:

$$N = numff \cdot numciclos$$

Como primer experimento, se ha utilizado un nivel de confianza del 90% y un margen de error del 5% dando lugar a, al menos, 73,21 SEUs/FF para b01 por lo que se ha optado por aproximar a 74 SEUs/FF para los circuitos b01-b12 debido a su similitud en tamaño. Es importante señalar que este número es proporcional al número de flipflops que tenga cada diseño testeado y el número de ciclos de reloj que contenga su testbench. Para el caso del filtro FFE, el valor obtenido es de 228 SEUs/FF.

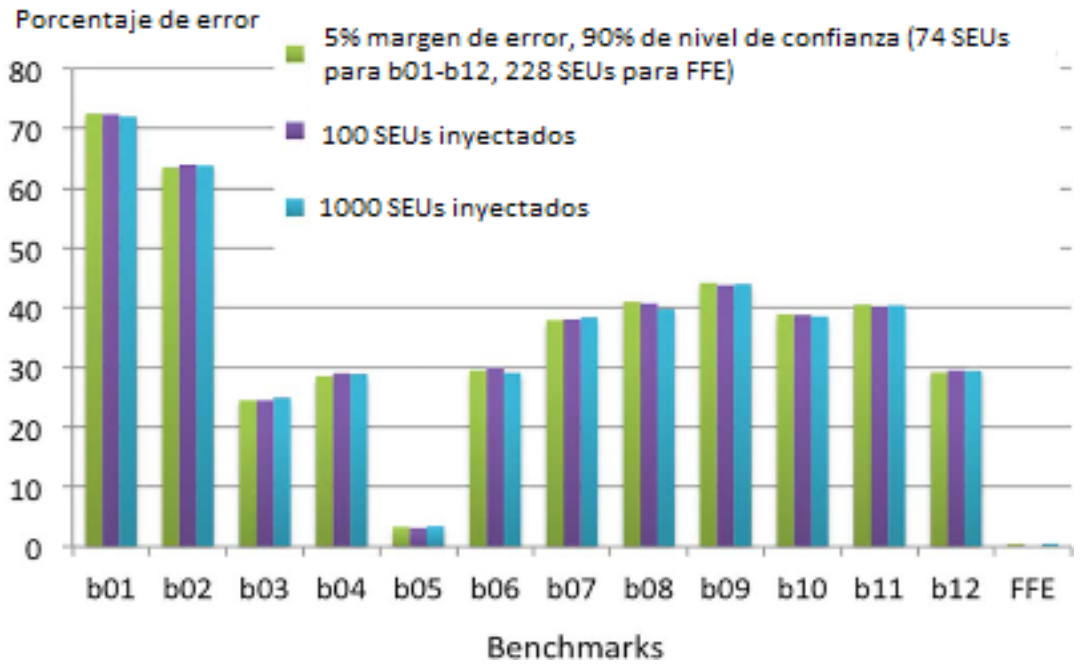

Figura 43: Porcentaje de fallos para distintas inyecciones de errores por FF

De los SEUs emulados, se ha obtenido el porcentaje que causan error a las salidas. Esto se muestra en la columna izquierda de cada uno de los experimentos mostrados en la Figura 43. Para validar la ecuación (1), en cada experimento se han añadido otras dos columnas que hacen referencia a experimentos donde se han inyectado 100 SEUs/FF y 1000 SEUs/FF respectivamente. En todas ellas, se puede observar que el porcentaje de SEUs que producen

error a las salidas del circuito son muy similares.

Como se aprecia en la Figura 43 , desde un punto de vista estadístico se puede concluir que los resultados tienen la misma validez que una inyección de fallos exhaustiva. De hecho, para cada experimento el gradiente de la recta de regresión que mejor se ajusta a los tres resultados calculados es, en el peor de los casos, 0,00027, lo cual es despreciable.

A partir de estos resultados, se concluye que la herramienta y la metodología utilizada son extrapolables a circuitos mayores donde una inyección de errores exhaustiva es inviable.

5.3. TMR

En esta sección se exponen los resultados de inyectar errores para las distintos versiones de TMR explicadas en el capítulo 4.1. Puesto que la técnica de protección aplicada es el TMR, se han utilizado solo los diseños del benchmark ITC'99 donde la metodología del TMR se puede aplicar, siendo éstos el subconjunto formado por b01-b13. En esta campaña de inyección de errores se excluyen al resto, siendo los circuitos b14-b16 diseños basados en procesador, en los cuales no se puede aplicar TMR y, por otro lado, los circuitos b16-b22, que son varias copias de diseños anteriores (b01-b13) y no aportan ninguna funcionalidad nueva.

En la Tabla VII se muestran los resultados de las inyecciones de errores. La tabla muestra el porcentaje de errores no enmascarables, esto es, que tienen efecto a la salida del sistema. Cada columna hace referencia a la versión del diseño según su versión de TMR aplicada. Estas versiones son las ya introducidas en el capítulo 4:

1. TMR simple (TMR1)
2. TMR + aislamiento (TMR2)
3. TMR + aislamiento + votador blindado (TMR3)

Circuito	No TMR	TMR 1	TMR 2	TMR 3
b01	1,146%	0,157%	0,040%	0,021%
b02	0,414%	0,061%	0,028%	0,014%
b03	2,609%	0,322%	0,043%	0,00%
b04	6,426%	1,024%	0,057%	0,003%
b05	0,041%	0,026%	0,013%	0,000%
b06	0,818%	0,166%	0,084%	0,010%
b07	0,143%	0,073%	0,033%	0,000%
b08	3,715%	0,361%	0,072%	0,024%
b09	2,826%	0,107%	0,018%	0,005%
b10	4,049%	0,726%	0,068%	0,001%
b11	4,765%	0,240%	0,052%	0,010%
b12	2,093%	0,234%	0,028%	0,002%
b13	6,293%	0,483%	0,083%	0,003%

Tabla VII: Resultados de inyección de errores en la memoria de configuración a diseños de ITC'99 con TMR

La técnica de TMR se refina progresivamente en las distintas versiones de TMR aplicado, como se concluye a partir de los resultados de la Tabla VII.

En el TMR simple (columna 3), donde las tres copias del circuito no son realmente independientes unas de las otras debido a optimizaciones de las herramientas de síntesis, se reduce considerablemente la tasa de errores respecto a los diseños sin TMR (columna 2). Esto permite que en el peor caso, de media, el 98,9% de los SEUs no tengan efecto en la salida del sistema y por lo tanto, no tengan consecuencias en el correcto funcionamiento de los diseños. Por lo general los circuitos grandes (b08-b13) se benefician más de esta técnica.

La diferencia entre el TMR simple con respecto al TMR con aislamiento (columna 4), como ya se ha explicado en la sección 4.1. consiste en que las zonas del circuito inferidas como comunes a las tres instancias (véase señales constantes y cableadas a VCC o GND)

también se triplican al igual que el resto de recursos. Por lo tanto, la disminución del número de errores con respecto al TMR simple, se debe a que esa parte de la lógica es también inmune a los SEUs.

Por último, la mejora aportada por el TMR + aislamiento + votador blindado (columna 5) con respecto al TMR + aislamiento es el blindaje de la parte común (y por lo tanto, vulnerable a SEUs), es decir, el votador, que no está triplicado. Sin embargo, en este tipo de TMR aún se producen errores. Estos se deben a SEUs que afectan a la matriz de conexionado de las entradas, de las salidas o de señales críticas del sistema (como el reset), ya que en estos casos el efecto del SEU se propaga a las tres copias del circuito.

5.4. Inyección de errores en redes neuronales de tipo Hopfield

En esta sección se presentan las pruebas realizadas en la implementación de una HNN tolerante a fallos (FT-HNN) y un estudio experimental de su robustez frente a SEUs, SETs y stuck-ats. Un stuck-at, es un tipo de error en el que la celda de memoria se queda fijada a un valor lógico determinado, o '0' ó '1'. En el estudio se incluyen los stuck-ats porque las inyecciones se han realizando utilizando dos herramientas: por un lado, se ha utilizado NETFI [43] la cual emula SEUs, SETs y stuck-ats en los flipflops. Es una funcionalidad que NESSY no implementa y por lo tanto, ambas herramientas se complementan. Por otro lado, se ha utilizado NESSY, para inyectar errores en la memoria de configuración. Para la solución robusta FT-HNN, descrita en la sección 4.2. y en mayor detalle en [44], se ha comparado con su versión original, HNN y otra versión basada en TMR, HNN-TMR.

En primer lugar, se ha evaluado el consumo de recursos de estas tres variantes de HNN. Los resultados se muestran en la Figura 44, donde la gráfica a) muestra los resultados obtenidos para NETFI y la b) muestra los de NESSY. En la figura también se distinguen los recursos utilizados según qué tipo de FPGA se haya utilizado para su implementación. Los dispositivos utilizados han sido una Virtex 4 XC4VLX40, en el caso de NETFI y una Virtex 5 XC5VLX110T, en el caso de NESSY.

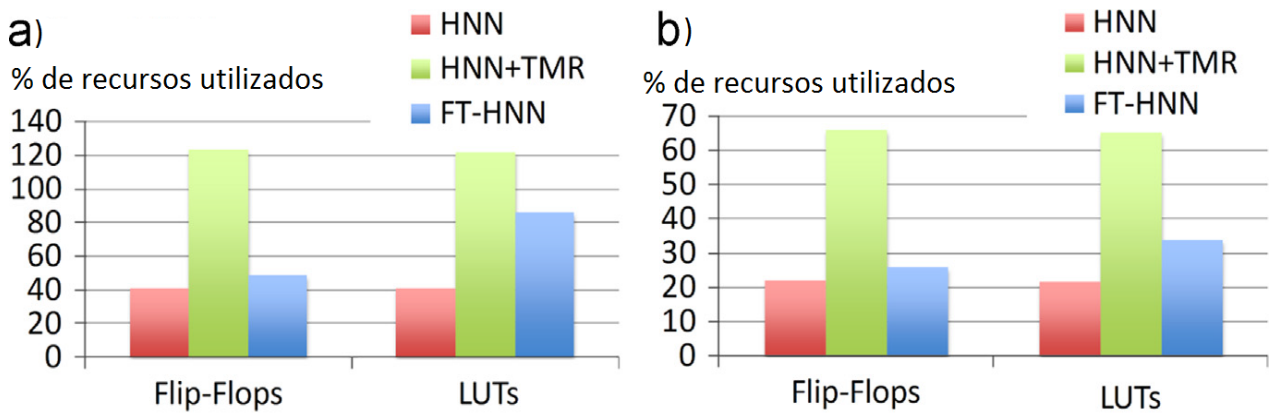

Figura 44: Consumo de recursos para las dos versiones de HNN

De media, la FT-HNN incrementa en 50,8% los recursos necesarios con respecto a la HNN original. Esto sin embargo, está lejos del 300% requerido por el diseño basado en TMR.

En segundo lugar, se han realizado inyecciones de errores sobre la HNN y la FT-HNN utilizando NESSY y NETFI. Para la inyección de errores con NETFI, los fallos obtenidos se han clasificado según sean:

- "Silenciosos", cuando el fallo no ha tenido efecto en el resultado. Es decir, es el equivalente en NESSY a un SEU que no produce error.
- "Errores", cuando la salida es distinta a la del circuito golden.
- "Timeouts", cuando debido a un SEU no se produce ninguna salida, es decir, la salida no varía con cambios en los valores de entrada.
- "Convergencias", cuando la salida es la esperada, pero esta se obtiene después del tiempo de ejecución esperado.

Los resultados se muestran en la Tabla VIII:

	Tipo de fallo	[#] inyecciones	Resultados		
			Errores	**Timeouts**	**Convergencias**
HNN	SET	81,831	79 (0.10%)	1658 (2.26%)	10,865 (13.28%)
	SEU	125,676	110 (0.09%)	2623 (2.09%)	15,195 (12.09%)
	Stuck-at-0	190,509	9566 (5.02%)	8417 (4.42%)	19,393 (10.18%)
	Stuck-at-1	98,316	5444 (5.54%)	3076 (3.13%)	7659 (7.79%)
FT-HNN	SET	137,801	20 (0.01%)	6 (0.004%)	4789 (3.48%)
	SEU	149,198	44 (0.03%)	4 (0.002%)	4193 (2.81%)
	Stuck-at-0	140,516	3897 (2.77%)	557 (0.40%)	1667 (1.19%)
	Stuck-at-1	151,803	4318 (2.84%)	337 (0.22%)	2486 (1.63%)

Tabla VIII: Resultados de la inyección de fallos sobre la HNN y FT-HNN usando NETFI

Como se observa en la tabla, los porcentajes de errores frente a SEUs y SETs se reducen drásticamente para el caso de la FT-HNN. Como se puede apreciar, los fallos críticos (véase errores + timeouts) se reducen de media del 2,36% al 0,14% en el caso de SEUs, del 2,18% al 0,032% para SETs, del 9,44% al 3,07% para stuck-at-0s y por último del 8,67% al 3,06% para stuck-at-1s. Los fallos de convergencia también se reducen. Por lo tanto, un aumento en recursos que refuerzan por diseño la HNN está justificado debido a su mayor robustez frente a SEEs, especialmente frente a SEUs. Un análisis en profundidad muestra que la máquina de estados de la FT-HNN (ver sección 4.2.) es la parte más vulnerable del diseño.

Para obtener la vulnerabilidad de SEUs en la memoria de configuración se ha utilizado NESSY y los resultados obtenidos se muestran en la Figura 45, donde en este caso, también se ha evaluado la versión basada en TMR. Como era de esperar, la sensibilidad frente a SEUs de la red reforzada mediante triple redundancia es menor que la HNN sin protección al igual que pasa con la reforzada por diseño, FT-HNN.

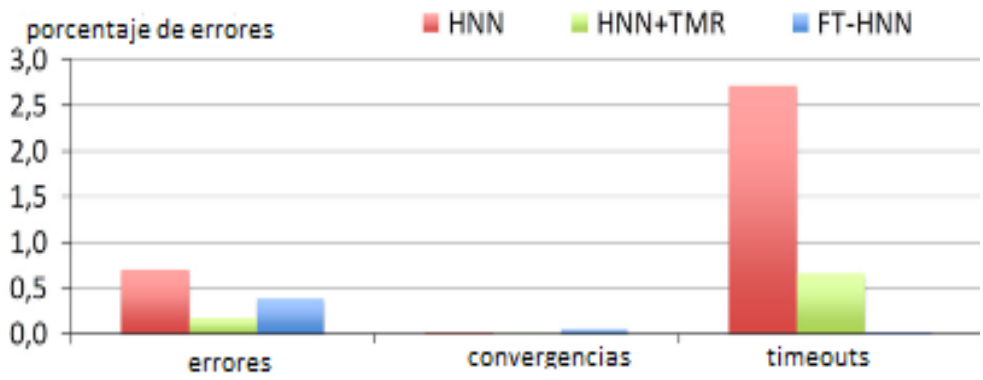

Figura 45: Resultados de inyección de errores usando NESSY

La primera conclusión que se obtiene es que la versión de HNN-TMR mejora en todos los aspectos la robustez del diseño, reduciendo significativamente el porcentaje de errores en todas sus categorías. Sin embargo, no hay que perder de vista la necesidad de triplicar todo el diseño y por lo tanto, usar al menos un 300% de recursos, lo que llevado a diseños grandes puede hacer que esta técnica sea inviable.

Por otro lado, se puede observar que la versión FT-HNN incrementa la sensibilidad frente a SEUs respecto a la HNN-TMR no reforzada en 0,386% pero a cambio, prácticamente se eliminan los errores de timeout (se reducen a 0,027%). Los fallos de convergencia también se incrementan ligeramente del 0,024% al 0,053% con respecto a la HNN+TMR. Sin embargo, en términos generales (errores + timeouts + convergencias), la FT-HNN mejora a la HNN sin protección en un factor de 7,41 (0,16% frente a 1,15%) de media; mientras que la HNN+TMR hace lo propio en un factor de sólo 2,47 (0,16% frente al 0,28%) de media. Por lo tanto, la conclusión que se puede sacar de este experimento es que la versión FT-HNN supone una sobrecarga del 50,8% en términos de recursos necesarios adicionales, pero a cambio es 7,41 veces mas robusta frente a SEUs que el diseño sin proteger. Si esta comparación se realiza con la versión con HNN+TMR, se tiene que triplicando los recursos, es decir, una sobrecarga del 300% en área, el diseño tiene un factor de protección de 2,47 con respecto al diseño sin proteger.

5.5. Inyección de errores en circuitos con DSPs

Finalmente en esta sección se presentan los resultados de inyección de errores en circuitos que utilizan los DSPs de las FPGAs Virtex-5. Para ello se han realizado dos experimentos, en primer lugar se inyectan errores en filtros digitales, al ser candidatos idóneos para la utilización de DSPs. Con esta prueba se pretende cuantificar las ventajas que ofrecen los DSPs frente a los CLBs debido a su estructura. En segundo lugar, se han realizado experimentos sobre diseños que han aplicado la técnica de protección desarrollada en la sección 4.3. que se basa en el análisis inteligente de un diseño para sustituir partes de éste implementadas en CLBs por DSPs, sin afectar a su funcionamiento lógico.

5.5.1. Filtros FFE y FIR genérico

En primer lugar se ha querido comprobar si el hecho de utilizar DSPs con respecto a CLBs mejora la fiabilidad del diseño. Para ello se ha utilizado para realizar inyecciones de errores el circuito FFE, que como se ha comentado es buen candidato a ser implementado con DSPs por su estructura. El filtro FFE es un circuito complejo que consta de varias etapas consecutivas de filtrado. En los experimentos, se han inyectado SEUs en diferentes versiones de este circuito: con 2, 4 y 8 etapas de filtrado, así como cuando se implementa utilizando DSPs y CLBs o solo CLBs. Para conocer la relación que existe entre la cantidad de lógica implementable por un DSP y un CLB se ha usado un filtro FIR genérico de una única etapa. Este diseño se ha escogido como circuito patrón, ya que utiliza toda la funcionalidad que ofrece un DSP para su implementación ya que su funcionalidad está descrita por la ecuación:

$$y(n) = a(n) * b(n) + c(n)$$

Se ha realizado esta prueba para obtener un cota superior en la reducción del número de

bits de configuración de los circuitos al implementarlos utilizando DSPs, ya que éstos pueden implementar la misma cantidad de lógica que un CLB, pero con menos bits de configuración cuando se aprovecha su arquitectura.

En la Tabla IX se muestran los resultados obtenidos para el filtro FFE en sus distintas configuraciones.

Circuito	**# Inyecciones totales**	**Bloques-CLB**		**Bloques-DSP**		**% Errores totales**
		# Inyecciones efectivas	**# Errores**	**# Inyecciones efectivas**	**# Errores**	
FFE (2) sin DSPs	276480	258048	7892	N/A	N/A	3,06%
FFE (2) con DSPs	128000	43776	1763	17920	838	4,22%
FFE (4) sin DSPs	555520	516096	15954	N/A	N/A	3,09%
FFE (4) con DSPs	128000	82944	2659	35840	1560	3,55%
FFE (6) sin DSPs	1013760	963072	52209	N/A	N/A	5,42%
FFE (6) con DSPs	256000	154368	7529	71680	4634	5,38%

Tabla IX: Resultados experimentales de la inyección de errores en diferentes versiones de FFE

Las filas 1, 3 y 5 presentan la versión sin DSPs para los filtros FFE con 2, 4 y 8 etapas respectivamente. Las filas 2, 4 y 6 hacen lo propio con las versiones con DSPs correspondientes. La columna 2 de la tabla muestra el número de inyecciones realizadas en toda la región reconfigurable reservada para el circuito, mientras que las columnas 3 y 5 sólo contabilizan las inyecciones que afectan a recursos realmente utilizados por el circuito bajo test (que aparecen identificadas como “inyecciones efectivas” en la tabla). A continuación, las columnas 4 y 6 distinguen los errores detectados en función de los bloques de recursos a los que afectan (bloques-DSP o bloques-CLB). Por último, la columna 7 muestra el porcentaje de errores en los distintos circuitos. En el caso de las versiones con

DSPs, se ha obtenido una media ponderada y estos porcentajes se corresponden a los errores en CLBs y DSPs, corregidos en función de su peso en el total de los errores. Los porcentajes han sido calculados con respecto al número de inyecciones efectivas realizadas, para no desvirtuar los resultados debido a mapear el circuito en una RPR de mayor tamaño que el circuito a testear.

Con los datos que se muestran en la Tabla IX se pueden comparar las distintas implementaciones del FFE y se observa que los porcentajes de error de las versiones con y sin DSPs son muy similares. Esto indica que los circuitos implementados con DSPs no son más resistentes a la radiación cósmica que los CLBs a nivel estructural. Sin embargo, las implementaciones equivalentes con DSPs tienen la ventaja adicional de ser más eficientes al implementar lógica, esto es, implementa la misma funcionalidad utilizando menos bits de configuración.

Las Tabla X y XI clasifican los errores encontrados en los circuitos evaluados en función del tipo de recurso afectado.

Circuito	**Instancia**		**E/S**		**Conexionado**		**Total**	
	# CLBs	**# Errores**	**# CLBs**	**# Errores**	**# CLBs**	**# Errores**	**# CLBs**	**# Errores**
FFE (2) con DSPs	17	1583	2	76	18	104	37	1763
FFE (2) sin DSPs	110	7824	2	57	2	11	114	7892
FFE (4) con DSPs	29	2497	7	137	2	25	38	2659
FFE (4) sin DSPs	220	14825	4	180	14	949	238	15954
FFE (6) con DSPs	58	6246	9	790	13	493	80	7529
FFE (6) sin DSPs	413	50187	5	309	22	1713	440	52209

Tabla X: Clasificación de resultados para inyección de errores en bloques-CLB del filtro FFE

Circuito	Instancia		E/S		Conexionado		Total	
	# DSPs	# Errores	#DSPs	# Errores	#DSPs	# Errores	#DSPs	# Errores
FFE (2) con DSPs	4	838	0	0	0	0	4	838
FFE (4) con DSPs	8	1560	0	0	0	0	8	1560
FFE (6) con DSPs	16	4634	0	0	0	0	16	4634

Tabla XI: Clasificación de resultados para inyección de errores en bloques-DSP del filtro FFE

NESSY permite distinguir el tipo de error según el bit de la memoria de configuración afectado corresponda a:

- Lógica: SEUs que afectan a bloques de recursos utilizados para implementar la lógica del circuito.
- E/S: estos errores son los debidos a los bits que afectan el rutado de las entradas y salidas. Cualquier circuito debe tener un conexionado que haga de interfaz entre su lógica y el resto del sistema o los pines de la FPGA. Un error de E/S se produce cuando un SEU afecta a dichas señales.
- Conexionado: errores relacionados con bloques de recursos lógicos no incluidos en ninguna de las dos categorías anteriores. Estos bloques de recursos no implementan ninguna lógica del circuito, pero sí se utiliza alguna de sus matrices de conexionado para rutar señales. Al igual que los errores de E/S, esta categoría de errores es inevitable debido a que el algoritmo de rutado de Xilinx decide utilizar ciertas matrices de conexionado asociadas a un bloque de recursos, para rutar las señales utilizadas por bloques de recursos vecinos. Para más información consultar Apéndice I.

Por otro lado, la implementación del circuito patrón de tipo FIR con y sin DSPs ha permitido averiguar que para las versiones implementadas sin utilizar DSPs, cada bloque-

DSP se sintetiza utilizando 106 bloques-CLB. Como son necesarios 8.960 bits de configuración para configurar un bloque-DSP, y 106 * 2.304 = 244.224 bits para configurar 106 bloques-CLB, el tamaño de la memoria de configuración necesaria para configurar estos recursos lógicos se reduce en un 96,33%. En el caso de los circuitos testeados en esta sección, la reducción en la información de configuración ha sido de un 76,53% de media, ya que hay parte del circuito que está implementada en CLBs. Por lo tanto, podemos concluir que un circuito es más resistente a los SEUs si utiliza DSPs debido a su mayor eficiencia en términos de área y existe una menor probabilidad de que una partícula de alta energía incida sobre éste y produzca un error.

5.5.2. FT-DSP

Finalmente se ha realizado una campaña de inyección de errores a circuitos donde se ha aplicado el uso inteligente de DSPs (introducido en 4.3.) para hacer más robusto un diseño que realice operaciones aritméticas. Esta campaña de inyección de errores es necesaria debido a que la sustitución de una parte de la lógica del circuito por otra (CLBs por DSPs), modifica el mapa de bits de configuración del circuito y, por lo tanto, su sensibilidad frente a SEUs. En función de qué región del diseño se haya decidido implementar sobre DSPs y cuál no, dará lugar a distintas versiones. Además, se pretende mejorar la robustez aportada por la técnica explicada en la sección 4.3. mediante la combinación con el TMR. Las distintas variables a la hora de implementar un diseño entonces son:

1. Tipo de operación aritmética a implementar.
2. Operaciones aritméticas y votador compartiendo recursos o no.
3. Aplicación de TMR, puede ser: a la operación aritmética, a la operación y al votador o no aplicar TMR.
4. Utilización de CLBs o DSPs para las distintas partes del diseño.

Las distintas operaciones aritméticas elegidas para realizar la inyección de errores son, la suma, la multiplicación y la multiplicación-acumulativa. Debido a la gran cantidad de

diseños resultantes de combinar la operación aritmética junto a su forma de implementación, se hace necesario definir una nomenclatura estructurada para facilitar la comprensión del tipo de prueba que se está comparando. De esta forma, el nombre de los diseños testeados se construirá de la siguiente manera, atendiendo a las características de su implementación:

- Nombre de la operación ("add", "mul" y "multacc") seguida de la anchura de los operandos. Si se han elegido DSPs para su implementación, se le añadirá el sufijo "dsp".
- Adicionalmente, si se aplica TMR a la operación aritmética, se añade el sufijo "_v" junto a un número de versión. Se distinguen cuatro posibles versiones dependiendo del esquema de implementación escogido para el votador:
  1. Se triplica el votador y tanto éste como el operador se implementan en DSPs independientes. En la Figura 46 se observa la estructura propuesta, donde la "O" en un DSP indica que se utiliza para la implementación de la operación lógica, mientras que "=" significa que ese DSP se utiliza como comparador para hacer de votador.
  2. Se triplica el votador y para cada una de las copias del votador, el DSP utilizado se utiliza para implementar una operación lógica. Esto se puede hacer porque el DSP puede realizar una operación aritmética y una comparación a la vez (Figura 47).
  3. Votador único y recursos independientes.
  4. Votador único y recursos compartidos. En este caso el DSP utilizado en alguna de las tres copias de la operación aritmética, también implementa el votador. Obviamente esta versión sea solo aplicable a los diseños donde la operación aritmética es implementada en DSP (sufijo "dsp"), ya que no se puede aplicar esto a un CLB.
- Por último, se añade otro número de versión para distinguir entre el recurso utilizado para la implementación del votador (si con CLBs o DSPs). Esta distinción de subversión se puede aplicar únicamente a las versiones 1. y 3. ya que no es posible compartir un CLB entre votador y la operación aritmética:
  0. Votador implementado con CLBs.

1. Votador implementado con DSPs.

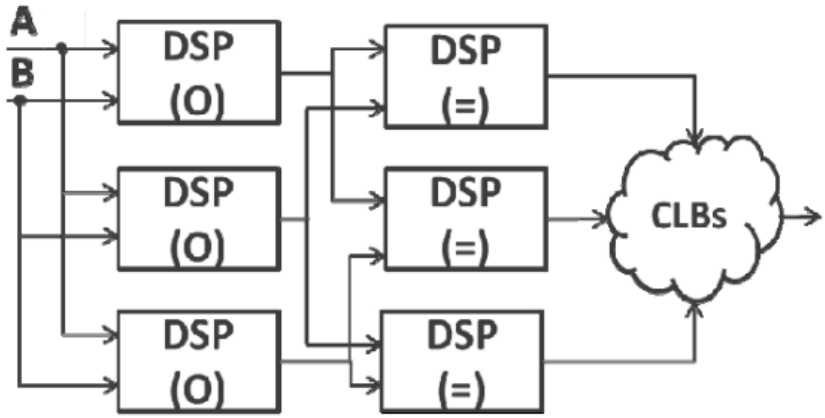

Figura 46: TMR con triple votador implementado en DSPs

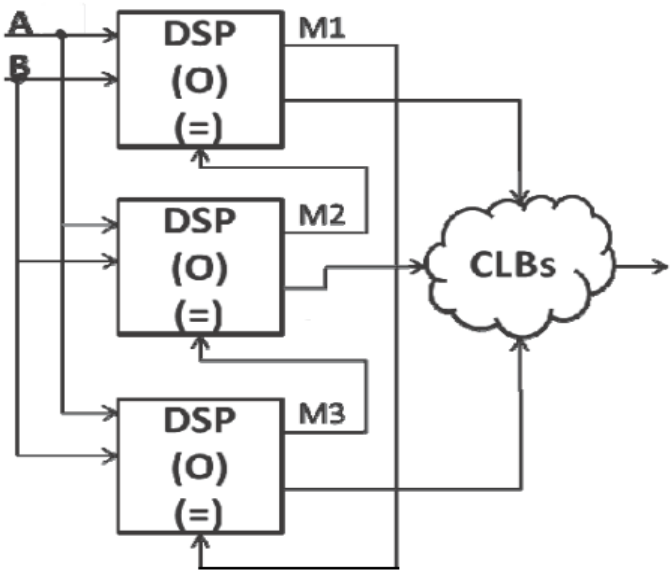

Figura 47: TMR con triple votador reutilizando DSPs

Los resultados obtenidos se basan en un gran número de inyeccion de errores, siendo 1.546.240 la cantidad de SEUs emulados en el circuito más grande implementado. En la Tabla XII se muestran los resultados de un subconjunto de todos los diseños evaluados. La columna 2 y 3 de la tabla indican el número de recursos utilizados, según hayan sido CLBs

o DSPs respectivamente. Las columnas 3-6 muestran el número de fallos absolutos en las salidas de los diseños, siendo la 4 los números de errores producidos en CLBs, la 5 en DSPs y la 6 el total de fallos.

Experimento	DSPs [#]	CLBs [#]	Fallos CLB [#]	Fallos DSP [#]	Fallos [#]
add16	0	48	1660	0	1660
add16_v3.1	1	34	2249	176	2425
add16dsp	1	0	0	1029	1029
add16dsp_v1.1	6	16	960	152	1112
add16dsp_v3.0	3	22	794	237	1031
add16dsp_v3.1	4	16	1015	177	1192
mul16	0	430	12323	0	12323
mul16_v1.0	0	1091	5543	0	5543
mul16_v1.1	3	1058	5567	245	5812
mul16_v3.0	0	1069	4507	0	4507
mul16_v3.1	1	1058	3190	276	3466
mul16dsp	1	0	0	1480	1480
mul16dsp_v1.0	3	80	3072	425	3497
mul16dsp_v1.1	6	32	1191	231	1422
mul16dsp_v2	3	32	1684	362	2046
mul16dsp_v3.0	3	48	1331	287	1618
mul16dsp_v3.1	4	32	1328	340	1668
mul16dsp_v4	3	32	3012	1475	4487
macc16	0	462	10811	0	10811
macc16dsp	1	0	0	1348	1348
macc16dsp_v1.1	6	32	794	553	1347
macc16dsp_v2	3	32	1103	244	1437
macc16dsp_v3.0	3	43	1301	323	1624
macc16dsp_v3.1	4	32	1351	238	1589

Tabla XII: Resultados de inyección de errores para circuitos basados en la técnica FT-DSP

Para una mayor claridad, se analizan los datos por separado agrupando los diseños en función de la operación que implementen. La Figura 48 compara los resultados para el operador add16, donde se puede observar que las versiones que utilizan DSPs, tienen un menor número de errores totales que las versiones con CLBs.

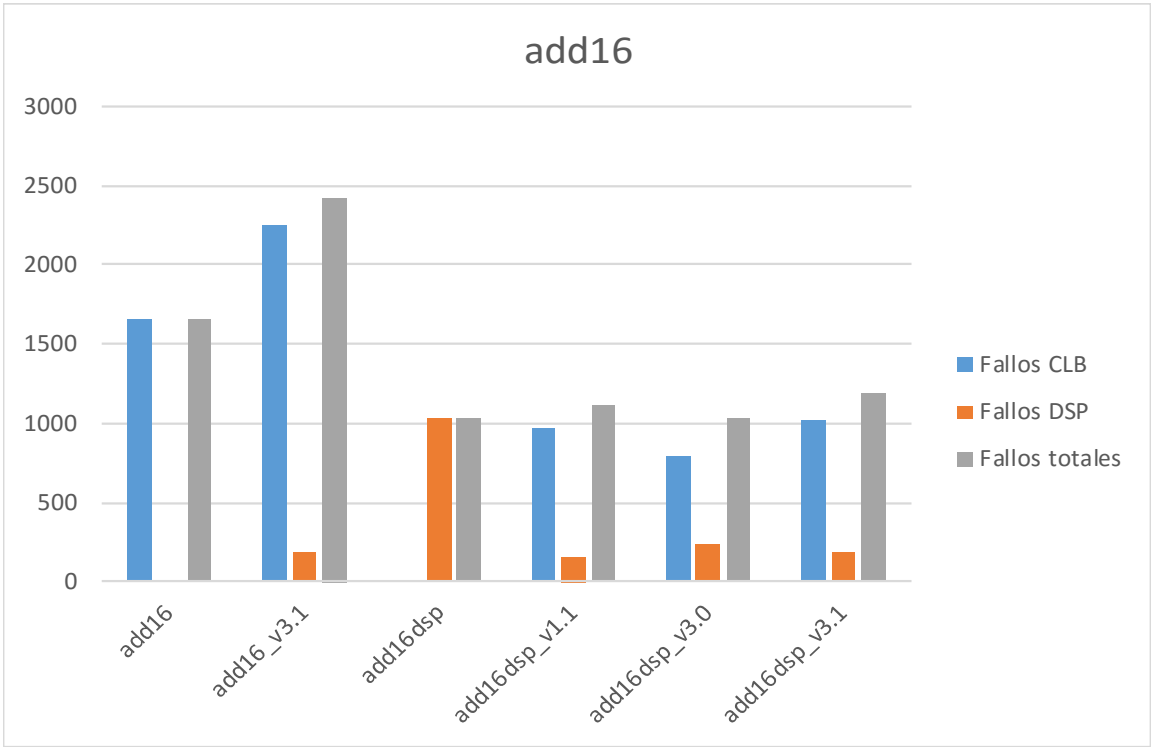

Figura 48: Resultados de inyección de errores para la operación add16 en circuitos basados en la técnica FT-DSP

Hay que destacar también, que para los casos donde se utilizan DSPs el TMR no aporta una mejora notable, siendo muy parecidos los resultados para las versiones con y sin TMR. Sin embargo, hay que tener en cuenta que en todas estas inyecciones de errores no se han tenido en cuenta los SEUs en celdas de memoria del circuito, las cuales se evitan si se aplica TMR. Por lo tanto, hay que valorar si en los diseños basados en DSPs que se quieran proteger, hay una gran cantidad de celdas de memoria que justifique los recursos adicionales utilizados debido a la aplicación del TMR.

Para el caso de la operación mul16, se agrupan sus circuitos de la misma forma que para add16 y se muestran los resultados en la Figura 49:

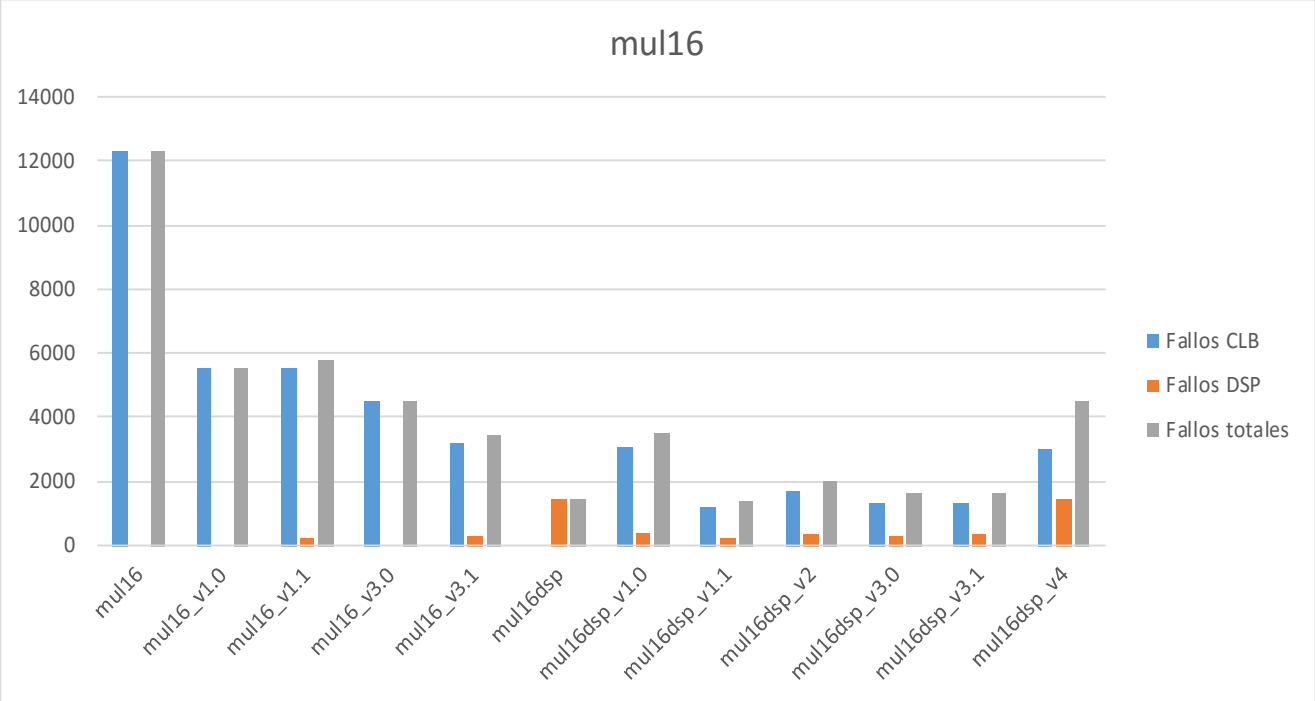

Figura 49: Resultados de inyección de errores para la operación mul16 en circuitos basados en la técnica FT-DSP

En este caso, existe un comportamiento similar al de los circuitos de la operación add16, donde se observa que la utilización de DSPs reduce el número de fallos producidos. Además, otra vez se aprecia que el TMR no mejora la robustez cuando el diseño ya utiliza DSPs, siendo la excepción, las versiones mul16dsp_v1.0 y mul16dsp_v4, donde en estos casos los fallos aumentan frente a un diseño que, a priori, es menos robusto al no aplicar TMR (mul16dsp). Este incremento de errores en términos absolutos, se puede achacar al incremento de recursos y, por lo tanto, al incremento de bits de memoria de configuración, siendo los bits asociados a la parte del votador los bits vulnerables.

Por último, para el operador macc16, los resultados se muestran en la Figura 50. Se vuelve a observar la tendencia de que al utilizar DSPs, el número total de fallos disminuye significativamente con respecto a cuando se utilizan CLBs. Igualmente, estos resultados reafirman que la aplicación de TMR a las versiones basadas en DSPs no ofrece una mejoría considerable a los bits de la memoria de configuración.

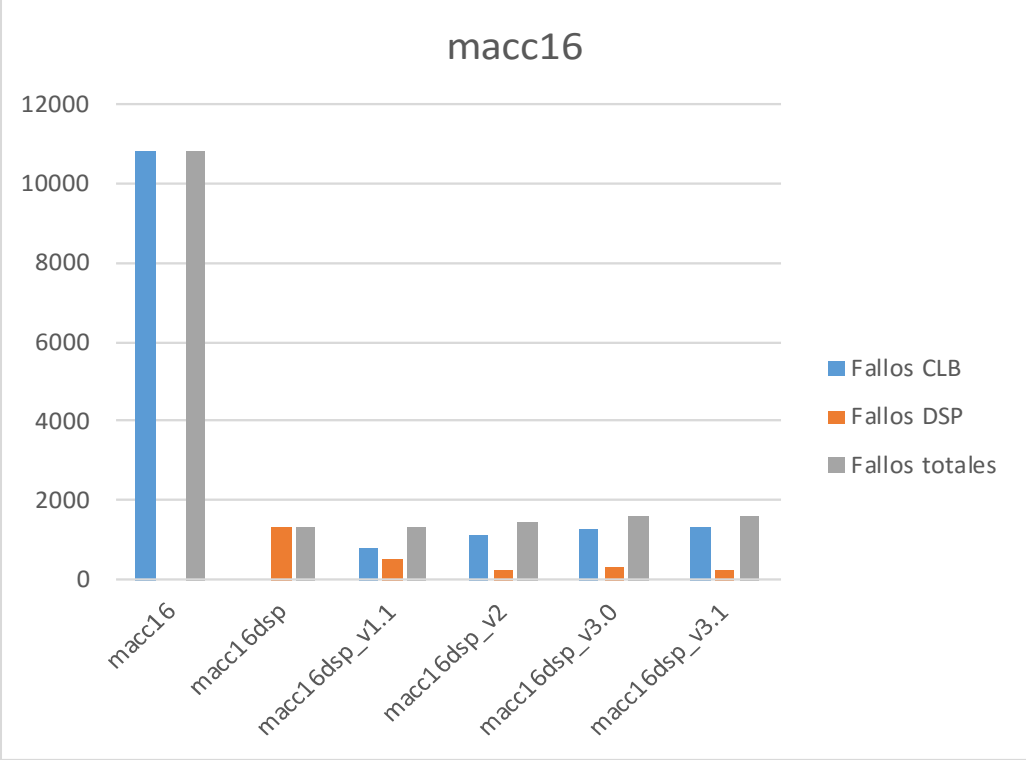

Figura 50: Resultados de inyección de errores para la operación macc16 en circuitos basados en la técnica FT-DSP

Como conclusión, una implementación basada en CLBs siempre es más vulnerable que su homóloga basada en DSPs. Para explicar esto, hay que tener en cuenta que la probabilidad de incidencia de un SEU aumenta con el número de bits configurables, por lo que la cantidad de fallos en términos absolutos dependerá del tamaño del diseño. Así, una operación más simple (add16, 48 CLBs) sufrirá menos fallos que una más compleja (macc16, 462 CLBs). Teniendo en cuenta que los DSPs necesitan menos bits de configuración para una implementación equivalente con CLBs, se puede decir que la utilización de éstos mejora la robustez del diseño.

El TMR, por otro lado, cuando se aplica a diseños basados en DSPs pierde efectividad (para SEUs en la memoria de configuración) con respecto a lo visto en diseños con únicamente CLBs. Sin embargo, no hay que olvidar que los fallos producidos en celdas de memoria del circuito se evitan al aplicar TMR.

Observando el resultado de todas las pruebas realizadas, la solución recomendada es el esquema que implica utilizar TMR en la operación aritmética y en el votador con DSPs

independientes, es decir, la variante “dsp_v1.1”. Esta decisión se debe a que esta versión ofrece, como mínimo, una protección similar a otras versiones basadas en DSPs, existiendo casos donde la protección es ligeramente mejor (mult16) o notablemente mejor (macc16).

La librería CIS de la herramienta utilizada para la implementación de los diseños con FT-DSP, ofrece otras opciones de implementación cuando no existen suficientes DSPs disponibles en la FPGA destino. Una solución factible en este caso, consiste en implementar el diseño con un único votador e implementar la operación aritmética con TMR, utilizando tanto para ésta como para el votador, bloques DSP independientes, es decir, el esquema “dsp_v3.1”. En estos casos se observa un ligero incremento de fallos respecto a la otra versión propuesta (“dsp_v1.1”). Sin embargo, necesita menos recursos al no triplicar el votador por lo que en ciertos casos, según los requisitos de área impuestos, la versión “dsp_v3.1” puede ser más pertinente.

Capítulo 6: Conclusiones y trabajo futuro

Esta tesis ha presentado dos metodologías para la emulación de SEUs en las FPGAs de Xilinx Virtex-5. Atendiendo al tipo de celda de memoria donde se quiera emular el fallo, se distingue entre flipflops y memoria de configuración. Estas metodologías son aplicables a otras FPGAs de Xilinx debido a sus similitudes en la arquitectura del dispositivo. Dicho trabajo ha dado lugar al diseño e implementación de una herramienta de inyección de errores llamada NESSY, que utiliza ambas metodologías presentadas. De esta manera, NESSY es una herramienta de inyección de errores híbrida, ya que se apoya tanto en el método de inyección basado en reconfiguración como en el basado en instrumentación. Esta decisión de diseño se ha visto motivada por el hecho de poder utilizar las ventajas de ambas y alcanzar un buen compromiso entre recursos utilizados y tiempo de inyección en comparación a otras técnicas de la literatura.

Además, se ha presentado un estudio experimental de la robustez de distintas técnicas de protección de circuitos implementados en FPGAs:

1. Los resultados presentados comprueban que la utilización de la técnica TMR mejora la robustez de los diseños frente a SEUs. También, se han presentado distintas versiones con mejoras respecto al TMR simple, como son el TMR con aislamiento y el TMR con aislamiento y votador blindado. Versiones que demuestran ser opciones más robustas que el TMR simple.
2. Se ha desarrollado y validado una técnica aplicable a diseños basados en redes neuronales de tipo Hopfield. Esta técnica mejora la fiabilidad de estos circuitos y requiere la utilización de pocos recursos adicionales en comparación a los necesarios por el TMR.
3. Por último, se ha hecho también un análisis comparativo de la utilización de DSPs frente a CLBs en términos de vulnerabilidad frente a SEUs. En relación a esto, se ha desarrollado y validado una técnica que se basa en la utilización de

dichos DSPs para la protección de circuitos. Esta técnica, tiene en cuenta la escasez de los DSPs disponibles en las FPGAs con respecto a los CLBs y ofrece alternativas de implementación para alcanzar un compromiso entre robustez y recursos.

Como trabajo futuro, debido a que actualmente NESSY solo soporta ficheros fuentes descritos en VHDL, se ha visto la necesidad de dar soporte también a Verilog debido a la tendencia actual de los principales fabricantes de dispositivos reconfigurables en utilizar este lenguaje de descripción hardware. Esto afectará a la herramienta NESSY a la hora de procesar los diseños utilizados, siendo necesario dar soporte a esta sintaxis para poder extraer información de la interfaz del modulo bajo test para poder adaptarlo al resto de la plataforma.

Otro punto importante está relacionado con la obsolescencia de los dispositivos utilizados. La FPGA Virtex 5 es cada vez menos utilizada y los nuevos diseños utilizan familias más modernas, por lo que la migración de la herramienta a FPGAs más nuevas es algo necesario. Se plantea hacer la herramienta compatible con las FPGAs Virtex 7 debido a su similitud en la arquitectura u otras familias más actuales.

Durante el desarrollo de los experimentos, se consideró interesante poder inyectar errores sobre diseños basados en procesadores más una memoria, debido a su obvia importancia en el ámbito del diseño digital. Esto implicaría modificaciones considerables en la forma en que NESSY gestiona la inyección de errores pero daría lugar a un amplio número de diseños nuevos sobre los qué estudiar la vulnerabilidad frente a SEUs.

Otro punto a considerar, es la validación de la herramienta mediante radiación real para tener pruebas experimentales de que nuestros métodos de inyección son correctos y por lo tanto, los resultados obtenidos válidos. Para ello, primero es necesario que exista un modo de inyección “estadístico” que emule de forma realista la aparición de SEUs bajo radiación real y separar el diseño bajo test de la parte estática en FPGAs diferentes.

Por último, quedaría pendiente utilizar dicha herramienta para el estudio de nuevas técnicas de protección de circuitos. Un posible camino es el de estudiar técnicas ad-hoc para arquitecturas concretas de circuitos, ya que suelen aportar mejores resultados que las

técnicas más generalistas. Gracias al conocimiento adquirido en el trascurso de esta tesis, los diseños basados en filtros digitales (utilizan en gran medida DSPs) o basados en otro tipo de redes neuronales, como las redes neuronales convolucionales serían una prioridad.

Capítulo 7: Publicaciones generadas

A raíz del trabajo de esta tesis doctoral, se han generado las siguientes artículos en revistas científicas indexadas en el Journal Citations Report (JCR):

- Serrano, F., Clemente, J. A. and Mecha, H. (2015). "A Methodology to Emulate Single Event Upsets in Flip-Flops Using FPGAs through Partial Reconfiguration and Instrumentation". *IEEE Transactions on Nuclear Science*, *62*(4), 1617-1624.
- Clemente, J. A., Mansour, W., Ayoubi, R., Serrano, F., Mecha, H., Ziade, H., El Falou, W. and Velazco, R. (2016). "Hardware Implementation of a Fault-tolerant Hopfield Neural Network on FPGAs". *Neurocomputing*, *171*, 1606-1609.

También se han generado las siguientes publicaciones en conferencias:

- Serrano, F., Alaminos, V., Clemente, J. A., Mecha, H. and Liu, S. F. (2012). "NESSY: Una Plataforma de Inyección de Errores para una FPGA Virtex-5". *Jornadas de Computación Reconfigurable y Aplicaciones (JCRA)*, pp. 22-27.
- F. Serrano, J. A. Clemente and H. Mecha (2013). "Un estudio de la robustez frente a SEUs de circuitos digitales implementados con los DSPs de las FPGAs". *Jornadas sobre Computación Reconfigurable y Aplicaciones (JCRA)*, pp. 119-125.
- F. Serrano, A. Alonso, S. Alcázar, B. Álvarez-Buylla and J. A. Clemente (2013). "Protección efectiva de circuitos implementados en FPGAs utilizando técnicas de triplicación sobre la plataforma NESSY". *Jornadas sobre Computación Reconfigurable y Aplicaciones (JCRA)*, pp. 113-118.
- Alaminos, V., Serrano, F., Clemente, J. A. and Mecha, H. (2012). "NESSY: An Implementation of a Low-cost Fault-injection Platform on a Virtex-5 FPGA". *13th European Conference on Radiation and its Effects on Components and Systems (RADECS)* (pp. 1-4). IEEE.

Capítulo 7

- Serrano, F., Clemente, J. A. and Mecha, H. (2013). “A study of the robustness against SEUs of digital circuits implemented with FPGA DSPs”. In *2013 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS),* pp. 1-4.
- Sanchez-Élez, M., Pardines, I., Serrano, F. and Mecha, H. (2016). “Radiation-Hardened DSP Configurations for Implementing Arithmetic Functions on FPGA”. *Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 1501-1504.

Apéndice I
Estructura de la memoria de configuración

Los recursos de las FPGAs de la familia Virtex se pueden ver como una matriz bidimensional de un conjunto de elementos configurables. Entre ellos, se distinguen los CLBs, DSPs y BRAMs. Debido a la manera en que la información asociada a estos elementos se organiza y direcciona en la memoria de configuración, es conveniente considerar a un conjunto de recursos del mismo tipo (CLB, DSP o BRAM) como un bloque de recursos.

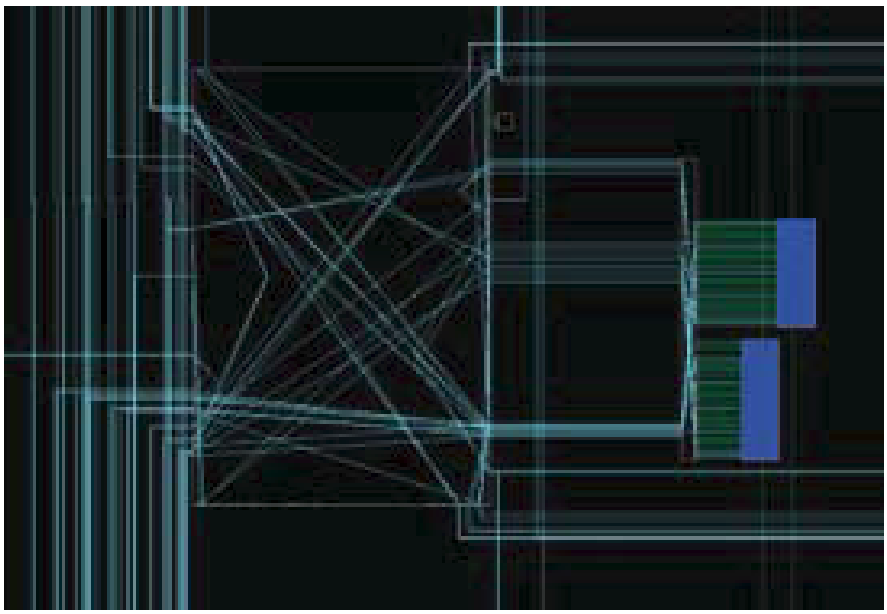

Figura 51: Ejemplo de Bloque-CLB visto desde la herramienta Xilinx FPGA editor

Cada bloque de recursos contiene un conjunto de recursos del mismo tipo (CLBs, DSPs, BRAMs...), más una o varias matrices de interconexionado asociadas a ellos. Existen tres tipos de bloques:

- Bloque-CLB: compuesto por un CLB y dos matrices de conexionado asociadas.
- Bloque-DSP: compuesto por dos DSPs y seis matrices de interconexionado asociadas.
- Bloque-BRAM: compuesto por dos BRAMs y diez matrices de interconexionado asociadas.

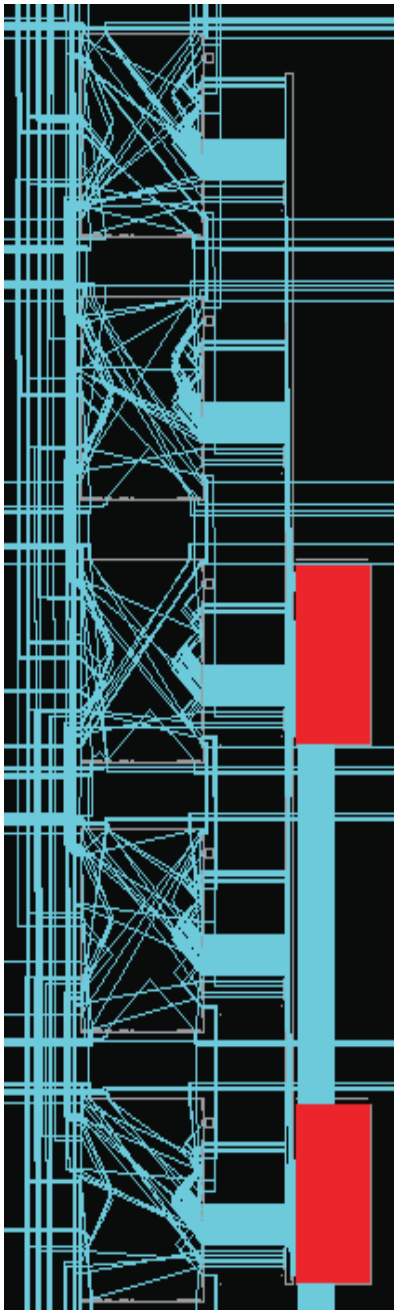

Figura 52: Ejemplo de bloque-DSP visto desde la herramienta Xilinx FPGA editor

La Figura 51 muestra un ejemplo de CLB y una matriz de interconexionado asociada y la Figura 52 hace lo propio para DSPs.

La razón de esta agrupación por bloques se debe a los frames. Un frame es la unidad mínima de configuración direccionable. Para inyectar un error en la memoria de configuración, es necesario modificar como mínimo un frame. Esto implica, que para inyectar un SEU sobre un único bloque-CLB, bloque-DSP o bloque-BRAM, se debe cargar la información correspondiente a un frame completo, el cual contiene información de configuración de todos los recursos del bloque asociado al bit que se quiere modificar.

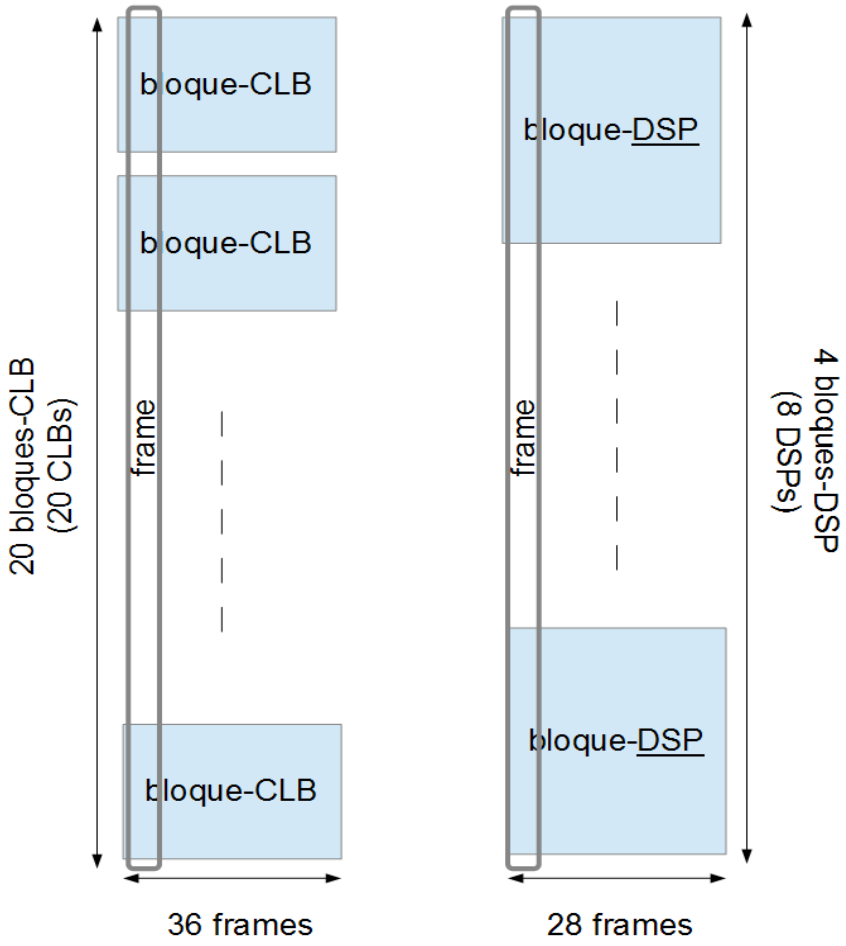

Figura 53: Comparativa entre un bloque-CLB y un bloque-DSP

La información de configuración de un frame está asociada a varios recursos adyacentes, todos del mismo bloque y situados en la misma columna. En el caso de la Virtex-5, la

información de un bloque de recursos adyacentes de una misma columna está contenida en un cierto número de frames de 40 palabras de 32 bits cada uno (en total, 40*32 = 1.280 bits de configuración por frame). El tamaño de dicha columna depende del tipo de recurso. Así, un frame contiene información de un total de veinte bloques-CLB, cuatro bloques-DSPs o bien de dos bloques-BRAM.

La Figura 53 muestra un ejemplo de la equivalencia de frames para una columna de bloques-CLB y bloques-DSP. Haciendo un estudio más en profundidad, se puede comprobar que la matriz de bloques de recursos no es regular, como se puede ver en la figura anterior; es decir, cada bloque de recursos no contiene el mismo número de frames. De hecho, el número de frames que corresponden a cada tipo de recurso es:

- Columna de 4 bloques-DSP = 28 frames.
- Columna de 20 bloques- CLB = 36 frames.
- Columna de 2 bloques-BRAM = 4 frames.

Por tanto, para la Virtex-5 cada conjunto de 4 bloques-DSP se reconfigura completamente a través de la información contenida en un total de 35.840 bits de configuración (28 frames * 1.280 bits/frame); mientras que a cada conjunto de 20 bloques-CLB, le correspondería un total de 46.080 bits (36 frames * 1.280 bits/frame) y para 2 bloques-BRAM serían 5.120 bits (4 frames * 1.280 bits/frame).

Además, de las 40 palabras de configuración comprendidas en un frame asociado a un conjunto de 20 bloques-CLB, cada 2 palabras están asociadas a un mismo bloque-CLB de la columna. Por tanto, hay un total de 36*2 = 72 palabras de configuración asociadas a un CLB y sus 2 matrices de interconexionado adyacentes, lo cual equivale a 2.304 bits de configuración por cada bloque-CLB, repartidos entre 36 frames. También se sabe que hasta la frame 27 para un bloque-CLB, todos los bits de configuración están asociados a la matriz de interconexionado, dejando el resto de frames (del 28 al 36*)* para el CLB.

En el caso de los DSPs, de las 40 palabras de configuración comprendidas en un frame asociada a un conjunto de 4 bloques-DSP, 10 de ellas van asociadas al mismo bloque-DSP. Por tanto, hay un total de 28*10 = 280 palabras de configuración asociadas a 2 DSPs y sus 5 matrices de interconexionado adyacentes (lo cual equivale a 8.960 bits de configuración por

cada bloque-DSP).

Por último, en el caso de los bloques-BRAM, son 20 palabras de cada frame las que configuran una BRAM y sus 5 matrices de interconexionado asociadas, dando un total de 20*4 = 80 palabras de configuración para cada bloque-BRAM (en bits, se necesitan 2.560 bits para configurar la lógica de un bloque-BRAM).

Referencias

1. **A. Kzar, , H. Lim, K. Mutter, S. Syahreza**. Modified Hopfield neural network algorithm (MHNNA) for TSS mapping in Penangstrait. IEEE International Conference on Signal and Image Processing Applications (ICSIPA), pp.187–192: 2013.

2. **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.

3. **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: 2011.

4. **Andrew S. Keys and Michael D. Watson**. Radiation hardened electronics for extreme environments. NASA Marshall Space Flight Center: [date unknown].

5. **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.

6. **Berg Melanie**. Complexity management and design optimization regarding a variety of triple modular redundancy schemes through automation. Marlug: 2010.

7. **C. Bender, P. Sanda, P. Kudva, R. Mata, V. Pokala, R. Haraden and M. Schallhorn**. Soft-error resilience of the IBM POWER6 processor input/output subsystem. IBM J. Res. Develop., Vol. 52, no. 3, pp. 285–292: 2008.

8. **C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia and L. Entrena**. Autonomous fault emulation: A new FPGA-based acceleration system for hardness evaluation. IEEE Trans. Nucl. Sci., vol. 54, no. 1, pp. 252–261: 2007.

9. **Corno F, Reorda M. and Squillero G.** RT-Level ITC'99 Benchmarks and First ATPG Results. IEEE. Design & Test of Computers: 2009.

10. **David G. Mavis, Paul H. Eaton**. Temporally redundant latch for preventing single event disruptions in sequential integrated circuits. Mission Research Corporation: 2000.

11. **D. de Andres, J.-C. Ruiz, D. Gil and P. Gil**. Fault emulation for dependability evaluation of VLSI systems. IEEE Trans. Very Large Scale Integr. (VLSI) Syst, vol. 16, no. 4, pp. 422–431: 2008.

12. **D.E. Rumelhart and J.L. McClelland, C.** PDP Research Group (Eds.), parallel distributed processing explorations in the microstructure of cognition. Foundations, MITPress, Cambridge, MA, USA, Vol 1: 1986.

13. **E. De Lucas**. Methodology to synthesize HDL code to make use of DSP blocks

presented in FPGAs. Trabajo fin de Máster en Ingeniería de Computadores. UCM: 2011.

14. **ESA-ESTEC**. Technique for radiation effects mitigation in ASICs and FPGAs handbook. [date unknown].

15. **F. Faure, P. Peronnard and R. Velazco**. Thesic+: A flexible system for SEE testing. Conference on Radiation and its Effects on Components and Systems (RADECS): 2012.

16. **G.M. Swift, D.J. Padgett and A.H. Johnstson**. A new class of single event hard errors [DRAM cell]. IEEE Transactions on Nuclear Science, Vol 41. no 6, pp. 2043-2048: 1994.

17. **J. Alvarado, V. Kilchytska, G. Berger and D. Flandre**. Efficient single event upset simulations of a tolerant PD SOI CMOS D Flip-Flop. 2009.

18. **J.J. Hopfield**. Neurons with graded response have collective computational properties like those of two-state neurons. Neurocomputing: Foundations of Research, MITPress, Cambridge, MA, USA, pp.577–583.: 1988.

19. **J. Mogollon, H. Guzman-Miranda, J. Napoles, J. Barrientos and M. Aguirre**. FTUNSHADES2: A novel platform for early evaluation of robustness against SEE. Proc. 12th Eur. Conf. Radiation and Its Effects on Components and Systems, pp. 169–174: 2011.

20. **J.M. Rabaey**. Digital integrated circuits. Prentice-Hall, 2nd Ed, NJ, USA: 2003.

21. **Johnson B.W. and Addison Wesley**. Design and analysis of fault tolerant digital systems: 1989

22. **Kakarla S.** Partial evaluation based triple modular redundancy for single event upset mitigation. Graduate School Theses and Dissertations: 2005.

23. **Kakarla S. and Katkoori S.** Partial evaluation based redundancy for SEU mitigation. Combinational Circuits : MAPLD, University of South Florida: 2005.

24. **Kartik Mohanram**. Closed-form simulation and robustness models for SEU-tolerant design. 2005.

25. **Kevin M. Warren, Andrew L. Sternberg, Robert A. Weller, Mark P. Baze, Lloyd W. Massengill, Robert A. Reed,, Marcus H. Mendenhall, and Ronald D. Schrimpf**. Integrating circuit level simulation and monte-carlo radiation transport code for single event upset analysis in SEU hardened circuitry. IEEE Transactions on Nuclear Science, Vol 55: 2008.

26. **K. Lilja, M. Bounasser, T. Assis**. Single event simulation and error rate prediction for space electronics in advanced semiconductor technologies. 2016.

27. **K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin**. SEU-induced persistent error propagation in FPGAs. IEEE Transactions on Nuclear Science,

Vol. 52, no. 6, pp. 2438–2445: 2005.

28. **L. Wissel, E. Cannon, D. Heidel, M. Gordon and K. Rodbell**. Alphaparticle, carbon-ion and proton-induced flip-flop single-event-upsets in 65 nm bulk technology. IEEE Transactions on Nuclear Science, vol. 55, no. 6, pp. 3375–3380: 2008.

29. **M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, A. Marmo, S. Pastore and G.R. Sechi**. A tool for injecting SEU-like faults into the configuration control mechanism of Xilinx Virtex FPGAs. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems: 2003.

30. **M. Ebrahimi, A. Mohammadi, A. Ejlali and S. G. Miremadi**. A fast, flexible and easy-to-develop FPGA-based fault injection technique. Microelectron. Reliab., vol. 54, no. 5, pp. 1000–1008: 2014.

31. **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 (ReCon- Fig), pp. 37-42: 2010.

32. **M. Sanchez-Elez, I. Pardines, F. Serrano, H. Mecha**. Radiation-Hardened DSP configurations for implementing arithmetic functions on FPGA. Design, Automation and Test in Europe (DATE): 2016.

33. **M. Stepanova, F. Lin and V.C. Lin**. A Hopfield neural classifier and its FPGA implementation for identification of symmetrically structured DNA motifs. VLSI Signal Process. Systems, Vol 48, no 3, pp. 239–254: 2007.

34. **Pratt B.** Improving FPGA design robustness with partial TMR. Reliability Physics Symposium Proceedings: IEEE International: 2006.

35. **R.D. Schrimpf and D.M. Fleetwood**. Radiation effects and soft errors in integrated circuits and electronic devices. World Scientific, Singapore: 2002.

36. **R. Koga, S.D. Pinkerton, T. J. Lie and K.B. Crawford**. Single-word multiple-bit upsets in static random access devices. IEEE Transactions on Nuclear Science, Vol. 40, no. 6, pp. 1941-1946: 1993.

37. **R. Leveugle, A. Calvez, P. Maistri and P. Vanhauwaert**. Statistical fault injection: quantified error and confidence. Design, Automation and Test in Europe (DATE): 2009.

38. **R. Leveugle, L. Antoni and B. Fehér**. Dependability analysis: a new application for run-time reconfiguration. Proc. Int. Parallel and Distributed Processing Symp.: 2003.

39. **R. R. Giroux and H. L. Hughes**. Space radiation affects MOSFET's. Electronics vol. 37: 1964.

40. **R. Velazco, G. Foucard and P. Peronnard**. Combining results of accelerated radiation tests and fault injections to predict the error rate of an application implemented in SRAM-based FPGAs. IEEE Transactions on Nuclear Science, vol. 57, no. 6, pp. 3500–3505: 2010.

41. **Shi**, **Zhi-Cheng Ni**, **Nirwan Ansari and Xi Min Zhang Yun Q.** Interleaving for combating bursts of errors. IEEE Circuitos and Systems, pp. 29-42: 2004.

42. **Sterpone, L. and Violante, M.** A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas. IEEE Transactions on Nuclear Science, Vol. 54, no. 4: 2007.

43. **W. Mansour and R. Velazco**. An automated SEU fault-injection method and tool for HDL-based designs. IEEE Transactions on Nuclear Science, vol. 60, no. 4, pp. 2728–2733: 2013.

44. **W. Mansour, R. Ayoubi, H. Ziade, R. Velazco and W. El Falou.** An optimal implementation on FPGA of a Hopfield neural network. Adv. Artificial Neural Systems: 2011.

45. **Xilinx Corporation**. Virtex-5 libraries guide for HDL designs. UG261: 2009.

46. **Xilinx Corporation**. Virtex-5 user guide. UG190 (v 5.4): 2012.

47. **Xilinx Corporation**. Virtex-5 fpga configuration user guide. UG191 (v3.11): 2012.

48. **CAD Group, Politecnico di Torino** [Online]. http://www.cad.polito.it/downloads/tools/itc99.html: 2014.