UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

Técnicas de inyección de fallos basadas en FPGAs para la evaluación de la tolerancia a fallos de tipo SEU en circuitos digitales

Autora:
Marta Portela García

Directora:
Dra. Celia López Ongil

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

Leganés, 2007

TESIS DOCTORAL

TÉCNICAS DE INYECCIÓN DE FALLOS BASADAS EN FPGAS PARA LA EVALUACIÓN DE LA TOLERANCIA A FALLOS DE TIPO SEU EN CIRCUITOS DIGITALES

Autora: Marta Portela García

Directora Dra.Celia López Ongil

Firma del Tribunal Calificador:

Firma

Presidente:	**Dr. Javier Uceda Antolín**
Vocal:	**Dr. Matteo Sonza Reorda**
Vocal:	**Dr. Antonio Torralba Silgado**
Vocal:	**Dr. Jaume Segura Fuster**
Secretario:	**Dr. Luis Entrena Arrontes**

Calificación:

Leganés, de de

Quiero dedicar este trabajo a mi familia por creer siempre en mí

A mis padres

A mis abuelos, a mi hermano y a Marín

Agradecimientos

En primer lugar quiero mostrar mi agradecimiento a los miembros del grupo de investigación dentro del cual se ha realizado esta tesis doctoral, Celia López, Luis Entrena y Mario García. Quiero agradecer a Celia su labor como directora, su constante apoyo y su sincera preocupación tanto en el terreno profesional como en el personal desde el primer día que me incorporé al departamento. Gracias a Luis por estar siempre dispuesto a ofrecer sus consejos y sugerencias. A Mario por su inestimable colaboración y por su dedicación. Gracias a los tres por todo lo que me habeis enseñado, por vuestra ayuda y porque es un verdadero placer trabajar con vosotros.

Grazie a Matteo Sonza Reorda y a su grupo de investigación por su colaboración durante mi estancia en el Politecnico di Torino, por darme la oportunidad de aprender y realizar un fructífero trabajo.

Quisiera mostrar mi agradecimiento a Raoul Velazco y *muito obrigado* a Marcelino Bicho dos Santos por sus valiosos comentarios sobre el trabajo realizado y sobre este documento.

Gracias a mis compañeros del Departamento de Tecnología Electrónica. En especial a los miembros del grupo de Diseño Microelectrónico y Aplicaciones y a aquellos que haceis que el día a día sea tan agradable.

También querría agradecer a mi familia su confianza y su ayuda para que pudiera dedicar mi tiempo y esfuerzo a la investigación; y a mis amigos sus ánimos durante la realización de esta tesis.

RESUMEN DE LA TESIS

Este trabajo de tesis doctoral presenta nuevas técnicas de inyección de fallos transitorios en elementos de memoria, que permiten la evaluación del comportamiento de los complejos circuitos digitales actuales en presencia de fallos SEU (*Single Event Upset*).

Se han propuesto técnicas de inyección que dan solución a la evaluación de la tolerancia a fallos SEU para distintos componentes de los sistemas digitales actuales, en los que se tiende a integrar distintos tipos de circuitos en un mismo chip, SoCs (*System on Chip*). El entorno de inyección en las soluciones propuestas en esta tesis se basa en emulación con dispositivos programables, FPGAs, realizándose las tareas relacionadas con la inyección desde la plataforma *hardware* de emulación. La implementación *hardware* del sistema de inyección minimiza la comunicación necesaria entre el *hardware* y un computador, siendo dicha comunicación la mayor limitación en la velocidad del proceso de inyección. En primer lugar, se presenta una técnica de inyección de fallos basada en la emulación de fallos con FPGA, que permite evaluar un circuito digital cuando se dispone de su descripción en un lenguaje de alto nivel, como VHDL. Por otro lado, se propone una solución para la inyección de fallos en circuitos microprocesadores basada en el uso de la infraestructura de depuración integrada en el propio microprocesador (OCD, *On-Chip Debugger*), para acceder a sus recursos internos (memorias y registros), en un componente comercial o prototipo final del microprocesador.

Cuando se dispone de la descripción del circuito, éste se implementa junto con el sistema de inyección en la FPGA y no es necesario establecer una comunicación con el exterior durante el desarrollo de una campaña de inyección, por lo que esta propuesta se ha denominado Emulación Autónoma. Al implementar el sistema completo de inyección en un único dispositivo (la FPGA) se aumentan la observabilidad y controlabilidad de los elementos del circuito. En este trabajo de investigación se han propuesto optimizaciones del proceso de inyección, basadas en la mayor accesibilidad al circuito que proporciona la Emulación Autónoma, para mejorar la eficiencia de las tareas de inyección de fallos y observación del comportamiento del circuito en presencia de fallos.

En esta tesis se describen y desarrollan tres implementaciones de técnicas de inyección basadas en Emulación Autónoma, denominadas *Time-Multiplexed*, *State-Scan* y *Mask-Scan*. Cada una de las tres implementaciones ofrece un compromiso distinto entre velocidad del proceso de inyección y recursos necesarios para su aplicación. La técnica *Time-Multiplexed* incluye el mayor número de optimizaciones y mejoras por lo que es la técnica que mayor velocidad consigue en el proceso de evaluación pero, para ello, requiere una cantidad de recursos también mayor que las otras dos implementaciones. Las otras dos técnicas son simplificaciones de la primera, por lo que utilizan menos recursos *hardware* en la emulación de fallos.

Además, se han desarrollado modelos de memoria que permiten aplicar la técnica *Time-Multiplexed* a circuitos con memorias empotradas. Los modelos se basan en controlar (para insertar los fallos) y observar (para detectar los errores y sus efectos) el contenido de memoria a través de las señales de control, el bus de datos y el bus de direcciones, evitando recorrer todas las palabras de datos. La inyección de fallos en circuitos con memorias empotradas es un problema de gran interés, puesto que éstas últimas son un componente cada vez más habitual en los diseños actuales. Además no se había propuesto hasta la fecha ninguna solución eficiente para la emulación de fallos en memorias. Esta aportación de la tesis permite inyectar fallos de forma rápida en memorias empotradas resolviendo el problema de su limitada accesibilidad. También para los modelos de memoria, se han propuesto distintas implementaciones en función de las prestaciones conseguidas y recursos *hardware* necesarios, denominados modelo Básico y modelo ECAM. El modelo Básico requiere menos recursos para su implementación, mientras que el modelo ECAM proporciona una mayor capacidad de análisis de los fallos.

Los experimentos realizados, tanto sobre circuitos de prueba como sobre circuitos industriales reales, prueban que la Emulación Autónoma acelera el proceso de inyección con respecto a otras soluciones propuestas, permitiendo inyectar millones de fallos en unos pocos segundos. La aceleración conseguida es de dos órdenes de magnitud, con la técnica *Time-Multiplexed*, con respecto a otras soluciones basadas en emulación, que a su vez proporcionan una aceleración de cuatro órdenes de magnitud con respecto a técnicas basadas en simulación. Esta notable aceleración en la inyección de fallos permite evaluar circuitos de gran tamaño, como los circuitos actuales, donde los posibles fallos suponen un número elevado, y para obtener una medida significativa de su tolerancia a fallos es necesario inyectar un gran conjunto de fallos en un tiempo razonable. Se ha comprobado experimentalmente la viabilidad de la solución presentada para la inyección de fallos en memoria y las características de los modelos de memoria propuestos, para ello se han realizado campañas de inyección sobre un microprocesador industrial en el que se inyectan fallos tanto en los biestables como en la memoria.

Por otro lado, la técnica de inyección que se propone en la tesis orientada a microprocesadores realiza la inyección de fallos y observación de sus efectos en el circuito a través de su OCD. El avance de las capacidades e infraestructuras de depuración en los microprocesadores actuales se debe al auge de SoCs y sistemas empotrados en los que, de otra forma, el acceso para depuración a dicho componente sería inviable o muy costoso. Estas capacidades proporcionan un mecanismo eficaz para acceder a los recursos internos del microprocesador, necesario para realizar la inyección de fallos y observar el comportamiento del circuito. El sistema de inyección propuesto controla el OCD mediante su interfaz JTAG, el más común para acceder a los microprocesadores actuales. Al igual que en el sistema de Emulación Autónoma, todas las tareas de inyección se realizan desde el *hardware*, una FPGA, que se conecta al microprocesador bajo estudio a través de su interfaz JTAG. Esta solución es aplicable a cualquier microprocesador con OCD e interfaz JTAG, lo que son características habituales en la actualidad.

Los experimentos desarrollados sobre microprocesadores comerciales (ARM y PowerPC) demuestran que esta técnica proporciona una solución para la inyección de fallos en componentes microprocesadores comerciales eficiente, de gran generalidad y que alcanza un compromiso entre velocidad y coste.

En resumen, se ha propuesto una solución precisa, rápida y de bajo coste para evaluar la tolerancia a fallos de tipo SEU de los circuitos digitales actuales, permitiendo la inyección de fallos en circuitos de gran tamaño con memorias y microprocesadores empotrados.

ABSTRACT

This PhD thesis presents new transient fault injection techniques to allow evaluating the behaviour of complex digital circuits, as modern circuits, with transient faults in memory elements, i.e., SEU (Single Event Upset) faults.

Fault injection techniques have been proposed to solve SEU tolerance evaluation in different components of systems on chip (SoCs). The fault injection environment of the proposed solutions in this thesis is emulation-based with FPGA, performing injection tasks from the emulation hardware platform. The hardware implementation of the injection system minimises the required communication between hardware and host computer that is a bottleneck in speed injection process. First of all, a transient fault emulation technique in FPGA devices aimed at evaluating a circuit, whose description is available in a hardware description language (as VHDL), is presented. Secondly, a fault injection technique aimed at evaluating fault tolerance in microprocessors is proposed. Such proposal is applied on a final prototype or a commercial component and it consists in using the debugger infrastructure integrated in the circuit (OCD, On-Chip Debugger) to access the microprocessor's internal resources (memory and registers).

On the one side, when the circuit description is available, the circuit is implemented in the FPGA together with the injection system and therefore the communication with the host PC is avoided during fault injection campaign. This fault injection technique has been called Autonomous Emulation. The monolithic hardware implementation for the injection system (a unique FPGA) provides better controllability and observability of the circuit under test, than other solutions. Some injection process optimisations are proposed in this research work in order to enhance the efficiency and the speed of the different injection tasks.

In this work, three implementations of the Autonomous Emulation system are proposed and developed. They are called Time-Multiplexed, State-Scan and Mask-Scan. Each one provides a different trade-off between area overhead and injection process speed-up. Time-Multiplexed technique includes more optimisations than the other techniques. Therefore, it obtains the highest speed-up in the evaluation process, but it requires more area overhead than the other implementations. State-Scan and

Mask-Scan techniques are simplified versions of Time-Multiplexed implementation, using less hardware resources to perform the fault emulation.

Furthermore, memory models have been developed in order to apply the Time-Multiplexed technique to digital circuits with embedded memories. Such models are based on controlling (to insert faults) and observing (to detect the errors and watch their effects) the memory data by means of the control signals, data bus and memory address bus, instead of accessing every memory word, that is a slow task, specially for large memories. The fault injection in embedded memories is a very interesting problem as they are components more and more usual in current digital designs. Besides, there is not an efficient solution for fault emulation in memories in the literature. This thesis' contribution allows the fault injection in embedded memories in a fast way, solving the accessibility limitation problem. Different implementations have been also proposed for the memory models, according to the trade-off between performance and hardware resources requirements; they are named basic model and ECAM model. The basic model involves less hardware resources, whilst the ECAM model provides a better performance in the result analysis task.

The experiments developed in this thesis consist in performing fault injection campaigns in benchmark circuits as well as in real ones. The experimental results prove that Autonomous Emulation speeds-up the injection process with respect to other existing solutions, making possible the injection of millions of faults in a few seconds. The injection process speed increases around two orders of magnitude using Time-Multiplexed with respect to other emulation-based solutions, what are faster than simulation-based techniques in four orders of magnitude. This notable enhancement in the injection speed allows the evaluation of the fault tolerance in large circuits, as the current ones. In modern circuits, all the possible SEU faults suppose a very high number of faults, and in order to obtain a significant measurement of the fault tolerance, injecting a large set of faults in reasonable time is necessary. The feasibility of the proposed memory models has also been analyzed performing fault campaigns in an industrial microprocessor, injecting faults in flip-flops as well as in memory.

On the other side, the fault injection technique, proposed in this PhD thesis, aimed at evaluating microprocessors using the OCD to insert the faults and to observe their effects in the circuit. Nowadays, enhanced debugging capabilities and integrated infrastructures are available in current microprocessors, due to the increasing use of SoCs and embedded systems, where, without an OCD, the debugging process would be infeasible or require a high cost. The OCD provides a mechanism to access microprocessor's internal resources and so it can be used to inject faults and to observe the circuit behaviour. The proposed fault injection system controls the OCD by means of the JTAG interface, what is the most common interface to access modern

microprocessors. As in the Autonomous Emulation System, all the injection tasks are performed in hardware, in an FPGA, that is connected to the microprocessor under test by means of the JTAG interface. This solution could be applicable to any microprocessor circuit with an OCD and a JTAG interface, what are the most common features nowadays.

Developed experiments in commercial microprocessors (ARM and PowerPC) show this technique provides an efficient solution to inject faults in microprocessors devices, applicable to a wide range of different processors and offering a trade-off between the injection process speed and its cost.

In summary, a fast, accurate and low cost solution to evaluate the SEU fault tolerance in modern digital circuits has been proposed. It allows fault injection in large circuits with embedded memories and microprocessors.

ÍNDICE

LISTA DE ACRÓNIMOS

ASIC	***Application Specific Integrated Circuit***
CAD	***Computer Aided Design***
CI	***Circuito Integrado***
ESA	***European Space Agency***
FPGA	***Field Programmable Gate Array***
LUT	***Look-Up Table***
MISR	***Multiple Input Shift Register***
OCD	***On-Chip Debugger***
SE	***Soft Error***
SEE	***Single Event Effect***
SEFI	***Single Event Functional Interrupt***
SEL	***Single Event Latch-up***
SER	***Soft Error Rate***
SET	***Single Event Transient***
SEU	***Single Event Upset***
SoC	***System on Chip***
SoPC	***System on Programmable Chip***
SRAM	***Static Random Access Memory***
TMR	***Triple Modular Redundancy***
VHDL	***VHSIC Hardware Description Language***
VHSIC	***Very High Speed Integrated Circuit***
VLSI	***Very Large Scale Integration***

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1. Introducción

Este trabajo de tesis doctoral presenta nuevas técnicas de inyección de fallos transitorios en elementos de memoria, para evaluar la tolerancia a fallos tipo SEU de circuitos digitales. El trabajo realizado está orientado a dar soporte a la creciente complejidad de los circuitos integrados actuales, para lo cual se necesitan técnicas de inyección más rápidas que las disponibles en el estado de la técnica, que permitan la evaluación de un gran número de fallos en un tiempo aceptable. Dado que los diseños digitales actuales tienden a la integración de distintos componentes electrónicos en un mismo circuito (SoC, *System on Chip*), se requieren técnicas capaces de inyectar fallos en diferentes tipos de componentes. Las técnicas de inyección que se presentan en este trabajo de tesis proporcionan soluciones para la evaluación de diversos circuitos digitales, como circuitos de aplicación específica (ASIC), con memorias empotradas o circuitos microprocesadores.

En este primer capítulo se presenta una breve introducción a la problemática tratada en la tesis, planteando el problema a resolver y los objetivos perseguidos. El capítulo está divido en tres apartados. En el primer apartado se justifica la importancia de que los circuitos digitales sean tolerantes a fallos y de disponer de métodos que permitan medir y comprobar el nivel de tolerancia de un circuito de forma rápida y eficaz. En ese mismo apartado, se indican las limitaciones de las soluciones existentes para la evaluación de la tolerancia a fallos de circuitos digitales, y que motivan la necesidad de proponer nuevos métodos y mejoras. En el segundo apartado se establecen los objetivos a cubrir con este trabajo de tesis y finalmente, en el tercer apartado se señala la estructura del resto del documento.

1.1 Motivación

Los circuitos VLSI (*Very Large Scale Integration*) son susceptibles de sufrir fallos debidos a causas naturales, principalmente a la radiación cósmica. Estos fallos se denominan Efectos de Eventos Simples, en inglés ***Single Event Effects*** (**SEEs**). Tradicionalmente, los SEEs han supuesto una preocupación para los diseñadores de aplicaciones que trabajan en entornos hostiles, como las aplicaciones aeroespaciales o militares sometidas al efecto de la radiación cósmica. A mediados de la década de 1970 se observaron los primeros SEEs en satélites en órbita y ya entonces se propusieron soluciones para este problema. Sin embargo, la evolución de la tecnología unida a la mayor complejidad de los circuitos actuales ha extendido el fenómeno de los SEEs a las aplicaciones en entornos tradicionalmente no hostiles (aplicaciones a nivel terrestre) y ha generado nuevas necesidades en la tolerancia a fallos de estos circuitos. Los SEEs en circuitos digitales utilizados en aplicaciones comerciales se comenzaron a detectar a partir de 1980. Por ejemplo, en 1999 algunos de los servidores de Sun Microsystems

Inc. encargados de tareas críticas tuvieron un funcionamiento erróneo debido a fallos transitorios originados por el entorno [EETimes]. Las características de las tecnologías actuales que han provocado un aumento de la sensibilidad de los circuitos a los SEEs se explican más adelante en este apartado.

Los SEEs más frecuentes, puesto que son aquellos que afectan tanto a aplicaciones aeroespaciales como a aplicaciones comerciales (terrestres), son fallos transitorios que no dañan al circuito físicamente, denominados ***soft errors,* SE**. En la Figura 1 se muestra cómo contribuyen a la tasa de este tipo de errores (*Soft Error Rate*, SER) los distintos elementos que componen los diseños típicos [Mitr05]. La contribución exacta depende de cada circuito, pero en general se asume que mayoritariamente (~90%) la tasa de error se debe a los fallos en elementos de memoria, ya sean biestables, *latches* o celdas de memoria. Este trabajo de tesis se centra, por tanto, en los SE que afectan a los elementos de memoria del circuito, provocando la inversión del valor lógico almacenado, denominados ***Single Event Upset***, **SEU**.

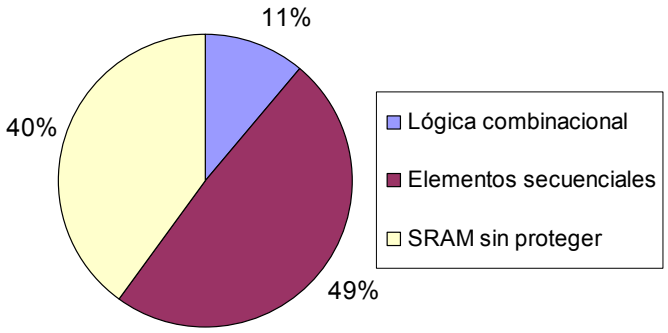


Figura 1. Contribución a la tasa de *soft error* de los distintos elementos que componen un circuito típico [Mitr05].

Como ya se ha dicho, actualmente, los SE (y los SEU) son un problema extendido a cada vez más tipos de aplicaciones, incluso a nivel terrestre debido al uso de las nuevas tecnologías sub-micrónicas y nanométricas. Los avances en la tecnología microelectrónica han permitido bajar el coste de los componentes electrónicos, lo que junto a la mejora de prestaciones y de consumo que ofrecen los CIs han hecho que su uso se extienda a multitud de aplicaciones. En la actualidad, los sistemas digitales se utilizan en un gran número de aplicaciones que requieren un alto nivel de fiabilidad como el control de tráfico ferroviario, aéreo o rodado, automoción, telecomunicaciones, sistemas bancarios, plantas nucleares, medicina, etc.

Por otro lado, otra de las consecuencias del desarrollo de la tecnología es el notable aumento de la sensibilidad de los CIs a los efectos del entorno. Este aumento de la sensibilidad de los CI a los efectos del entorno está considerado como uno de los problemas presentes y futuros a afrontar por los diseñadores y fabricantes en el *International Technology Roadmap for Semiconductors* (ITRS, [ITRS]). El ITRS es un

informe realizado cada dos años por las asociaciones de la industria de semiconductores más importantes del mundo[1] con el objetivo de identificar y prever los problemas críticos a resolver en los CIs en un horizonte de 15 años. En las últimas ediciones de dicho informe [ITRS] se recogen los avances realizados en cada generación de tecnología CMOS, en función de factores como el tamaño característico y capacidad de integración, tensión de alimentación o frecuencia de funcionamiento. Además, se prevé que el desarrollo de la tecnología continúe en el futuro y por lo tanto, la importancia de los SE sea cada vez mayor.

La Figura 2 representa la evolución del tamaño característico en la tecnología CMOS. La disminución del tamaño del transistor implica un aumento de la densidad de integración. En concreto y según los datos publicados en el ITRS, el número de transistores por unidad de superficie se duplica cada dos años. Además, como se observa en la Figura 2 se prevé que los tamaños típicos sigan disminuyendo en los próximos años.

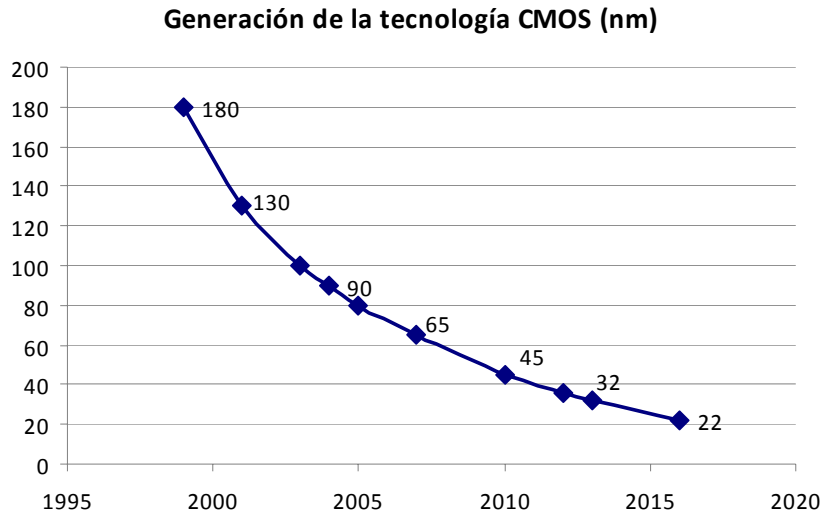


Figura 2. Evolución pasada y futura de la tecnología CMOS de acuerdo con los datos publicados en [ITRS].

La disminución del tamaño del transistor conlleva la reducción de la zona sensible a un SEE, sin embargo al aumentar la capacidad de integración y por lo tanto el número de transistores por dispositivo, la probabilidad de que un evento genere un fallo aumenta, pudiendo incluso verse afectados varios elementos adyacentes a consecuencia del mismo evento (fallos múltiples). En la Figura 3 se representan los datos publicados en [Baum05] relativos a la SER en memorias SRAM. Puede observarse como la tasa de

[1] *European Semiconductor Industry Association* (ESIA), *Japan Electronics Information Technology Industries Association* (JEITA), *Korean Semiconductor Industry Association* (KSIA), *Taiwan Semiconductor Industry Association* (TSIA), *United States Semiconductor Industry Association* (SIA)

SE por bit es prácticamente constante a partir de la tecnología de 180nm, mientras que aumenta considerablemente a nivel de sistema.

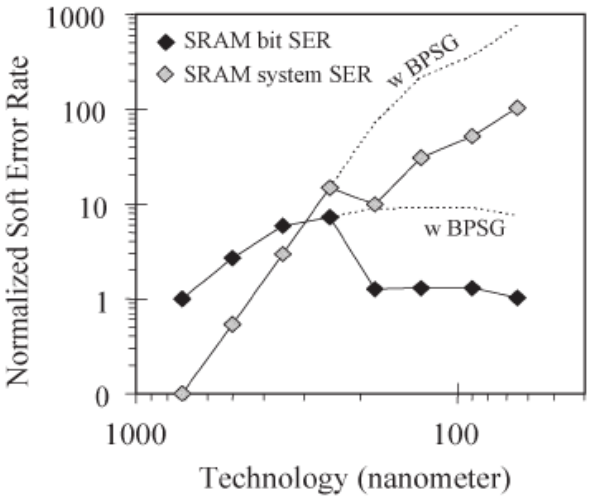


Figura 3. SER para memorias SRAM en función de la tecnología CMOS utilizada para su fabricación [Baum05]

Otras características del escalado de la tecnología son la reducción de la tensión de alimentación y de la capacidad de los nodos del circuito, lo que implica la disminución de la carga crítica (carga mínima necesaria para producir una transición en el valor lógico almacenado). Estos factores tienen como consecuencia la reducción de los niveles aceptables de ruido por lo que son necesarias energías menores para que se produzca un SE, y una partícula de baja energía puede modificar el valor de tensión de una celda. Todos estos factores reducen la resistencia natural a SEUs y aumentan la sensibilidad de los CIs a dichos efectos.

Por otra parte, el incremento de la velocidad de funcionamiento conlleva el uso de puertas lógicas cada vez más rápidas, con retardos comparables a la duración de un SE, y frecuencias de reloj mayores, lo que aumenta la probabilidad de que un fallo en la lógica combinacional se propague hasta un elemento de memoria y sea almacenado.

En [Baum05] se señala que en la actualidad los SEs son el mecanismo de avería dominante en la tecnología CMOS, induciendo una tasa de avería mayor, en hasta tres órdenes de magnitud, que todos los demás mecanismos de fallo combinados (como la electromigración, rotura del óxido de puerta, etc.). Por lo tanto, los SEs suponen hoy en día un problema a resolver y además se prevé que su importancia aumente al incrementar la SER de los sistemas digitales, tanto en los elementos de memoria como en lógica combinacional.

Un fallo en los circuitos utilizados para ejecutar funciones críticas puede dar lugar a grandes pérdidas económicas o incluso originar daños personales, por lo que es fundamental que los circuitos sean tolerantes a fallos y no pierdan por completo su

funcionalidad en presencia de estos efectos. Para que un circuito sea tolerante a fallos, y reducir así la SER, debe resistir los efectos de dichos fallos. Esto se consigue endureciendo (en inglés *hardening*) las zonas más críticas con estructuras robustas capaces de sobreponerse al fallo y de continuar su funcionamiento normal. Una vez que se robustece un circuito, es necesario evaluar la SER o la tolerancia a fallos alcanzada en el CI para comprobar que el nivel de fiabilidad es adecuado para la aplicación dada. Si el análisis indica que las especificaciones relativas a la SER no se alcanzan, deben identificarse los componentes más sensibles para continuar aplicando técnicas que endurezcan el circuito. Por lo tanto, la evaluación de la tolerancia a fallos es una tarea esencial en el proceso de diseño de un CI robusto.

La tolerancia a fallos de un circuito depende fuertemente de la funcionalidad desempeñada. En realidad, en un CI en funcionamiento no todos los bits afectados por un SE producen los mismos efectos. Para no realizar una sobre-estimación de la tasa de error es necesario considerar el uso que se realiza de las distintas partes del circuito, el enmascaramiento lógico que se produce como consecuencia de la carga de trabajo desempeñada, y la velocidad del circuito, que afecta a la propagación de los SE en la lógica combinacional. Por ejemplo, si un SE ocurre en una parte del circuito que no se utiliza durante la ejecución de la aplicación, no conlleva ningún efecto. De esta forma, la tolerancia a fallos de un circuito determinado se verá afectada por dos aspectos fundamentales que son la sensibilidad de la tecnología y el impacto de la funcionalidad. Por ello, es necesario estudiar el comportamiento del circuito ante la presencia de fallos, para verificar la respuesta de las estructuras tolerantes a fallos insertadas en el circuito.

La evaluación de la tolerancia a fallos es un problema ampliamente estudiado por la comunidad científica. Esta evaluación puede hacerse de forma analítica o experimental [Prad96]. Los métodos analíticos se basan en modelos teóricos y parámetros que son difíciles de extraer para circuitos complejos como los actuales. Dentro de los métodos experimentales, la inyección de fallos es una solución ampliamente aceptada y utilizada por la comunidad científica. Este método consiste en forzar artificialmente fallos en el circuito y comparar su comportamiento con los resultados que se obtendrían con un circuito sin fallos. La técnica de inyección de fallos más realista y precisa consiste en someter un prototipo del circuito a estudiar bajo un haz de iones pesados, analizando su respuesta y, por tanto, su tolerancia a fallos frente a SEUs. De esta forma, se provocan fallos reales, puesto que la principal fuente de SEUs es la radiación cósmica (ionizante), sobre un prototipo del circuito. Sin embargo, el coste del equipamiento necesario para implementar esta técnica es muy alto. Además, un posible rediseño supondría un alto coste al requerir la fabricación de un nuevo prototipo. Estos factores, unidos a la dificultad en la localización de las zonas más sensibles a fallos, puesto que la observabilidad está limitada, hacen que la inyección de

fallos mediante la radiación con iones pesados se utilice sólo como una medida final de la tolerancia a fallos, para certificar los componentes. Se precisan, por tanto, métodos de inyección aplicables durante la etapa de diseño del circuito para obtener medidas preliminares de la tolerancia a fallos alcanzada.

Una solución consiste en utilizar una descripción del circuito y mediante simulación, evaluar su comportamiento ante fallos. Este método permite seleccionar y aplicar distintos modelos de fallo, así como elegir las zonas a comprobar y los nodos a analizar. Esta flexibilidad en la medida de la tolerancia a fallos implica un alto coste en términos de tiempo de CPU y de recursos de memoria RAM empleados en la simulación de fallos, especialmente cuando se trata de circuitos muy grandes.

Por otra parte, la emulación *hardware* es una técnica de reciente aplicación en el campo de la evaluación de la tolerancia a fallos que está sustituyendo a la simulación en el caso de diseños complejos. Las técnicas de emulación consiguen acelerar el proceso de inyección con respecto a la simulación, al tiempo que mantienen las ventajas proporcionadas por los métodos basados en simulación, como facilitar la localización de zonas sensibles o el rediseño barato del circuito, si se requiere, al aplicarse en la etapa de diseño.

Sin embargo, los CIs actuales son cada vez más complejos con cientos de millones de transistores. Por lo tanto, el número de posibles fallos que pueden afectar a los actuales CIs aumenta con el tamaño del circuito por lo que para obtener medidas significativas de su tolerancia a fallos es necesario inyectar un gran número de fallos y observar el efecto que provocan. La evaluación de un número significativo de fallos con las técnicas de inyección propuestas por otros autores implica un coste temporal o en recursos considerable, siendo insuficientes para los requisitos de los circuitos complejos actuales. Esto hace necesario avanzar en las técnicas de inyección de fallos, para proporcionar una solución rápida y de bajo coste al proceso de la evaluación de la tolerancia de un circuito frente a SEUs.

Las técnicas de inyección deben adaptarse a las necesidades de los diseños actuales que tienden a la integración de múltiples componentes en un mismo chip, SoCs (*System on Chip*) y SoPCs (*System on Programmable Chip*). Un SoC está constituido por uno o más microprocesadores, memoria empotrada, lógica para funciones específicas e incluso por lógica reprogramable (SoPC), y se caracteriza por maximizar la reutilización de bloques existentes (IP, *Intellectual Property*) mejorando la productividad de la etapa de diseño. En [ITRS] se hace referencia a la importancia cada vez mayor del diseño SoC, por reducir el coste de diseño y el tiempo de desarrollo del sistema hasta que llega al mercado (*time-to-market*). De forma que, se requieren

soluciones generales para la inyección de fallos de manera rápida y eficiente que permitan la evaluación de distintos componentes o circuitos.

1.2 Objetivos

En este trabajo de tesis doctoral se desarrollan y analizan técnicas de inyección de fallos transitorios, de tipo SEU, para evaluar la tolerancia a dichos fallos que presenta un circuito digital. El objetivo global que se persigue consiste en proporcionar una solución que permita la inyección de fallos en circuitos digitales complejos de forma rápida, eficaz y de bajo coste. Este objetivo general puede desglosarse en los siguientes puntos:

- **Velocidad**. La aceleración del proceso de evaluación con respecto a las soluciones existentes es un requisito fundamental en las técnicas de inyección necesarias para evaluar circuitos actuales para permitir la inserción de un gran número de fallos en un tiempo aceptable. Este es el principal objetivo de este trabajo de tesis.
- **Generalidad**. El método de evaluación debe ser aplicable a un amplio rango de circuitos digitales y componentes electrónicos, como microprocesadores, memorias empotradas o circuitos de aplicación específica.
- **Coste**. La solución propuesta debe aplicarse en la etapa de diseño del circuito tolerante a fallos para facilitar su diseño y reducir su tiempo de desarrollo, permitiendo un rediseño barato y rápido del circuito si fuese necesario. De esta forma se reduce el coste de desarrollo del circuito robusto. Además, es muy interesante que la propia técnica de inyección sea una solución de bajo coste.
- **Sencillez en su aplicación**. Debe ser fácilmente integrable en el ciclo de diseño, minimizándose los requisitos necesarios para su uso. Automatizar la generación del sistema de inyección facilita y simplifica su aplicación para distintos circuitos.
- **Eficiencia**. Debe obtenerse una medida fiable y realista de la tolerancia a fallos del circuito bajo estudio.

En definitiva, se pretende proponer técnicas de inyección que cubran las necesidades motivadas por los circuitos y tecnologías actuales, alcanzando un compromiso entre la velocidad del proceso de inyección, que es el principal objetivo, y y el coste.

1.3 Organización del documento de tesis

El documento de esta tesis doctoral está dividido en siete capítulos. Los tres primeros capítulos introducen los conceptos generales relacionados con la tolerancia a

fallos y presentan el estado de la técnica con una revisión de las técnicas de inyección existentes hasta el momento en la literatura. Los capítulos cuarto y quinto describen las aportaciones originales propuestas en esta tesis doctoral, que consisten en técnicas de inyección de fallos transitorios para la evaluación de la tolerancia a fallos en circuitos digitales. El capítulo sexto recoge los resultados experimentales obtenidos y finalmente, en el capítulo séptimo se exponen las conclusiones extraídas y las líneas futuras de trabajo a seguir. En la Figura 4 se representa la organización del documento de tesis. A la izquierda de la figura se muestra el número de capítulo y el título, y a la derecha se indica el concepto principal presentado en dicho capítulo.

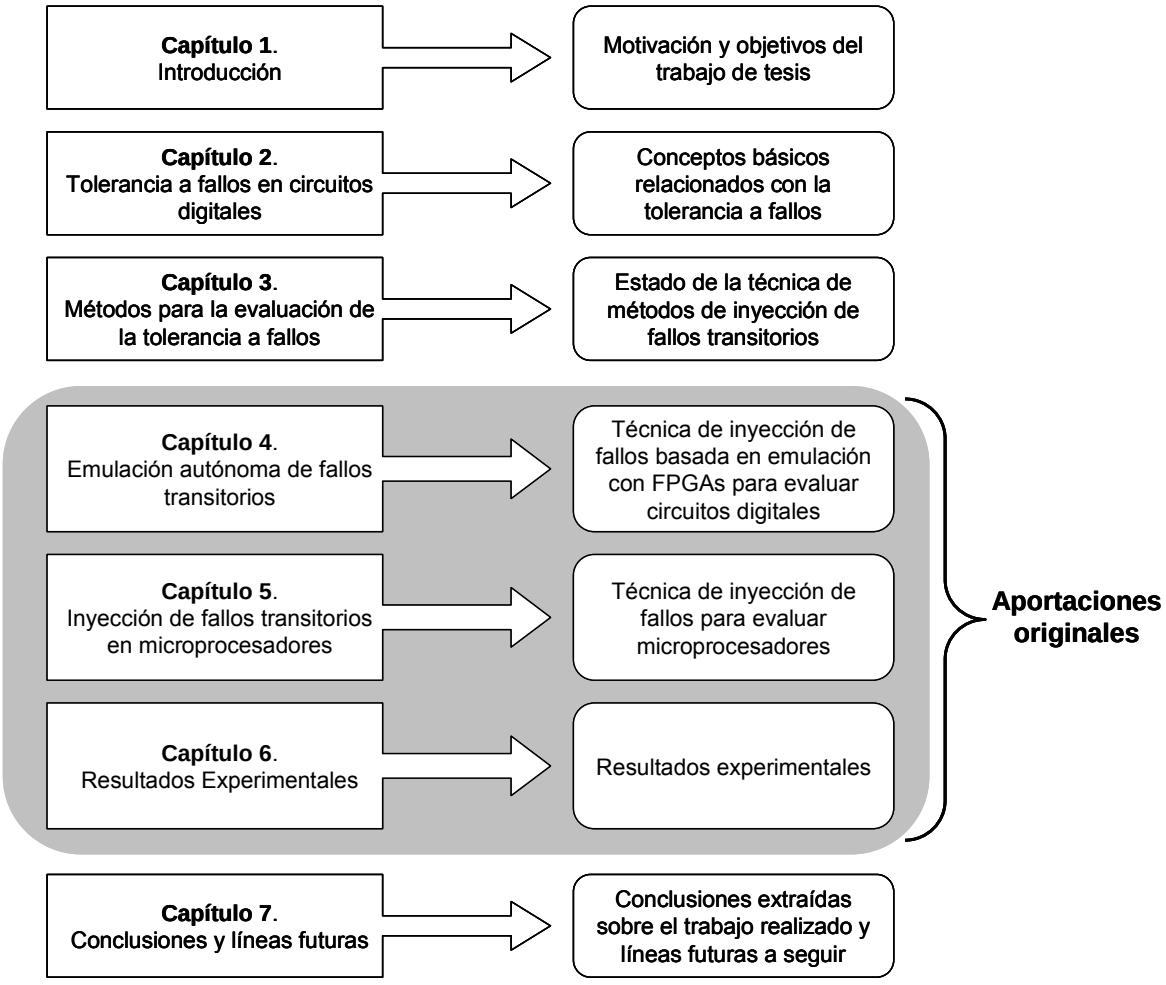


Figura 4. Organización del presente documento de tesis doctoral.

En el capítulo 2 se presentan los conceptos teóricos relacionados con la tolerancia a fallos en circuitos digitales. El principal objetivo de dicho capítulo es enmarcar el problema a estudiar en este trabajo de tesis dentro de la tolerancia a fallos de circuitos digitales, que es una temática muy amplia. Para ello, se define la terminología utilizada en la literatura y se expone una visión global de la problemática relacionada con la tolerancia a fallos, centrando el problema a tratar dentro de las técnicas de inyección de fallos.

El capítulo 3 se centra en los métodos de evaluación de la tolerancia a fallos de circuitos digitales y describe el estado actual de la técnica en métodos de inyección de fallos transitorios, que es el ámbito en el que se centra el presente trabajo de tesis.

En el capítulo 4 se detalla una técnica de inyección de fallos basada en emulación con FPGA, para evaluar la tolerancia a fallos transitorios (SEU, *Single Event Effect*) de circuitos digitales. Dicha técnica se propone en esta tesis como aportación original. La solución propuesta se denomina sistema de Emulación Autónoma y se basa en una arquitectura de emulación en la que todas las tareas de inyección se realizan en *hardware,* en la FPGA, con el objetivo de acelerar notablemente el proceso de inyección con respecto al estado de la técnica.

El capítulo 5 presenta una técnica de inyección de fallos transitorios (SEU) dirigida a evaluar circuitos microprocesadores. Esta técnica de inyección es una aportación original de la tesis y está orientada a la inyección de fallos en implementaciones comerciales de circuitos microprocesadores que disponen de una infraestructura de depuración integrada con una interfaz JTAG, requisito habitual en la mayoría de los microprocesadores actuales. Además, se proponen optimizaciones para el caso particular en el que el circuito microprocesador está integrado en un SoPC (*System on Programmable Chip*).

El capítulo 6 muestra los resultados experimentales obtenidos aplicando las técnicas de inyección propuestas en esta tesis. Los experimentos realizados tienen como objetivo probar la viabilidad de las técnicas de inyección propuestas, la aceleración del proceso de inyección lograda con respecto a otras técnicas de inyección y su eficiencia. Se muestran resultados obtenidos con circuitos de prueba y con circuitos industriales complejos, en función del tiempo, la cobertura de fallos y el coste añadido en área.

Finalmente, el capítulo 7 recoge las conclusiones extraídas del trabajo realizado y las líneas futuras.

CAPÍTULO 2

TOLERANCIA A FALLOS EN CIRCUITOS DIGITALES

2. Tolerancia a Fallos en Circuitos Digitales

En este capítulo se presenta una visión global del problema de la tolerancia a fallos en circuitos digitales con el objetivo de establecer las bases que permitan posteriormente enmarcar este trabajo de tesis. La tolerancia a fallos en circuitos digitales es una temática muy amplia y en este capítulo se presentan los aspectos más relevantes relacionados con ella, incluyendo la descripción de la terminología y los conceptos básicos que serán usados posteriormente en este documento. En este capítulo, se exponen los tipos de fallos que pueden afectar a un circuito, se explican las soluciones típicas utilizadas para hacer que un sistema sea robusto frente a posibles fallos y las métricas existentes para evaluar el nivel de tolerancia a fallos de un circuito.

2.1 Introducción

Un circuito es tolerante a fallos si puede continuar realizando sus funciones en presencia de errores *hardware* o *software* [John89], es decir, la calidad del servicio que presta dicho circuito no disminuye cuando el sistema se ve afectado por algún fallo. Esta propiedad, la calidad del servicio, denominada confiabilidad, o en inglés *dependability,* es el objetivo que se persigue durante el diseño de un sistema tolerante a fallos. La confiabilidad de un sistema se mide estudiando las siguientes características del circuito:

- Fiabilidad (*reliability*): indica la continuidad del servicio prestado. Se mide como la probabilidad de que el circuito opere correctamente a lo largo de un periodo de tiempo (t_0, t) dado que funcionaba en t_0. Cuando un sistema es tolerante a fallos puede continuar con su funcionamiento aunque ocurra un fallo, por lo que tendrá una alta fiabilidad. Sin embargo, una alta fiabilidad no implica necesariamente que el sistema sea tolerante a fallos. Por ejemplo, en un sistema no tolerante, pero con una tasa de fallos muy baja el valor de la fiabilidad sería alto.

- Disponibilidad (*availability*): se mide como la probabilidad de que el circuito opere correctamente y esté disponible para ejecutar sus funciones en un instante de tiempo dado. Esta propiedad, a diferencia de la fiabilidad, depende de la velocidad a la que los fallos pueden ser reparados. Por ejemplo, en un sistema que se recupera rápidamente de un error pero en el que la tasa de fallos es alta, la fiabilidad sería baja mientras que la probabilidad de que en un instante determinado el sistema funcionase correctamente sería alta.

- Seguridad (*safety*): es una medida de la capacidad de un circuito de no dañar a otros circuitos ni a su entorno cuando se produce un mal funcionamiento. Por ejemplo, un fallo en el sistema de ABS de los automóviles no evita que se pueda

seguir frenando. La seguridad se mide como la probabilidad de que un circuito funcione correctamente o de que en caso de que falle lo haga de un modo seguro.

- Capacidad de ejecución (*performability*): es la probabilidad de que el sistema funcione con cierto nivel de calidad en un instante de tiempo determinado, es decir, de que al menos algunas funciones del sistema permanezcan activas. Este atributo se aplica en sistemas en los que un fallo no interrumpe el funcionamiento pero disminuye su calidad. Por ejemplo en un sistema multiprocesador en el que uno de los procesadores falle, el sistema puede continuar su ejecución pero pierde prestaciones como velocidad o memoria disponible.

- Mantenibilidad (*maintainability*): mide la facilidad con la que un circuito se puede reparar una vez que ha fallado. La matenibilidad se cuantifica como la probabilidad de que un sistema con avería pueda estar de nuevo operativo en un instante de tiempo determinado. El proceso de restaurar un sistema incluye la localización del problema, la reparación física y la puesta en funcionamiento. La tarea de localizar el error suele ser el factor limitante. Como se verá más adelante algunas de las técnicas de tolerancia a fallos existentes proporcionan métodos de diagnóstico automático de errores, de forma que la tolerancia a fallos puede mejorar considerablemente la mantenibilidad de un sistema.

- Testabilidad (*testability*): es la capacidad que tiene el sistema para comprobar partes del circuito y la calidad de sus funciones. La testabilidad permite medir la facilidad con la que se puede realizar un test sobre el diseño. Esta característica permite localizar y detectar fallos lo que repercute en la mantenibilidad del sistema mejorándola.

En resumen, los objetivos que se persiguen al diseñar un circuito son, además de que ejecute correctamente su función y sea efectivo en coste, que sea fiable, fácil de testar, fácil de reparar y seguro, es decir, que sea confiable y por lo tanto tolerante a fallos.

El proceso de diseño de un circuito tolerante a fallos se representa en la Figura 5. Este proceso consta fundamentalmente de dos tareas: la aplicación de técnicas de tolerancia a fallos, y la evaluación del nivel de confiabilidad alcanzado. Ambas tareas se realizan de forma iterativa hasta que el circuito a diseñar cumple con las especificaciones de confiabilidad.

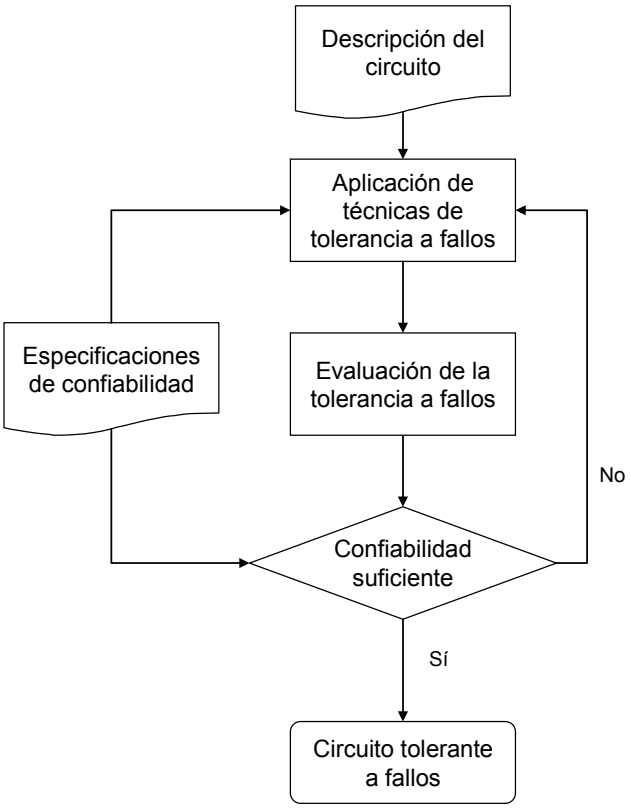


Figura 5. Proceso de diseño de un circuito tolerante a fallos.

El método principal para introducir tolerancia en un circuito consiste en añadir redundancia. Las técnicas de tolerancia a fallos se explican con detalle en el apartado 2.3. Una vez se han aplicado técnicas para alcanzar tolerancia a fallos en un circuito, es necesario evaluar el efecto de dichas técnicas y comprobar que el circuito presenta un nivel de confiabilidad adecuado a las especificaciones impuestas. Las métricas existentes para evaluar la tolerancia a fallos de un circuito se describen en el apartado 2.4. La evaluación de la tolerancia a fallos del circuito persigue dos objetivos de igual importancia:

- Detectar las partes del circuito en las que un fallo daría lugar a un mal funcionamiento inaceptable. Esas partes se conocen en la literatura como zonas sensibles o débiles del circuito y son las zonas en las que es necesario aplicar técnicas de tolerancia a fallos.
- Comprobar si el nivel de tolerancia a fallos del circuito es adecuado. Una vez que se han aplicado técnicas para lograr tolerancia a fallos en un circuito, es necesario evaluar su efecto para comprobar que el nivel de confiabilidad alcanzado cumple con las especificaciones establecidas.

Un aspecto esencial a tener en cuenta en el diseño de circuitos tolerantes a fallos es el tipo de fallo que puede afectar al sistema. Este factor forma parte de las especificaciones a cubrir por el diseño al igual que lo es el nivel de confiabilidad a

alcanzar. De hecho, el tipo de fallo determina la elección del diseñador sobre qué técnicas para lograr tolerancia a fallos son más adecuadas y se ajustan mejor al tipo de fallo especificado.

En el siguiente apartado se presentan los tipos de fallos existentes, cómo se originan y qué efectos tienen en un circuito digital.

2.2 Tipos de fallos

Un **fallo** en un circuito indica un **defecto** físico o imperfección en algún componente *hardware* o *software* del circuito, como por ejemplo lo sería un cortocircuito o un defecto en un dispositivo semiconductor [John89]. La manifestación de dicho fallo se denomina **error**[2]. Así, en el caso del cortocircuito puede ocurrir que una línea del circuito quede fijada a un valor de tensión. A su vez un error puede provocar un funcionamiento deficiente del circuito denominado **avería**. El objetivo del diseño de un sistema tolerante a fallos es minimizar el número de averías que se producen cuando existe un fallo y/o detectar dichas averías.

Existen distintos tipos de fallos que pueden clasificarse en función de sus características. Los atributos a considerar para caracterizar un fallo varían de unos autores a otros. A continuación se describen los atributos más significativos y utilizados en la literatura [John89][Aviz04] (Figura 6):

- **Naturaleza.** Con respecto a su naturaleza, los fallos se pueden clasificar en función de la parte del circuito a la que afectan. Así, se pueden distinguir fallos *hardware* o *software*, y si son fallos *hardware* pueden a su vez clasificarse en fallos digitales o analógicos.
- **Intención.** Los fallos pueden ser accidentales o deliberados. Dentro de los fallos deliberados se pueden distinguir los fallos intencionados con el objetivo de causar daño, fallos *maliciosos*, de los fallos originados sin dicha intención, fallos *no maliciosos*.
- **Causas fenomenológicas.** Este atributo especifica si el fallo se ha originado por la acción del hombre o por causas naturales, sin ninguna participación por parte del ser humano.
- **Fronteras del sistema.** Las causas de un fallo pueden ser externas como interferencias o internas al circuito como el deterioro físico.

[2] Teniendo en cuenta que únicamente son interesantes aquellos fallos que provocan un error, los términos *fallo* y *error* pueden encontrarse en la literatura utilizados indistintamente. En este documento, en general, se hace distinción entre ambos términos, aunque en ocasiones, y debido a la terminología presente en la literatura, se considera que ambos pueden usarse con igual validez.

- **Ciclo de diseño.** Un fallo puede producirse en cualquiera de las etapas del ciclo de diseño. Así se distinguen entre fallos en la etapa de especificación (errores de diseño), fallos en los componentes durante el proceso de fabricación o fallos provocados durante la operación normal del circuito.
- **Duración**. Existen tres tipos de fallos dependiendo de su duración. El fallo es *permanente* si está localizado en el espacio y tiene duración ilimitada. Un fallo se clasifica como *intermitente* si además de tener una duración limitada se repite en el tiempo en la misma posición de forma no periódica. El fallo es *transitorio* si aparece durante un corto periodo de tiempo y luego desaparece.

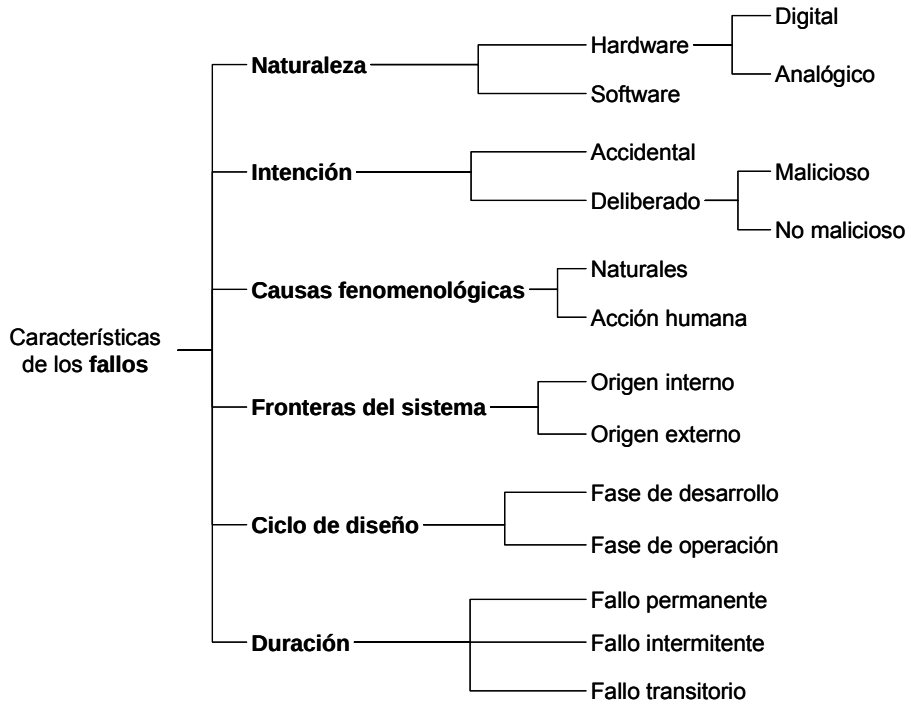


Figura 6. Clases elementales de fallos

Un fallo queda determinado por la combinación de características que presenta. No todas las combinaciones son posibles, por ejemplo un fallo originado por causas naturales no puede presentar una intención deliberada y maliciosa. En la Figura 7 se representan algunas combinaciones de características más comunes como el ciclo de diseño, las causas y la duración del fallo. Los fallos permanentes se deben en su mayor parte a defectos de fabricación y al envejecimiento del circuito, aspectos que llevan asociada una localización espacial. Las causas y mecanismos que originan fallos intermitentes son similares a los de los fallos permanentes. De hecho algunos mecanismos por envejecimiento pueden en ocasiones manifestarse como fallos intermitentes hasta que provocan el fallo permanente, de forma que un elevado porcentaje de los fallos intermitentes aparecen como preludio de fallos permanentes. Por otra parte, los fallos transitorios se deben principalmente a causas ambientales que

no producen un defecto físico, como las interferencias o la radiación cósmica, y se conocen con su denominación en inglés *soft error*.

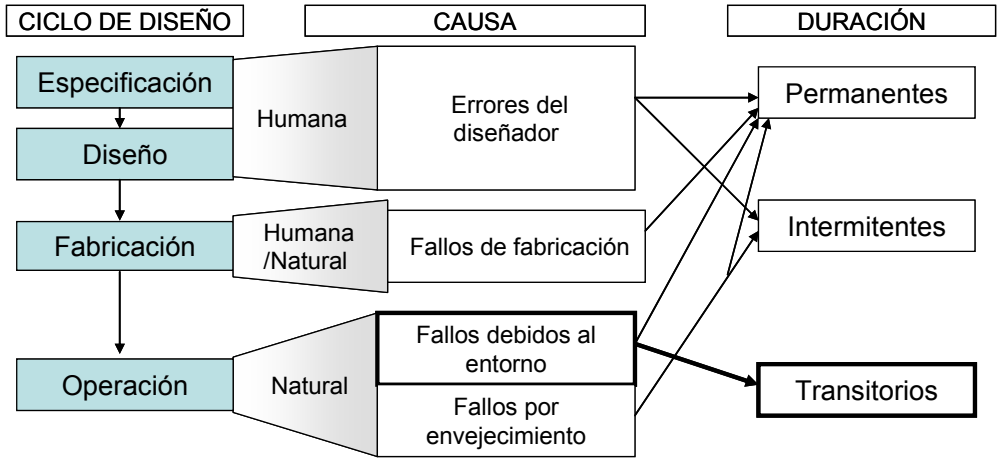


Figura 7. Relaciones entre las diferentes clasificaciones de fallos.

Debido a la complejidad del problema, el estudio de soluciones para cada una de las etapas del ciclo de diseño se ha desarrollado como una temática específica[3]. La tolerancia a fallos se centra en estudiar el problema de los fallos originados durante la fase de operación. Como consecuencia del desarrollo de la tecnología, los circuitos digitales son cada vez más sensibles a las condiciones externas, aumentando el número de fallos transitorios que afectan a un circuito digital durante su fase de operación. Este trabajo de tesis, se centra en el estudio de fallos transitorios, originados por causas fenomenológicas naturales, durante la fase de operación, y que afectan a lo recursos *hardware* internos de circuitos digitales. A continuación se describen los mecanismos por los que se generan dichos fallos transitorios debidos al entorno y qué efectos producen en el comportamiento de los circuitos digitales.

2.2.1 Fallos debidos al entorno. Efectos de la radiación

Los factores ambientales, como por ejemplo variaciones en la temperatura, variaciones en la tensión de alimentación, interferencias electromagnéticas (EMI), y en especial la **radiación cósmica**, pueden afectar al comportamiento normal de un circuito originando un fallo. El fuerte desarrollo de las tecnologías VLSI ha provocado un aumento significativo de la sensibilidad a dichos factores. Se han reducido las dimensiones de los transistores hasta escalas nanométricas (65 nm), las tensiones de alimentación han disminuido (hasta 0,9 V[4]) y las frecuencias de funcionamiento se han incrementado siendo los circuitos cada vez más rápidos (con frecuencias de ~10GHz[4]).

[3] En la etapa de diseño, los errores se evalúan y detectan utilizando validación funcional y una vez se fabrica el circuito se aplica el test de fabricación.

[4] Características de la tecnología utilizada por ejemplo en una de las celdas estándar utilizadas por Fujitsu en la fabricación de ASICs (*Application Specific Integrated Circuit*) (2007)

Estos avances implican una reducción de los niveles de ruido aceptables lo que aumenta la sensibilidad del sistema a las fuentes de ruido y empeora su confiabilidad.

La radiación cósmica consiste en una serie de partículas cargadas (protones, partículas alfa, iones pesados, como por ejemplo núcleos de hierro) y ondas altamente energéticas que provienen del espacio. Es una radiación ionizante, lo que significa que interacciona con la materia a la que llega con la suficiente energía como para provocar la pérdida de electrones en los átomos, ionizando las moléculas. Otro efecto que puede resultar de dicha interacción es el desplazamiento de los átomos fuera de sus posiciones en la estructura cristalina, modificando las propiedades del material. Sin embargo, para las energías incidentes típicas la ionización es el mecanismo de absorción dominante, por lo que es la causa principal de los fallos originados en los sistemas digitales [Schr07].

Debido al carácter ionizante de la radiación cósmica, cuando dicha radiación incide sobre un semiconductor, genera pares electrón-hueco. El campo eléctrico en los circuitos recoge a los portadores creándose un pulso de corriente de magnitud similar a la de operación de los transistores, lo que puede provocar un mal funcionamiento del circuito. Este proceso se representa en la Figura 8 [Baum05].

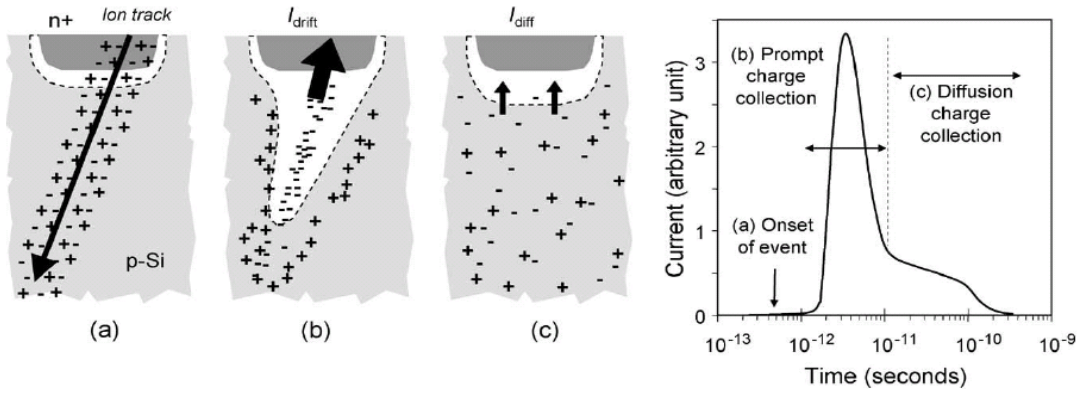


Figura 8. Generación de carga en un circuito integrado como consecuencia de la radiación cósmica y el pulso de corriente provocado [Baum05].

El flujo de radiación cósmica depende de la altitud sobre la superficie terrestre, siendo menor al disminuir ésta. Cuando dicha radiación atraviesa la atmósfera terrestre los rayos cósmicos reaccionan con las partículas de la atmósfera generando cascadas de partículas secundarias. Las partículas secundarias con suficiente energía para continuar atravesando la atmósfera interaccionan a su vez generando nuevas partículas, y así sucesivamente hasta que algunas partículas (menos del 1% del flujo de partículas inicial) alcanzan la superficie terrestre. La Figura 9 muestra la evaluación de la radiación cósmica a través de la atmósfera hasta alcanzar la superficie terrestre [EDN05]. Dentro del flujo de radiación cósmica que alcanza la superficie terrestre los neutrones son uno de los componentes más numerosos. Los neutrones son partículas sin

carga por lo que no generan directamente la ionización de un material, como el silicio en los circuitos integrados pero, debido a su masa, generan iones de alta energía que sí provocan la aparición de pares electrón-hueco dando lugar a un pulso de corriente.

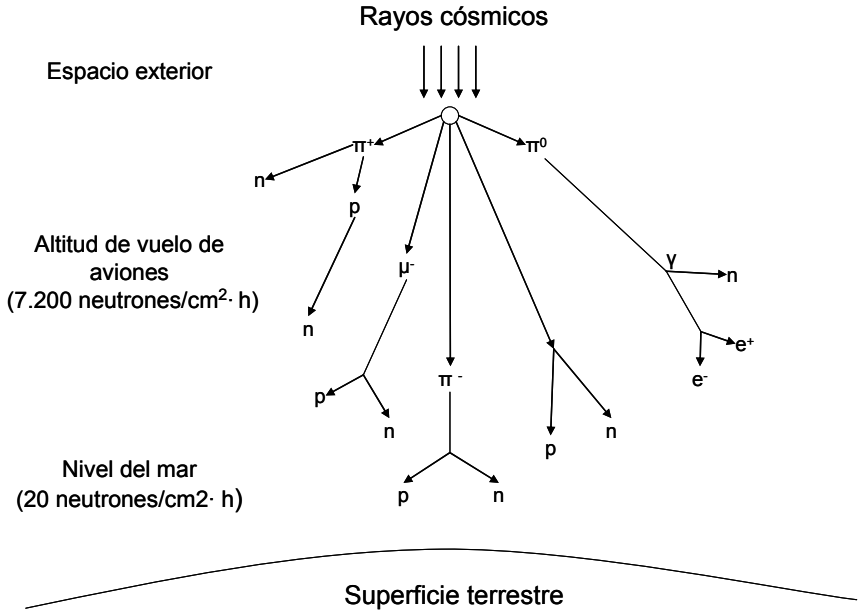


Figura 9. Evolución de la radiación cósmica a su paso por la atmósfera hasta alcanzar la superficie terrestre [EDN05]

Por lo tanto, las aplicaciones aeroespaciales son las más sensibles, al funcionar expuestas a la radiación cósmica sin la protección de la atmósfera, por lo que el estudio de los efectos de la ionización es un problema fundamental en dicho ámbito. Sin embargo, debido a las características de la tecnología actual también la radiación que llega a la superficie terrestre, principalmente neutrones, provoca fallos en los circuitos digitales funcionando a nivel terrestre, puesto que la corriente generada por la interacción es suficiente para generar un fallo. Como consecuencia, los efectos de la radiación se han generalizado, planteando un problema de gran dificultad para las aplicaciones más críticas, tales como las de automoción, medicina, control del tráfico, aeroespaciales, etc. El efecto de la radiación en las tecnologías actuales está considerado como un factor a resolver para continuar con el progreso tecnológico [ITRS].

La ionización produce dos efectos: efectos por dosis total (***total ionizing dose, TID***), debido al carácter acumulativo de la carga generada, y efectos transitorios, denominados ***Single Event Effects***, ***SEE***, generados por la acción de una única partícula. A continuación se detallan estos dos efectos.

2.2.1.1 Dosis total de ionización

Los efectos por dosis total son el resultado de la exposición a la radiación de forma continuada que degrada los componentes y provoca fallos en el funcionamiento del circuito. La radiación genera la aparición de pares electrón-hueco en el óxido de

silicio de los transistores MOS (Figura 10). Los huecos tienen una movilidad más baja que los electrones y mientras que los electrones generados son arrastrados, los huecos quedan atrapados en el óxido apareciendo una carga neta positiva. Esta carga atrapada modifica el valor de la tensión umbral de puerta del transistor o atrae a electrones hacia la superficie aumentando las corrientes de fuga (*leakage current*). A lo largo del tiempo estas cargas van aumentando y se acumulan en la interfaz entre el aislante y el silicio modificando las propiedades del semiconductor. La ionización por dosis total puede provocar que el sistema deje de funcionar.

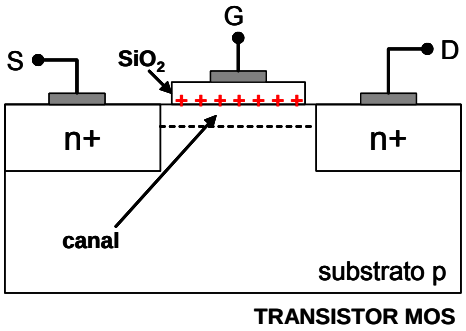


Figura 10. Carga atrapada en el aislante debido a la ionización por dosis total

En las tecnologías actuales, con dimensiones cada vez menores, se ha incrementado la resistencia frente a dosis total ya que el volumen en el que la carga se origina es menor [Poup05][Schr07] y por lo tanto la cantidad de carga generada también es menor. Por el contrario, la sensibilidad a los SEEs está aumentando y en la actualidad se consideran un serio problema.

2.2.1.2 Eventos simples

Cuando una partícula cargada, como las partículas α o los protones, golpea un transistor MOS se generan cargas a lo largo de su camino en forma de electrones y huecos (Figura 11). La fuente y el drenador recogen las cargas generadas dando lugar a un pulso de corriente.

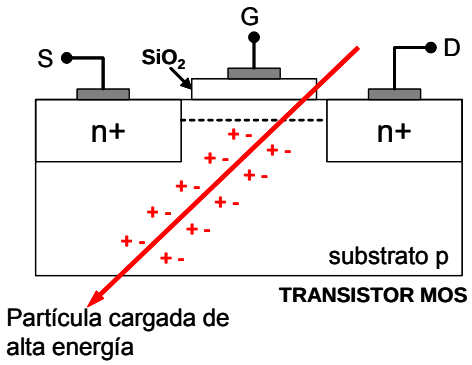


Figura 11. Generación de pares electrón-hueco por la acción de una única partícula

La carga depositada da lugar a distintos efectos. Cuando el fallo producido es un fallo físico se denomina "*hard error*", mientras que los fallos que no dañan permanentemente al circuito se denominan "*soft error*" (SE).

Los efectos más generalizados son los SE y se describen a continuación:

- ***Single-Event Upset, SEU.*** Ocurre cuando una partícula impacta en un transistor perteneciente a un biestable o a una celda de memoria y la carga generada es comparable con la carga crítica (carga necesaria para que el transistor conmute), provocando el cambio del valor lógico almacenado. Es un fallo transitorio que se modela invirtiendo el valor original del bit afectado durante un ciclo de reloj. El elemento de memoria almacena el valor modificado tras el SEU hasta que se procede a escribir un nuevo valor. Este modelo se denomina *bit-flip* y está ampliamente aceptado por la comunidad científica.
- ***Single-Event Transient, SET.*** Un SET se produce cuando una partícula impacta en un transistor perteneciente a lógica combinacional, generando carga que origina un pulso de tensión erróneo de corta duración, en el orden de 100 ps [Poup05]. En tecnologías nanométricas la duración de la transición es comparable al retardo de propagación de la puerta, por lo que el error se puede propagar a través del circuito y llegar a las salidas o almacenarse en un elemento de memoria o varios. De este modo un SET puede dar lugar a múltiples *bit-flip*. Estos efectos transitorios están cobrando más interés con el aumento de las frecuencias de funcionamiento en los circuitos actuales.
- ***Single-Event Functional Interrupt, SEFI.*** Un SEFI se produce cuando un SE provoca la inversión del valor de un bit en un registro crítico de un sistema de control produciendo una interrupción del funcionamiento. Por ejemplo, cuando un SE afecta a la memoria de configuración de una FPGA, a la lógica de reset de un dispositivo, etc. El mecanismo de generación de estos fallos es el mismo que para los SEUs, pero el resultado es distinto porque mientras que un SEU puede no tener un efecto final en la operación del circuito, un SEFI implica por definición un mal funcionamiento que sólo puede corregirse reiniciando la aplicación. En el ejemplo anterior de la FPGA sería necesario reprogramar el dispositivo de nuevo para recuperar su funcionamiento [Baum05].
- ***Multiple-Cell Upset/Multiple-Bit Upset, MCU/MBU.*** Son múltiples fallos transitorios producidos como consecuencia de un único evento. Debido al aumento de la capacidad de integración cada vez es mayor la densidad de transistores en un circuito integrado por lo que está aumentando la probabilidad de que una única partícula genere un fallo transitorio en varios elementos de memoria. Cuando el fallo múltiple se produce en lógica se denomina MBU. Si el

fallo se produce en memoria se denomina MBU si afecta a varios bits de una misma palabra, mientras que se conoce como MCU si las celdas afectadas no pertenecen a la misma palabra de memoria [JEDEC].

- ***Single-Event Latch-up, SEL.*** Un SEL sucede cuando, en tecnología CMOS con substrato, la carga generada por la partícula ionizante activa transistores bipolares parásitos, que se forman entre el sustrato y las diferentes zonas dopadas de los transistores, abriendo un camino entre alimentación y masa y provocando un cortocircuito. Para eliminar este fallo es necesario interrumpir la alimentación del circuito. Este efecto es muy peligroso puesto que las grandes corrientes que se generan pueden dañar los componentes del circuito permanentemente y dejarlo inservible. El SEL puede evitarse utilizando tecnología SOI (*Silicon on Insulator*) puesto que así se eliminan los transistores parásitos o aplicando técnicas de *layout* para aumentar la resistencia de diseños CMOS a este efecto. De hecho, en las aplicaciones espaciales críticas se utilizan generalmente tecnologías resistentes a SEL [Poup05].

De todos los efectos descritos los SEUs son los efectos más notables y más frecuentes debido a las características de las tecnologías actuales y este trabajo de tesis se centra en este tipo de fallos. Sin embargo, con el incremento en las frecuencias de funcionamiento se prevé una tasa creciente de SETs por lo que estos efectos se están convirtiendo en un problema cada vez más importante. Recientemente, se han realizado investigaciones en las que se estudian los efectos de los SETs como si fuesen equivalentes a múltiples *bit-flips*.

Como se ha dicho anteriormente, este trabajo de tesis se centra en el efecto que producen fallos SEUs en circuitos digitales por lo que sólo se consideran fallos transitorios en elementos de memoria (*bit-flip*).

2.3 Técnicas para alcanzar tolerancia a fallos en circuitos digitales

La funcionalidad de los circuitos integrados se ve alterada por los efectos de la radiación. Es importante asegurar la confiabilidad en las aplicaciones de seguridad crítica protegiendo a los sistemas frente a dichos efectos. El proceso a seguir para desarrollar un circuito tolerante a fallos implica localizar las zonas más sensibles e insertar estructuras tolerantes a fallos en ellas, esta última tarea se denomina proceso de endurecimiento[5]. Las técnicas de tolerancia a fallos pueden consistir en soluciones tecnológicas o en técnicas basadas en diseño.

[5] Proveniente del término en inglés *hardening process*, se utiliza en el ámbito de los fallos producidos como efecto de la radiación cósmica, ámbito en el cual se centra este trabajo de tesis.

Las **soluciones tecnológicas** consisten en la modificación del proceso de fabricación con el objetivo de que el circuito fabricado sea menos sensible a los efectos provocados por la radiación disminuyéndose la tasa de fallos e incluso eliminándose por completo algunos de los efectos. Existen tecnologías orientadas a ser robustas frente a radiaciones que se utilizan en las partes críticas de las aplicaciones aeroespaciales. Estas tecnologías, denominadas *radiation-hard* o *rad-hard*, garantizan la tolerancia a fallos por dosis total y una menor sensibilidad a fallos SEL que son los efectos que pueden dar lugar a fallos permanentes o a la destrucción del circuito, y aumentando la resistencia a SEUs [ATMEL]. La tecnología *rad-hard* requiere un proceso de fabricación especial mucho más caro que para otra tecnología por lo que se usa únicamente en aplicaciones muy específicas.

Las **técnicas de tolerancia a fallos por diseño** permiten utilizar las tecnologías CMOS comerciales alcanzando unos niveles de tolerancia a fallos aceptables. Entre estas técnicas se distinguen distintos tipos dependiendo a la clase de efecto frente al que se quiere aumentar la fiabilidad. Para fallos TID se aplican soluciones a nivel físico, modificando el layout del circuito. Una técnica ampliamente utilizada consiste en sustituir el layout habitual para transistores MOS (Figura 12.a) por una estructura en la que una de las difusiones se rodea por completo con la puerta para minimizar los contactos entre el aislante y el substrato que es donde se producen las corrientes de fugas debidas a TID. En la Figura 12.b se ilustra un ejemplo de esta técnica [Vela00]. En cualquier caso hay que señalar que la tolerancia a estos efectos ha mejorado con las nuevas tecnologías debido a la reducción de las dimensiones dónde se genera la carga.

En el caso de fallos SEL, también existen técnicas de diseño que disminuyen sus efectos aunque las tecnologías actuales son menos sensibles. Incrementar la distancia entre los dos transistores parásitos que originan el efecto, o usar un gran número de contactos a alimentación y masa en el circuito son algunas de las posibles medidas a aplicar.

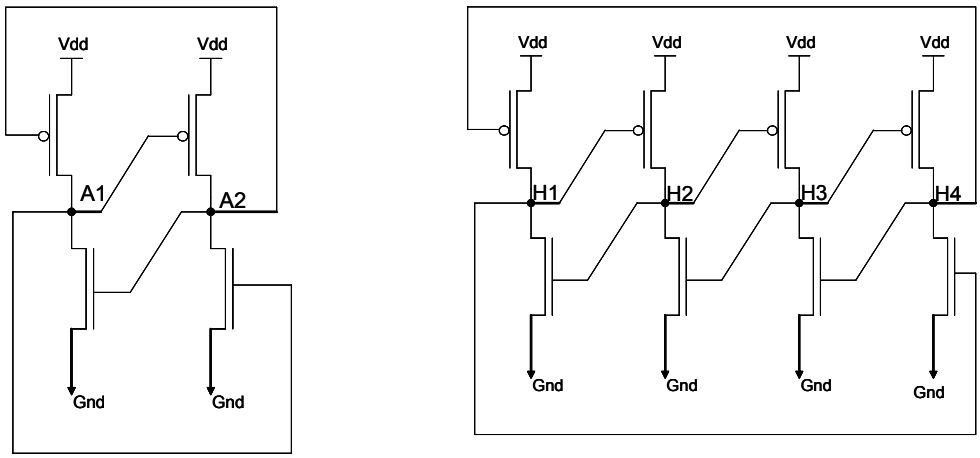
a) b)

Figura 12. Layout de transistor MOS tradicional y modificado para ser tolerante a efectos TID. (a) Layout de un transistor MOS tradicional. (b) *Layout* de un transistor MOS tolerante a corrientes de *leakage*. (*Enclosed Layout Transistor*, ELT)

Sin embargo, la sensibilidad a SETs y SEUs ha aumentado en las últimas décadas y se espera que continúe creciendo. Los fallos SET no eran un problema hasta

hace unos pocos años porque la duración del pulso generado era mucho menor que el retardo de puerta y no tenía ningún efecto. Además, puesto que un SET que se transmite a través del circuito puede modelarse con múltiples *bit-flip*, la mayor parte de las técnicas de endurecimiento existentes están orientadas a endurecer los circuitos frente a SEUs. Las soluciones para fallos SEUs consisten en introducir redundancia en el circuito integrado. Esto puede hacerse a nivel de transistor modificando la celda de memoria, o a nivel más alto, RT (*Register Transfer*) o de sistema, introduciendo estructuras redundantes durante el diseño del circuito. En [Nico05][Cali96][Facc07] se recogen las propuestas más representativas.

Existen varias propuestas de celdas de memoria endurecidas. Una solución consiste en modificar el layout de la celda, aumentando el tamaño del transistor o añadiendo capacidades adicionales para conseguir incrementar la carga necesaria para producir un cambio en el funcionamiento del transistor (carga crítica). Otras técnicas se basan en modificar la arquitectura de la celda. En [Cali96], se presenta una estructura para celdas de memoria SRAM (*Static Random Access Memory*), denominada *Dual Interlocked Cell, DICE*, que es tolerante a cualquier fallo transitorio que afecte a un único nodo de la celda, Figura 13.b.



a) b)

Figura 13. Técnica de endurecimiento de la celda de memoria SRAM. Se modifica la estructura tradicional de la celda (a) por una estructura endurecida (*Dual Interlocked Cell*) (b)

En una celda SRAM tradicional, Figura 13.a, cuando un SEE afecta a un nodo, el valor almacenado cambia. Sin embargo, en una celda *DICE*, dos de los nodos siguen almacenando el valor correcto. Supongamos que la celda de memoria contiene un uno lógico, eso significa que H1 = 0, H2 = 1, H3 = 0 y H4 = 1. Si se produce un fallo transitorio en el nodo H2, se induce una inversión de valor en el nodo H3 pero los nodos H1 y H4 no se ven afectados, de modo que en cuanto el SEE desaparece el valor original se restaura. Aplicando esta solución en tecnologías de 90 nm se obtiene una inmunidad total a SEUs durante el modo estático de operación, mientras que en el modo

dinámico pueden almacenarse estados erróneos si se produce una escritura durante el periodo de recuperación del dato. El mayor inconveniente que presenta este método, o de otros en los que se modifica la estructura de la celda para hacerla tolerante, es el incremento en la cantidad de área necesaria y la disminución de la velocidad. Por ejemplo, para tecnologías de 90 nm, un biestable basado en DICE presenta un 81% de aumento de área con respecto a un biestable industrial [Nico05].

Por otra parte, existe una gran variedad de técnicas que se aplican a nivel de sistema o de transferencia de registros. Consisten en añadir redundancia a la información almacenada, ya sea *hardware*, de información, temporal o *software*. Estas técnicas son fácilmente aplicables por el diseñador en las primeras etapas del ciclo de diseño y con un bajo coste en comparación con otro tipo de soluciones, por lo que son las soluciones más utilizadas. Por estas razones, prestamos especial atención a este tipo de métodos presentando a continuación las diferentes técnicas existentes en detalle.

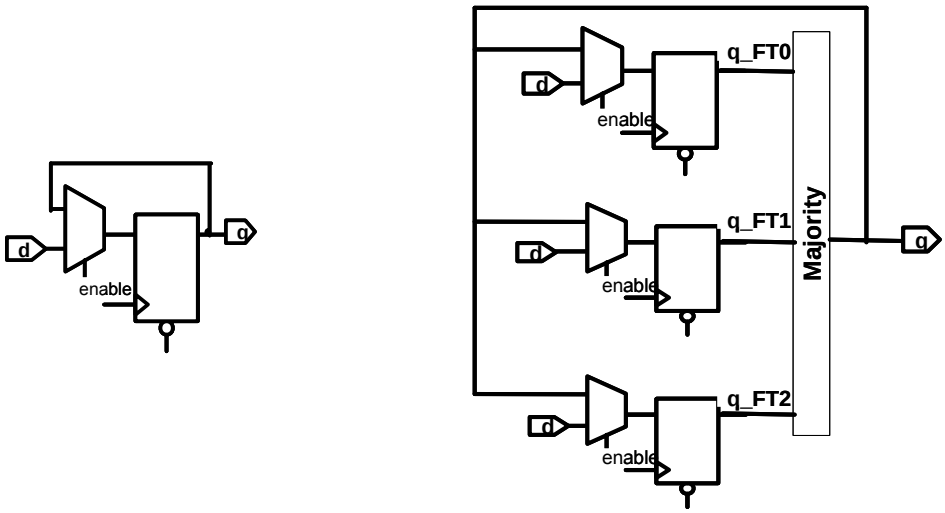
2.3.1 Redundancia hardware

Las técnicas de redundancia *hardware* consisten en replicar bloques lógicos del circuito. Estos componentes realizan la misma tarea de tal forma que si un módulo falla es posible detectarlo o corregir el fallo. Dependiendo de la acción que se tome cuando se produzca un fallo las técnicas se denominan pasivas, activas o híbridas.

Las técnicas de redundancia *hardware* son **pasivas** cuando no requieren ninguna actividad por parte del circuito y se usan cuando el objetivo es enmascarar los fallos y no detectarlos o corregirlos. La gran mayoría de estas técnicas utilizan el mecanismo de *votación por mayoría*. Este mecanismo consiste en replicar el componente *hardware* en cuestión y elegir como salida correcta del circuito aquella que se obtenga mayoritariamente. Así, si uno de los módulos replicados tiene un fallo y la salida del resto de módulos es correcta el fallo se enmascara.

La forma más común de redundancia *hardware* pasiva consiste en replicar N-veces el bloque que se quiere hacer tolerante a fallos, siendo N un número impar, y realizar una votación por mayoría de las salidas para obtener la salida correcta del módulo. Esta técnica se denomina NMR (*N-Modular Redundancy*), por ejemplo una estructura 5MR consiste en replicar 5 veces el mismo módulo. La aplicación de NMR permite la corrección de hasta *N div 2* fallos de tipo SEU. La técnica con N = 3 es la más habitual y se conoce como TMR (*Triple Modular Redundancy*). Como ejemplo ilustrativo, en la Figura 14.a) se representa el módulo a triplicar de circuito original y en la Figura 14.b) se muestra el resultado de aplicar la técnica TMR a dicho módulo. Un SEU en uno de los biestables queda enmascarado por la acción del votador, que genera la salida mayoritaria y por lo tanto libre de error. Esta técnica es válida para errores

simples, en el caso en el que se produjesen dos fallos en el mismo módulo (en dos de los biestables para el ejemplo de la Figura 14 la salida del votador sería errónea.



a) b)

Figura 14. Mecanismo de votación por mayoría, TMR. (a) Módulo original. (b) Módulo triplicado con votación

El problema que presenta esta técnica es que su correcta ejecución depende de la votación puesto que un SET en la señal de salida de dicho módulo generaría un error que se podría almacenar en los tres biestables si se propaga.

Por otro lado, las técnicas **activas** son aquellas que detectan la existencia de fallos y realizan alguna acción para recuperar el sistema eliminando o sustituyendo el *hardware* defectuoso, en vez de enmascarar los errores como en el caso de las técnicas pasivas. Estas técnicas se usan en aplicaciones en las que se permite un funcionamiento erróneo durante un tiempo limitado (mientras se reconfigura el circuito) y requieren un menor grado de redundancia. La mayor parte de estas técnicas constan de detectores de error y módulos de repuesto. Los módulos de repuesto permanecen inactivos hasta que se detecta un error en un componente *hardware* y éste se sustituye por su módulo de repuesto. Como ejemplo, la técnica activa más sencilla es la denominada duplicación con comparación, Figura 15.

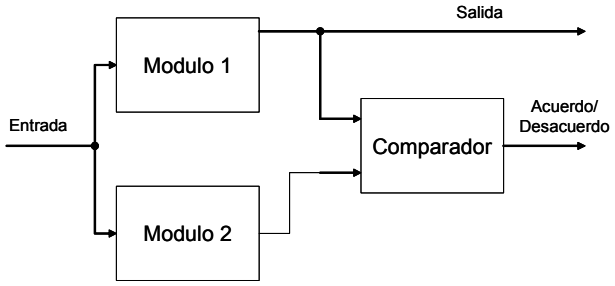


Figura 15. Método de redundancia *hardware* activo: duplicación con comparación.

La técnica de duplicación con comparación consiste en duplicar el bloque *hardware* que queremos hacer tolerante, de tal forma que ambos ejecutan las mismas operaciones en paralelo, comparandose las salidas de ambos módulos. En caso de que no sean iguales se produce un mensaje de error. Esta técnica en particular no permite corregir el error sino sólo detectarlo. Para corregirlo se aplican técnicas de reemplazo por un módulo de repuesto (*standby sparing)*. Consiste en uno o varios módulos de repuesto que sustituyen al módulo original cuando este falla. Los repuestos pueden funcionar paralelamente al módulo original para que el reemplazo sea instantáneo y no implique una interrupción en el funcionamiento normal del sistema, o pueden permanecer inactivos hasta que se detecte el error de forma que los módulos de repuesto no consumen potencia hasta que no son necesarios.

En el caso de procesadores, una técnica habitual es el uso de un *perro guardián* o *watchdog* como método para detectar errores. Esta técnica espera durante un tiempo determinado a que el sistema genere una señal para indicar que la operación continúa con su ejecución normal. En caso de que dicha señal no se active, el *watchdog* genera una señal de error. Por ejemplo, un SEU en el contador de programa de un procesador puede ocasionar una pérdida de secuencia en la ejecución de la aplicación impidiendo que éste alcance cierto punto o finalice. El uso de un *watchdog*, permitiría detectar ese error y tomar medidas para corregirlo, como por ejemplo recomenzar la ejecución.

Por ultimo, los métodos **híbridos** combinan técnicas pasivas y activas y su coste *hardware* es el mayor de los tres métodos. Aúnan el enmascaramiento de fallos, para prevenir que se produzcan salidas erróneas en el circuito, y la detección, localización y recuperación de fallos, para reconfigurar el circuito tras un fallo. Un ejemplo sería aplicar TMR, técnica pasiva, junto con módulos de repuesto, técnica activa. En esta técnica híbrida la salida del votador enmascara el fallo y se utiliza para identificar el módulo con fallo y activar su reemplazo por un módulo de repuesto.

2.3.2 Redundancia de información

En un circuito digital la información se transmite a través de códigos binarios. Un código binario es un modo de representación de datos o información, que sigue ciertas reglas establecidas, características de cada código, mediante los dígitos 0 y 1. Es posible que durante una transmisión una parte de la información sufra un fallo, por ejemplo invirtiéndose el valor de uno de los bits, de forma que el dato que llega al receptor es incorrecto. Si el dato recibido no cumple las reglas del código utilizado, la información es inválida y el receptor detectará el fallo. Sin embargo, si el fallo ha originado un dato incorrecto pero válido (cumple con las reglas del código fijado), es necesario utilizar algún mecanismo que permita al receptor la detección del fallo con el objetivo de evitar que el receptor considere como correcto un dato que es erróneo. Para

solucionar este problema se añade información redundante usando códigos de detección de errores (*Error Detection Codes, EDC)* o de corrección de errores (*Error Correcting Codes, ECC)*. Cada técnica de redundancia de información está caracterizada por su distancia mínima de Hamming, que indica el número de fallos que dicha técnica es capaz de detectar. La distancia de Hamming entre dos datos indica el número de bits en los que ambos datos difieren, de forma que no existen dos datos válidos que difieren en un número de bits menor que la distancia mínima de Hamming. Por ejemplo, si la distancia mínima de Hamming de una codificación determinada es de dos, un fallo en una única posición da lugar a una palabra no válida para esa codificación, detectándose el error. A continuación de describen algunas de las técnicas basadas en redundancia de información más utilizadas [John89]:

- **Códigos de paridad**. Los códigos de paridad son los códigos de detección más simples. El más sencillo consiste en añadir un bit al final de la palabra de información a transmitir, es por tanto un código separable. Este bit indica si el número de 1s que aparecen en la palabra es par o impar. Por ejemplo, si el dato original es el 0010, el dato con código de paridad impar (el dato final contiene un número de 1's impar) es 00100. En este código la distancia mínima de Hamming es de dos por lo que se pueden detectar únicamente fallos simples. Además los códigos de paridad también permiten la detección de un número impar de fallos.

- **Códigos m-de-n (*m-of-n*)**. Un código m-de-n es un EDC que consiste en codificar el dato con *n* bits de los cuales *m* son 1's. Para ello el código más sencillo es el código *i*-de-*2i* dónde, para un dato original de *i* bits (0100, donde *i* es cuatro), se añaden *i* bits de información, de forma que el código resultante contenga un número de 1's igual a *i* (01000111). Este código tiene una distancia mínima de Hamming de 2 por lo que permite detectar cualquier fallo simple. Este código también permite detectar fallos múltiples unidireccionales[6].

- **Sumas de comprobación (*checksum*)**. A diferencia de los códigos anteriores, que detectan fallos en una palabra de información, la suma de comprobación o *checksum* se utiliza para la detección de errores en la transmisión de bloques de datos, es decir, en un conjunto de palabras. El código *checksum* es una palabra de información adicional, que se añade al final de los bloques de datos que se van a transmitir y se genera mediante la suma de dichos datos. La diferencia entre varias implementaciones de esta técnica reside en cómo se genera la suma. La más sencilla es la suma binaria despreciando posibles desbordamientos, denominada suma de comprobación de precisión simple. Otra forma consiste en

[6] Los fallos unidireccionales son aquellos en los que o bien todos los errores consisten en cambios de 0 a 1, o bien todos los fallos suponen un cambio de 1 a 0.

tener en cuenta el desbordamiento usando para ello más bits para representar el *checksum*.

- **Códigos de Berger**. El código Berger consiste en añadir al final de cada palabra de información un número de bits *i* que indica el número de 1's presente en el dato original, representado en complemento a 1. Este código permite detectar todos los fallos múltiples unidireccionales[6], añadiendo el menor número de bits de comprobación posibles. El número de bits que es necesario (*i*) es una función de los bits de información (*n*), $i = \lceil \log_2(n+1) \rceil$. Por ejemplo, sea la información original 0010, con cuatro bits, para codificarlo con el código Berger habrá que añadir tres bits (necesarios para representar hasta el número cuatro). Puesto que hay un único 1 lógico el dato resultante es: 0010110.

- **Códigos de corrección de errores de Hamming**. Estos códigos son una extensión del código de paridad. Dentro de los códigos Hamming, el más sencillo detecta y corrige fallos simples. Consisten en agregar *i* bits de paridad para corregir *n* bits de información, donde $2^i \geq i + n + 1$. Así, la información original se divide en grupos de datos para cada uno de los cuales se genera un bit de paridad. Por ejemplo para los datos (n_3, n_2, n_1, n_0) son necesarios tres bits de paridad (i_3, i_2, i_1). Los bits de información se dividen en grupos y se calcula para cada grupo su bit de paridad: (n_3, n_2, n_1, i_3), (n_3, n_2, n_0, i_2), (n_3, n_1, n_0, i_1). De esta forma si hay un fallo en el bit n_3 los tres bits de paridad indicarán el error, si es el bit n_2 el bit erróneo serán los bits de paridad i_3 e i_2 los que indicarán el error, para n_1 son i_3 e i_1, y para n_0, i_2 e i_1. Por lo tanto se detecta exactamente que bit ha sufrido el fallo y así se puede corregir. En el caso del dato 0100, $i_3 = 0$ (paridad impar), $i_2 = 0$, $i_1 = 1$, y por tanto el código de Hamming sería 0100001[7].

Los códigos de corrección de errores son el mecanismo más habitual para reducir la tasa de errores en memorias [Nico05], en particular en SRAM al ser este tipo de memoria el más sensible a SEUs con las nuevas tecnologías. Sin embargo, hay que tener en cuenta, que el área necesaria es mayor para los códigos correctores que para los que únicamente detectan errores, en especial si corrigen errores múltiples, por lo que el diseñador debe encontrar un compromiso entre la penalización en área y complejidad del sistema, y la tolerancia a fallos que se obtiene.

Como se ha señalado anteriormente, el incremento de la capacidad de integración conlleva un aumento de la tasa de fallos múltiples puesto que una única

[7] Normalmente en los códigos de Hamming los bits de paridad se intercalan con los bits de datos, en vez de añadirlos al final, para facilitar el proceso de corrección del fallo. Sin embargo, por simplicidad, en el ejemplo se muestran al final del dato.

partícula puede afectar a varios transistores. Para combatir los errores múltiples pueden combinarse distintos códigos de detección. Por ejemplo, para fallos dobles se usa el código Hamming junto con un bit de paridad. Cuando se produce un fallo simple tanto el código Hamming como el bit de paridad lo detectan y gracias al primero es posible localizar el bit erróneo dentro de la palabra y corregirlo. Cuando el fallo es doble, el código Hamming detecta un error, pero el bit de paridad es correcto. Por lo tanto con esta combinación de técnicas para fallos simples sería posible detectar los errores dobles y corregir los simples.

Las técnicas de redundancia de información se utilizan en el diseño de circuitos auto-comprobantes (*self-checking*) capaces de detectar sus propios fallos durante el curso de su operación normal. Estos circuitos se usan para implementar módulos de comprobación, como por ejemplo el comparador necesario en la técnica *hardware* de duplicación con comparación o el votador en la técnica TMR. Cuando ocurre un fallo el resultado a la salida del circuito auto-comprobante se corresponde con una palabra no válida para la codificación utilizada, y sólo en el caso en que no haya fallo se obtiene una palabra válida.

2.3.3 Redundancia temporal

La redundancia temporal es una técnica eficaz para detectar o enmascarar fallos transitorios debido al carácter temporal de estos. Además, las técnicas de redundancia temporal requieren generalmente pocos elementos *hardware* extra para su implementación y control a diferencia de las soluciones con redundancia *hardware* y de información. Por lo tanto, esta técnica es adecuada en circuitos en los que el *hardware* es un recurso más crítico que el tiempo, ya sea por el coste, el tamaño o el peso. La redundancia temporal puede implementarse de dos formas distintas:

- Repitiendo los cálculos en varios instantes de tiempo para luego comparar los resultados obtenidos, Figura 16 [John89].
- Almacenando los datos en distintos instantes de tiempo sin necesidad de repetir el cálculo, Figura 17 [Nico99].

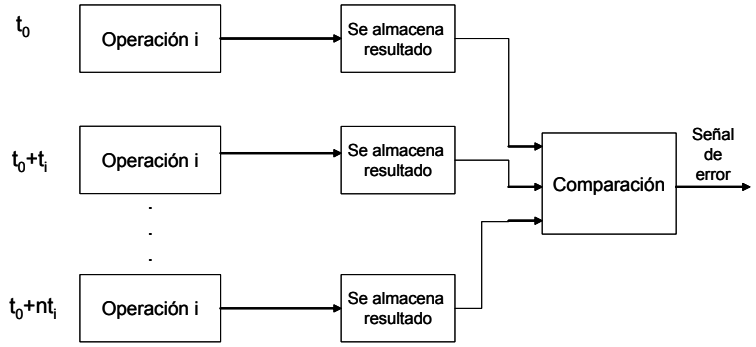


Figura 16. Redundancia temporal basada en recalcular los datos tras un cierto intervalo de tiempo.

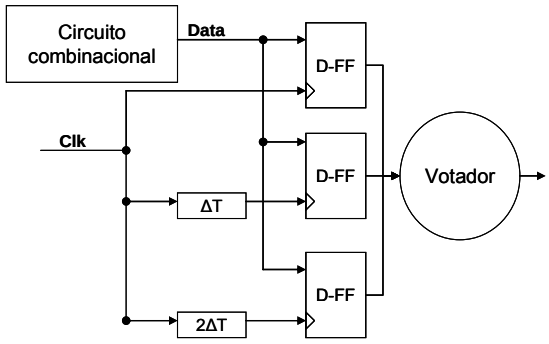


Figura 17. Redundancia temporal almacenando el dato en distintos instantes

Las técnicas basadas en recalcular los datos conllevan una penalización notable en la ejecución, pero pueden ser utilizadas para detectar no solo errores transitorios sino también errores permanentes, añadiendo una cantidad mínima de *hardware*. Estas técnicas se basan en repetir las operaciones tras haber modificado los operandos en algún modo. Para ello se realiza la operación en un instante de tiempo t_0 y se guardan los resultados obtenidos R_1. A continuación, se transforman los operandos con la función F y se vuelve a realizar la misma operación en otro instante de tiempo $t_1 > t_0$, almacenándose los nuevos resultados R_2. Sobre estos últimos se aplica la función inversa F^{-1} obteniéndose R_3. Entonces, si no hay fallos los resultados R_1 y R_3 deben coincidir. Dependiendo de la modificación aplicada se distinguen distintas técnicas. Las propuestas existentes más destacadas son las siguientes:

- Repetición de las operaciones con lógica complementaria. Se aplica típicamente a la transmisión de datos aunque también puede utilizarse en aquellos circuitos en los que $y(x) = \overline{y}(\overline{x})$. Consiste en transmitir en primer lugar el dato original, para a continuación, transmitir el dato complementado.
- Repetición de las operaciones tras realizar un desplazamiento del operando (*REcomputing with Shifted Operands, RESO*).
- Repetición de las operaciones tras realizar un intercambio entre las partes más y menos significativas del operando (*REcomputing with Swapped Operands, RESWO*), Figura 18.

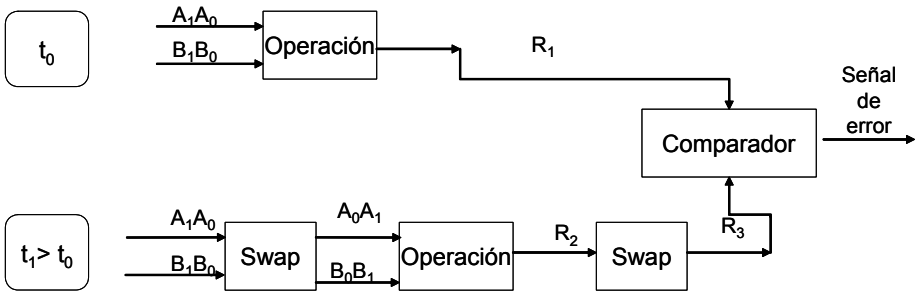


Figura 18. Uso de la redundancia temporal para la detección de fallos permanentes

- Repetición de las operaciones utilizando duplicación con comparación (*REcomputing with Duplication with Comparison, REDWC*). Consiste en combinar redundancia temporal con la redundancia *hardware*. Se divide cada operando en dos partes. Primero se realiza la operación por duplicado (redundancia *hardware*) de la parte más significativa y se comparan los resultados para detectar posibles errores en la parte más significativa del resultado. Después se realiza la operación (redundancia temporal) con la parte menos significativa también por duplicado obteniendo la parte menos significativa del resultado.

Por otra parte, las técnicas de redundancia temporal basadas en almacenar el mismo dato en varios instantes de tiempo, en vez de en recalcular los datos, están orientadas a detectar o enmascarar únicamente fallos transitorios como es el caso de SETs y SEUs. En [Nico99] se presenta una solución cuyo esquema se muestra en la Figura 17. Varios registros almacenan el mismo dato utilizando relojes retrasados un tiempo δ, donde δ es mayor que la duración de cualquier posible pulso transitorio. Sólo uno de los dos biestables cargará un valor erróneo en caso de que se produzca un SET en la parte combinacional del circuito. La salida del votador mostraría el valor correcto en el instante $t0+2\delta+t_{vot}$, donde t0 es el instante de tiempo en el que la está disponible la salida del circuito combinacional y t_{vot} es el retraso que introduce el módulo de votación. Si en lugar de un SET se produce un SEU también se enmascara el fallo. Utilizando dos registros en vez de tres, se detectaría cualquier SEU o SET (fallo simple) pero no podría enmascararse.

2.3.4 Redundancia *software*

En aplicaciones que usan microprocesadores, se puede utilizar la redundancia *software*. Esta técnica proporciona flexibilidad y no requiere ninguna modificación *hardware* por lo que se puede usar para detectar y corregir fallos permanentes y transitorios con un coste muy bajo de implementación. Este mecanismo consiste en replicar parte del código o añadir *software* extra que realice funciones de comprobación, de modo que implica un incremento en los requisitos de memoria. Los fallos pueden generar errores en los datos o en el flujo de control de la ejecución y existen distintas técnicas orientadas a tratar cada uno de estos problemas [Chey00][Oh02].

Las técnicas de comprobación de control de flujo se basan en dividir el programa a ejecutar en bloques de ejecución básicos. Un bloque básico es un conjunto de instrucciones que se ejecutan de forma consecutiva, sin contener ningún salto o llamada a función, excepto quizás en la última instrucción, ni tampoco una instrucción que sea el destino de una llamada o salto, excepto en el caso de la primera instrucción. Cada bloque se identifica de manera única al inicio y cuando finaliza se comprueba que dicho

identificador es correcto. De esta forma, si hubiese un error y se llegase a la mitad de un bloque el identificador no tendría el valor correcto y se detectaría un fallo.

Por otro lado, para obtener tolerancia a los fallos que se producen en los datos, se duplican ciertas variables comprobando la consistencia de ambos valores cuando se produce su lectura. Normalmente, ambos tipos de técnicas se aplican conjuntamente para cubrir tanto fallos en los datos como en el control de la ejecución.

2.3.5 Otras técnicas

Las técnicas presentadas en los apartados anteriores, están dirigidas a endurecer circuitos integrados de aplicación específica (ASIC, *Application Specific Integrated Circuit*) de forma general. Los circuitos actuales tienden a la integración de distintos componentes en un mismo chip, SoC, y a la extensión del uso de circuitos lógicos programables como las FPGAs (*Field Programmable Gate Arrays*) o los SoPCs, proporcionando altas prestaciones con altas densidades de integración. De esta forma, los diseños actuales, cada vez más complejos, presentan nuevos retos en tolerancia a fallos que requieren el desarrollo y propuesta de nuevas técnicas y soluciones para el diseño de circuitos tolerantes a fallos.

Un SoC es un sistema que integra distintos componentes como un procesador empotrado, lógica de usuario o memoria. Frecuentemente, el diseño de estos sistemas se basa en la reutilización de componentes ya existentes, por lo que no es posible modificar notablemente dichos componentes con la inserción de técnicas de redundancia para conseguir que sean tolerantes a fallos. La solución que se plantea en la literatura consiste en aplicar técnicas de endurecimiento de forma global a la arquitectura del sistema. En [Bern06a] se propone una técnica de detección de fallos orientada a endurecer SoCs. La solución propuesta consiste en añadir un bloque adicional encargado de observar las operaciones de otros componentes y combinarlo con técnicas de redundancia *software* para el control de flujo y datos (no se modifica la lógica de cada componente individualmente). Esta técnica puede aplicarse también en el endurecimiento de SoPCs, implementando el módulo de detección en la lógica programable disponible en el sistema [Bern06b].

También los dispositivos lógicos programables, como las FPGAs, requieren de técnicas de tolerancia a fallos específicas. El estudio de los efectos de *SEEs* en FPGAs y el desarrollo de técnicas de mitigación y evaluación de fallos dedicados a dichos dispositivos es un campo de investigación muy amplio y todavía abierto, que queda fuera del alcance de esta tesis. Sin embargo, con el objetivo de proporcionar una visión global de la temática relacionada con la tolerancia a fallos se presenta una breve revisión de los problemas que plantean estos dispositivos con respecto a los fallos debidos a la radiación y las posibles soluciones.

Las FPGAs proporcionan una solución interesante en múltiples aplicaciones, por sus prestaciones y coste. En especial, las FPGAs basadas en SRAM, muy sensibles a SEEs[8], son dispositivos muy potentes para cada vez más aplicaciones, gracias a que pueden ser reprogramadas tantas veces como sea necesario modificando la memoria de configuración (que es una memoria SRAM). Un SEU en la memoria de configuración puede modificar las funciones lógicas implementadas en el dispositivo o las interconexiones (produciendo cortocircuitos, circuitos abiertos o asignaciones erróneas de señales), interrumpiendo el funcionamiento normal, SEFI. Los bits de la memoria de configuración suponen generalmente más de un 95% del número total de bits susceptibles de sufrir un SEU en una FPGA, por lo que, es evidente la necesidad de prestar una especial atención a este tipo de errores.

La tolerancia a fallos en FPGAs se puede lograr de dos formas diferentes:

- Aplicando soluciones relacionadas con la tecnología y la arquitectura interna, es decir, desarrollando FPGAs fabricadas por elementos tolerantes a fallos. Esta solución conlleva la fabricación de un nuevo chip endurecido (*rad-hard FPGAs*) por lo que es muy costosa. Además las FPGAs *rad-hard* ofrecen menos prestaciones que las comerciales. Las tecnologías con las que se fabrican proporcionan menor densidad de integración y no son reconfigurables si eliminan completamente los efectos SEUs de la memoria de configuración (FPGAs de antifusibles). Por lo tanto su uso se limita a aplicaciones de seguridad crítica con alto presupuesto (por ejemplo, en sistemas espaciales).
- Aplicando técnicas de endurecimiento por diseño a alto nivel, introduciendo redundancia en el circuito implementado, en el proceso de rutado, redundancia temporal, etc. Estas técnicas permiten utilizar FPGAs comerciales con altas prestaciones y un bajo coste. Las técnicas que pueden aplicarse para endurecer la lógica de usuario, como los registros, las memorias empotradas, multiplexores, etc., son las mismas técnicas que se pueden aplicar en ASICs, descritas en el apartado 2.3 [Lima06]. Sin embargo, para resolver el problema de los fallos en la memoria de configuración es necesario desarrollar nuevas soluciones, como la reconfiguración periódica del dispositivo o, en inglés, *scrubbing*, tanto completa como parcial, o las técnicas de tolerancia a fallos en el rutado [Ster06].

[8] Las últimas FPGAs disponibles en el mercado hasta el momento (Virtex-5 de Xilinx y Cyclone-III de Altera) están fabricadas con una tecnología CMOS de 65 nm y pueden trabajar a una frecuencia máxima de 440 [Altera] o 550 MHz . Por lo tanto, es evidente, que son dispositivos muy sensibles a los efectos de la radiación, y, debido a la gran cantidad de elementos de memoria, en especial a fallos SEUs.

2.3.6 Aplicación de las técnicas de endurecimiento en circuitos reales

El diseño de sistemas tolerantes a fallos depende de las características del sistema a endurecer. Las técnicas de tolerancia a fallos descritas en los apartados anteriores presentan diferentes ventajas e inconvenientes y solucionan distintos tipos de efectos. En aplicaciones reales, debido a su complejidad, es necesario utilizar varias técnicas tolerantes a fallos conjuntamente, alcanzando un compromiso entre el coste en área y prestaciones, y el nivel de confiabilidad. Como ejemplo ilustrativo, [Poup05] presenta las técnicas de mitigación de SEUs que se han aplicado en las diferentes generaciones de microprocesadores para aplicaciones espaciales desarrollados por la Agencia Espacial Europea (ESA, *European Space Agency*). En concreto, el microprocesador LEON2-FT incluye TMR en los biestables, códigos de detección y corrección de errores en el fichero de registros, código de paridad en las memorias caché y triplicación del árbol de reloj con cierta desviación (*skew*) para evitar SETs.

El éxito del proceso de endurecimiento de un diseño depende de los diseñadores y de su experiencia debido principalmente a dos razones:

- Deben decidir las partes del circuito en las que hay que aplicar redundancia y qué técnicas son las más adecuadas, alcanzando un compromiso entre las penalizaciones introducidas por las distintas técnicas y el nivel de confiabilidad que se logra.
- A excepción de herramientas académicas [Entr01][Varg00], no existen herramientas comerciales que permitan endurecer un diseño de forma automática por lo que la inserción de estructuras tolerantes se realiza de forma manual.

Una vez aplicadas las técnicas de tolerancia a fallos, es fundamental comprobar que el comportamiento de las estructuras y modificaciones insertadas es correcto y que el nivel de confiabilidad del diseño es adecuado. Por lo tanto la evaluación de las técnicas de mitigación aplicadas es una tarea esencial en el proceso de diseño de un circuito tolerante a fallos.

2.4 Medida de la confiabilidad

Un factor fundamental para lograr el éxito en el diseño de sistemas tolerantes a fallos es la validación del diseño con respecto a las especificaciones de confiabilidad. Para ello es necesario disponer de métricas y parámetros específicos que permitan cuantificar la tolerancia a fallos de un sistema. Estos parámetros pueden ser cualitativos o cuantitativos. Las medidas cualitativas describen de forma subjetiva los beneficios de un diseño sobre otro, como la flexibilidad, transparencia, etc. Por otro lado los

parámetros cuantitativos son medidas objetivas representadas con un valor numérico que permiten obtener información acerca de los siguientes aspectos:

- Saber si un sistema alcanza los niveles de confiabilidad necesarios para la aplicación concreta.
- Comparar la fiabilidad de distintos sistemas.
- Evaluar el efecto de utilizar distintas técnicas de endurecimiento o combinaciones de éstas a un mismo circuito.

En este apartado se presentan los distintos identificadores existentes para medir de forma cuantitativa la confiabilidad de un sistema dado.

2.4.1 Tasa de averías

La tasa de averías es el número de funcionamientos erróneos del sistema durante cierto período de tiempo. Normalmente se predice su valor esperado, aunque también puede observarse el circuito en funcionamiento para obtener una medida de este parámetro (medidas en tiempo real; en el capítulo 3 se presentan los distintos métodos de evaluación). En el contexto de la tolerancia a fallos, sólo se tiene en cuenta este parámetro durante la vida útil del componente, despreciando las averías ocurridas durante la primera fase de funcionamiento (mortalidad infantil), debido a causas relacionadas con el proceso de fabricación, y la última fase por desgaste debido al uso. Durante la vida útil se puede considerar que la tasa de averías es constante [John89], véase la Figura 19.

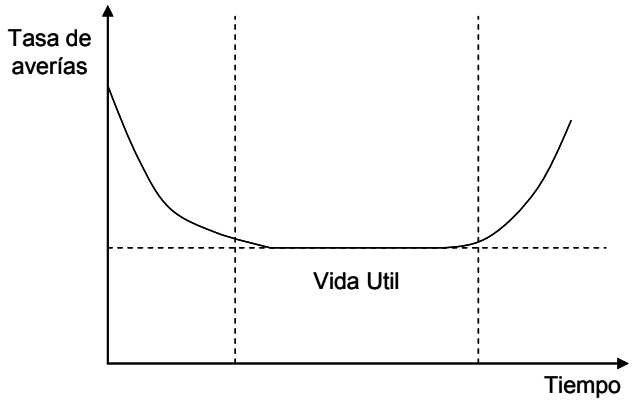


Figura 19. Relación entre la tasa de averías y el tiempo

El valor constante de la tasa de averías correspondiente a la vida útil del dispositivo se expresa como el número de averías ocurrido en 10^9 horas (FIT, *Failure In Time*). Este valor depende de factores como la madurez del proceso de fabricación utilizado, el entorno, la complejidad del circuito, la cantidad de test realizados después de la fabricación o el efecto de la temperatura. Típicamente, en los circuitos actuales sin

mecanismos de tolerancia a fallos el valor de la tasa de averías está comprendido entre 1000 y 50.000 FIT por CI [Baum05][ITRS].

2.4.2 Sección eficaz

La sensibilidad de un dispositivo frente a los efectos de la radiación puede medirse realizando un test de radiación. Este método consiste en aplicar un haz de partículas ionizantes al dispositivo que se quiere caracterizar observando los efectos que se producen. La sección eficaz, σ, es el parámetro que caracteriza la sensibilidad del diseño bajo estudio y mide la probabilidad de que una partícula que interacciona con el dispositivo origine un SEE. Se calcula de acuerdo con la siguiente ecuación:

$$\sigma = \frac{\#\,\text{de eventos ocurridos}}{\text{Fluencia de partículas}} \quad (\text{cm}^2) \qquad (1)$$

donde la fluencia de partículas es el número de partículas por unidad de área del haz de radiación que incide en el dispositivo.

Este parámetro se estudia en función de la energía transferida por el haz de partículas aplicado (LET, *Linear Energy Transfer*). De aquí se obtiene el valor de la energía mínima necesaria para provocar SEEs en el dispositivo. Sin embargo la sección eficaz de un sistema depende notablemente de la actividad que realice. En [Rezg01] se propone una metodología para medir la sensibilidad de un circuito sin necesidad de realizar test de radiación con cada nueva carga de trabajo. Para ello se propone calcular la sección eficaz del circuito para una cierta actividad *A* como sigue:

$$\sigma(A) = \sigma \cdot \tau \qquad (2)$$

donde σ es la sección eficaz cuando el dispositivo está inactivo y τ es la tasa de error del circuito para la carga de trabajo *A* o cobertura de fallos, descrita en el apartado 2.4.4. La tasa de error puede obtenerse, sin necesidad de radiar (aplicando alguna de las técnicas de inyección de fallos que se detallan en el capítulo 3), como la relación entre el número de errores ocurridos y los fallos inyectados en el circuito.

2.4.3 Tiempos medios

Otros parámetros utilizados para medir la calidad de un sistema de forma complementaria a la tasa de averías son los siguientes:

- El tiempo medio para que ocurra una avería (**MTTF**, *Mean Time To Failure*). Se define como el tiempo esperado que el sistema funcionará correctamente antes de que ocurra la primera avería y se puede calcular como $\int_0^\infty R(t)dt$, siendo *R(t)* la fiabilidad del sistema, esto es, la probabilidad de que funcione correctamente

durante un intervalo de tiempo determinado. Esta expresión se puede aplicar siempre que la función de la fiabilidad tienda a cero en el infinito.

- El tiempo medio entre averías (**MTBF**, *Mean Time Between Failure*). Es el tiempo medio transcurrido entre dos funcionamientos erróneos del sistema. Su valor suele ser del orden del MTTF y se puede estimar como la relación entre un cierto intervalo de tiempo y el número de fallos ocurridos en ese período.
- El tiempo medio para reparar el sistema (**MTTR**, *Mean Time To Repair*). Es el tiempo medio necesario para reparar el sistema de una avería. Su valor, se puede estimar como $\frac{\sum_{i=1}^{N} t_i}{N}$ donde t_i es el tiempo necesario para reparar la avería i-ésima y N es el número total de averías ocurridas. Asumiendo que una vez reparado el sistema éste recupera totalmente sus capacidades, se puede establecer que MTTR = MTBF-MTTF, de forma que el tiempo medio entre fallos es la suma del tiempo necesario para reparar el sistema más el tiempo esperado para que suceda otro error.

2.4.4 Cobertura de fallos

Uno de los parámetros más significativos para evaluar los mecanismos de tolerancia a fallos es la cobertura de fallos. La cobertura de fallos puede referirse a las distintas capacidades que introducen las estructuras tolerantes a fallos, dependiendo de los mecanismos tolerantes disponibles en el sistema y los requisitos impuestos por el diseñador. Por ejemplo, se puede medir la cobertura de detección de fallos, la cobertura de recuperación de fallos, o de localización. Lo más habitual es utilizar el término de "cobertura de fallos" como la cobertura de recuperación tras un fallo.

La cobertura de recuperación se define matemáticamente como la probabilidad de que el sistema se recupere tras un fallo. Esta función de probabilidad depende del instante de tiempo en el que se produce el fallo, la localización y tipo del mismo y la actividad del sistema. Dicha función de probabilidad suele ser desconocida, por lo que el cálculo analítico de la cobertura de fallos es muy difícil e inusual. Normalmente, se realizan medidas estadísticas de forma experimental. El método más común consiste en desarrollar una lista de todos los posibles fallos que pueden afectar al sistema y analizar cuantos de esos fallos se pueden detectar, localizar o recuperar. Entonces, la cobertura de fallos sería la relación entre los fallos detectados o recuperados y el número total de fallos. También suele denominarse *tasa de error*. Para realizar este proceso, lo primero es definir qué puede ser un error y a qué partes del sistema puede afectar. El presente trabajo de tesis se centra en los fallos de tipo SEU, que son fallos transitorios en elementos de memoria, como se ha explicado en el apartado 2.2.

2.4.5 Latencia y propagación del error

La latencia de los errores y su propagación a través del sistema son dos parámetros que permiten al diseñador analizar los errores originados por un fallo. La latencia de un fallo es el tiempo transcurrido desde el instante en que se produce el fallo hasta que el error que provoca es detectado, enmascarado o provoca una avería. Por otro lado, una vez generado un error, este puede generar errores en otras partes del dispositivo propagándose el efecto debido a la actividad del sistema.

En un sistema tolerante a fallos es deseable que las latencias sean pequeñas y que el fallo permanezca contenido en una zona determinada del circuito. En caso contrario, un fallo latente puede dar lugar a una avería en cualquier momento, en especial si el efecto del fallo puede propagarse fácilmente hacia las salidas del sistema. El análisis de estos parámetros permite determinar dónde es necesario insertar detectores o estructuras tolerantes, es decir, identificar las partes más sensibles del circuito, facilitando el diseño endurecido del mismo.

2.4.6 Análisis de las propiedades de la confiabilidad

Las características de la confiabilidad pueden cuantificarse para evaluar el nivel de confiabilidad que tiene un sistema. Así pues, se obtienen estimadores para la fiabilidad, disponibilidad, seguridad y mantenibilidad. Estas medidas pueden hacerse experimentalmente o analíticamente. El método experimental consiste en observar las propiedades y el funcionamiento del sistema cuando opera normalmente. Por ejemplo, en el caso de la fiabilidad (probabilidad de que un opere correctamente a lo largo de un periodo de tiempo (t_0, t) dado que funcionaba en t_0), habría que disponer de una serie de sistemas idénticos funcionando normalmente y observar cuantos de ellos fallan en un intervalo de tiempo establecido. Sin embargo, este método requiere una cantidad de tiempo y recursos (réplicas del mismo sistema suficientes para obtener un resultado estadísticamente significativo) inaceptable. La alternativa es utilizar métodos analíticos basados en modelos probabilísticos, como modelos de Markov que se describen en el capítulo 3.

2.5 Resumen y conclusiones

En este capítulo se han definido los conceptos más relevantes relativos a la tolerancia a fallos en circuitos digitales. La tolerancia a fallos es la propiedad de un sistema digital por la cual se asegura una cierta calidad en el funcionamiento de dicho sistema en presencia de fallos. Por lo tanto, la calidad del servicio prestado por el sistema o confiabilidad es el objetivo a cubrir a la hora de diseñar un sistema tolerante a fallos. La confiabilidad engloba distintos conceptos como la fiabilidad, la seguridad, la disponibilidad, etc.

Los sistemas digitales se utilizan en un número creciente de aplicaciones, muchas de las cuales deben ser confiables puesto que un fallo podría originar grandes daños económicos o incluso humanos, como por ejemplo las aplicaciones médicas. Por otro lado, el desarrollo de las tecnologías actuales ha dado lugar a un aumento de la sensibilidad a las condiciones del entorno incrementándose la tasa de fallos que sufre un circuito durante su vida útil por interferencias con el entorno. Por estas razones, el diseño de sistemas tolerantes a fallos es un tema de creciente importancia.

El primer paso para conseguir que un circuito sea tolerante a fallos es definir y determinar qué tipo de fallos pueden afectarlo. En este capítulo se han descrito los principales efectos que el entorno provoca en un sistema, especialmente los debidos a la radiación ionizante proveniente del espacio. En concreto los fallos SEU afectan a elementos de memoria por lo que su efecto permanece almacenado en el circuito hasta que se realiza una nueva escritura en la celda afectada. Este fallo, que se modela como una inversión en el contenido del elemento de memoria afectada (*bit-flip*), puede propagarse por el sistema y dar lugar a una avería si alcanza las salidas. Los fallos SEUs son los efectos en los que se centra el presente trabajo de tesis.

El proceso por el cual se diseña un sistema tolerante a fallos consta de dos tareas: la aplicación de técnicas de tolerancia a fallos o proceso de endurecimiento y la evaluación del nivel de confiabilidad del sistema. Se ha presentado el estado de la técnica de los métodos de evaluación de tolerancia a fallos. Se puede obtener confiabilidad con soluciones tecnológicas o con soluciones basadas en diseño. Las soluciones tecnológicas (*rad-hard*) implican una modificación en el proceso de fabricación por lo que son más caras que las soluciones basadas en el diseño y su uso queda restringido a aplicaciones con alto presupuesto, como las aplicaciones aeroespaciales. Dentro de las técnicas basadas en diseño las más utilizadas son aquellas que se aplican a nivel de sistema o de transferencia de registros porque se introducen durante la etapa de diseño y suponen un coste bajo en relación con otras soluciones. Consisten en añadir redundancia a la información almacenada, ya sea *hardware*, de información, temporal o *software*. En este capítulo se describen las técnicas de redundancia más importantes como la triplicación de módulos con votación (TMR), los códigos de paridad, de Hamming, etc. En el caso de que el dispositivo a endurecer sea una FPGA en vez de un ASIC hay otros efectos a considerar, por lo que son necesarias técnicas de mitigación de fallos específicas. Se han presentado de forma breve las diferencias y particularidades existentes y la complejidad del problema. Así pues, el estudio de la tolerancia a fallos en FPGA es un tema amplio que actualmente atrae el interés de muchos investigadores pero que queda fuera del objeto de este trabajo de tesis.

Una vez aplicadas las técnicas necesarias para lograr que un sistema sea tolerante a fallos, debe evaluarse el nivel de confiabilidad alcanzado. Esta evaluación suele utilizarse también al inicio del proceso de endurecimiento para detectar las zonas del circuito o partes del sistema en las que es necesario insertar estructuras redundantes. Como en todo proceso de medida hay que definir parámetros con los que cuantificar la confiabilidad. Es deseable que estos parámetros se puedan calcular o estimar con anterioridad a la puesta en funcionamiento del sistema bajo estudio. Para ello existen métodos analíticos basados en técnicas probabilísticas. Sin embargo, debido a la complejidad de los diseños actuales el cálculo analítico suele ser inviable y es necesario recurrir a métodos experimentales.

En el siguiente capítulo se describen los distintos métodos para la evaluación de la tolerancia a fallos de un circuito digital. Se presentan las propuestas realizadas hasta el momento por otros autores, analizando sus ventajas y limitaciones.

CAPÍTULO 3

MÉTODOS PARA LA EVALUACIÓN DE LA TOLERANCIA A FALLOS

3. Métodos para la Evaluación de la Tolerancia a Fallos

En el capítulo anterior se presentaron las técnicas existentes para obtener un sistema tolerante a fallos. En el proceso de aplicación de dichas técnicas o proceso de endurecimiento, es fundamental medir la tolerancia a fallos alcanzada. En este capítulo se describen los distintos métodos para la evaluación de la tolerancia a fallos, dando especial importancia a las técnicas de inyección de fallos de las cuales se presenta el estado de la técnica.

3.1 Introducción

La evaluación de la confiabilidad es una tarea esencial dentro del proceso de diseño de circuitos tolerantes a fallos. El principal objetivo del proceso de evaluación es verificar si el circuito bajo estudio tiene el nivel de tolerancia a fallos adecuado. Además, se utiliza para predecir el comportamiento del circuito en presencia de fallos y proporciona información durante el diseño sobre las partes del circuito que necesitan ser endurecidas. Existen distintos métodos para realizar la evaluación del nivel de confiabilidad de un sistema. Básicamente se puede distinguir entre métodos analíticos y métodos experimentales [Bens03][Prad96].

Los **métodos analíticos** tienen como objetivo obtener un valor numérico de las propiedades de la confiabilidad (fiabilidad, disponibilidad, seguridad, etc.), utilizando modelos matemáticos basados en teorías estadísticas y de probabilidad, como por ejemplo modelos de Markov, redes de Petri estocásticas o modelos combinatorios. Los métodos analíticos permiten caracterizar los mecanismos de tolerancia a fallos de forma exacta y predecir su comportamiento frente a fallos cuando el modelo desarrollado es preciso. Sin embargo, la representación matemática de circuitos reales es una tarea de gran dificultad y los modelos que se obtienen suelen ser demasiado complejos para realizar los análisis. Cuando se aplican simplificaciones en dicho modelado se pierde precisión reduciéndose la validez de los resultados obtenidos. Además, en el caso de utilizar IPs (*Intellectual Property*), lo cual es muy habitual en los sistemas actuales, la estructura interna del circuito es desconocida y no se dispone de la información necesaria para desarrollar un modelo matemático del circuito.

Los **métodos experimentales** se basan en medir directamente los parámetros de confiabilidad del sistema. Se distinguen dos mecanismos diferentes:

- Observar y registrar el comportamiento del circuito durante su fase de operación estudiando el efecto que provocan los fallos reales[9]. De esta forma es posible obtener datos como el tiempo medio entre fallos o la probabilidad de

[9] En inglés *life testing* o *real-time testing* o *field testing*

suceso de un fallo, aparte de la cobertura de fallos y otras características de los mecanismos de tolerancia a fallos disponibles.

- Introducir fallos en el circuito artificialmente, es decir, inyectar fallos, y observar su efecto.

La observación y medida del sistema durante la fase de operación con una carga de trabajo real proporciona información sobre los errores y averías que se producen de forma natural. Esta información incluye datos referentes a la sensibilidad de la tecnología en el entorno real en el que opera el sistema, a la frecuencia con la que se producen los fallos en dicho entorno y al comportamiento real de los mecanismos de tolerancia a fallos de los que dispone el sistema. Es, por tanto, el método que ofrece los resultados más realistas.

La medida de la tolerancia a fallos en la fase de operación no es un método adecuado para predecir el comportamiento del sistema frente a fallos ni para facilitar la inserción de técnicas de redundancia durante su diseño. Se utiliza principalmente para verificar las hipótesis realizadas en los modelos analíticos y validar otros métodos de evaluación de la confiabilidad utilizados en fases anteriores del ciclo de diseño. Para realizar una experiencia en la fase operacional es necesario disponer de un sistema real y utilizar un tiempo de test adecuado. Teniendo en cuenta que la tasa de averías de sistemas críticos tolerantes a fallos puede estar comprendida entre 10^{-5} y 10^{-9} averías/hora, el tiempo necesario para obtener datos estadísticamente relevantes puede ser excesivo (cientos de años). Para reducir el tiempo de observación necesario, estos experimentos se desarrollan en estaciones situadas a una altitud elevada para que el flujo de partículas sea mayor que a nivel terrestre. Otra forma de reducir el tiempo de observación consiste en realizar el test en tiempo real sobre varios dispositivos en lugar de sobre un único circuito de manera que la probabilidad de que ocurra un fallo aumenta. Aún así este método de medida no es viable para todos los circuitos que se diseñan y fabrican en la actualidad.

La forma de acelerar la medida experimental de la tolerancia a fallos de un sistema es inyectar fallos artificialmente para observar la respuesta del circuito. Como se ha indicado en el apartado de introducción, las técnicas de inyección de fallos pueden aplicarse en distintas etapas del ciclo de diseño y tienen los siguientes objetivos: validar las técnicas de tolerancia a fallos implementadas en el sistema, ayudar en el proceso de endurecimiento al diseñador indicando qué partes del circuito no cumplen con las especificaciones requeridas de confiabilidad y prever cuál será el comportamiento del circuito en presencia de fallos. La inyección de fallos se aplica de forma relativamente rápida y eficaz en circuitos reales y complejos. Por lo tanto, presenta soluciones a los principales inconvenientes de aplicar métodos analíticos o realizar experimentos en

campo. Estas características hacen de dichas técnicas un método potente y ampliamente aceptado para la evaluación de la tolerancia a fallos de un circuito.

Para hacer un análisis completo y preciso de las propiedades de confiabilidad de un sistema es necesario aplicar distintas clases de métodos. Los experimentos en la fase de operación normal en combinación con métodos analíticos son necesarios para obtener medidas reales de confiabilidad, validando las técnicas de inyección. Las recientes experiencias publicadas en [Lese05] y [Autr06] son ejemplos de la utilidad y necesidad de realizar estos estudios. [Lese05] presenta los resultados de sensibilidad a SEUs obtenidos con experimentos en campo de FPGAs de Xilinx fabricadas en diferentes tecnologías CMOS. El experimento consiste en múltiples conjuntos de 100 unidades del dispositivo a estudiar localizados a diferentes altitudes. Los resultados presentados recogen los datos observados durante más de un año. [Autr06] presenta los resultados preliminares del proyecto ASTEP (*Altitude SEE Test European Platform*), localizado en los Alpes franceses. El objetivo de este proyecto, operativo desde marzo del 2006, es estudiar la tasa de SEE en memorias SRAM. Sin embargo, las campañas de experimentos en tiempo real son muy costosas tanto por el tiempo necesario como por las instalaciones y dispositivos empleados, por lo que este tipo de experimentos no son muy numerosos. Estas razones, junto con la necesidad de aplicar métodos durante el diseño de circuitos o sistemas tolerantes a fallos hacen que las técnicas de inyección sean una solución muy interesante y extendida. A continuación se presenta una revisión del estado de la técnica de la inyección de fallos.

3.2 Descripción general de las técnicas de inyección de fallos

Las técnicas de inyección de fallos estudian la tolerancia a fallos de un circuito comparando el comportamiento del circuito sin fallo con su comportamiento en presencia de fallos insertados artificialmente. Para ello, son necesarias herramientas específicas para la inyección de los fallos y la monitorización de sus efectos.

En [Arla90] se presenta el modelo ***FARM*** que describe los principales elementos involucrados en la implementación y desarrollo de técnicas de inyección de fallos. Según este modelo, una campaña de inyección de fallos está caracterizada por cuatro elementos:

- El conjunto de fallos ***F*** (*fault*) a inyectar.
- El conjunto de entradas al circuito ***A*** (*activation*) que especifica el dominio usado para ejercitar las funciones del sistema.
- El conjunto de resultados que deben ser leídos ***R*** (*readout*).
- Las medidas ***M*** (*measurement*) derivadas de los resultados leídos.

Un experimento de inyección consiste en seleccionar un fallo del conjunto *F* y activar una carga de trabajo *A*. Se observan las reacciones del circuito obteniendo los datos *R* necesarios que permitan obtener las medidas de la confiabilidad perseguidas *M*. El primer requisito a cubrir a la hora de realizar una inyección de fallos es determinar y acotar el espacio de fallos *F* frente a los cuales el circuito debe ser tolerante a fallos. El espacio de fallos de un circuito es multidimensional, donde las dimensiones son el instante de tiempo en el que sucede el fallo y su duración, el tipo de fallo y la localización.

El presente trabajo de tesis se centra en fallos transitorios de tipo SEUs por su importancia en los circuitos actuales, por lo que, utilizando el modelo *bit-flip* descrito en el capítulo 2, un fallo puede afectar a cualquier elemento de memoria en cualquier instante de tiempo. De esta forma, el problema de la inyección de fallos SEUs es bidimensional, siendo necesario fijar la localización y el tiempo para caracterizar un fallo. El carácter bidimensional de los fallos transitorios dificulta enormemente la evaluación de la tolerancia a fallos puesto que es necesario controlar espacial y temporalmente la inyección de dichos fallos. Además el gran número de posibles fallos ha llevado a numerosos investigadores a descartar la inyección exhaustiva, limitando el espacio de fallos a un subconjunto de todos los posibles. La selección de dicho subconjunto se realiza utilizando distribuciones de probabilidad (uniforme o de Poisson) bajo la asunción de que dicho subconjunto es representativo y permite generalizar los resultados obtenidos.

El comportamiento de un circuito y de los mecanismos de mitigación de fallos depende de la carga de trabajo que se ejecuta. Así pues, los resultados de la evaluación de la confiabilidad van a verse influidos por la actividad del circuito. Cada posible actividad del circuito se describe con un banco de pruebas formado por un conjunto de vectores de entrada. Estos vectores deben ser elegidos de forma que el circuito ejercite la mayor funcionalidad posible o al menos aquellas partes que se consideran más críticas cuando el objetivo principal es validar los mecanismos de tolerancia a fallos. Cuando el principal objetivo de la campaña de inyección es predecir el funcionamiento del circuito en presencia de fallos se utilizan cargas de trabajo típicas, representativas de la carga de trabajo que tendrá el circuito en su modo normal de operación. Fijar una carga de trabajo influye en la selección de los fallos a inyectar en la medida en que determina el tiempo de ejecución y las partes del circuito que se activan. Por lo tanto, la actividad determina los posibles instantes de inyección y las localizaciones del circuito donde es interesante inyectar (introducir fallos en las zonas inactivas de un circuito no implica ningún efecto en su funcionamiento).

Los resultados leídos consisten en una clasificación de los efectos producidos por cada fallo inyectado. Esta clasificación se obtiene comparando la respuesta del

circuito en presencia de fallos con su respuesta sin fallos. Generalmente, un fallo puede tener los siguientes efectos:

- Es una avería (*failure*) cuando la inyección de fallo provoca un funcionamiento erróneo del sistema.
- Es silencioso (*silent*) cuando el efecto desaparece por completo sin producir ninguna avería, bien porque un mecanismo de tolerancia a fallos lo ha enmascarado (por ejemplo una estructura TMR cuando el fallo se produce en uno de los biestables que lo forma) o bien por la propia actividad del sistema (como por ejemplo ocurre cuando un fallo se inyecta en un biestable justo en el instante anterior a que se cargue un nuevo valor).
- Es latente (*latent*) cuando el efecto permanece almacenado en el circuito pero no ha provocado ninguna avería durante el tiempo que dura el experimento de inyección. Este tipo de efectos deben ser estudiados con atención porque pueden dar lugar a un mal funcionamiento si la ejecución continúa.
- El fallo se clasifica como detectado (*detected*) cuando existe un mecanismo de tolerancia a fallos que detecta el error, generalmente, con el objetivo de activar la recuperación del circuito de dicho de fallo.

Esta clasificación general puede ser modificada con el objetivo de obtener más información o por el contrario resumirla. Por ejemplo, en microprocesadores es interesante distinguir entre distintos tipos de averías, diferenciándose entre errores en las salidas o pérdidas de la secuencia de ejecución. Por otro lado, en algunos circuitos con escasa observabilidad no es viable comprobar si el fallo permanece almacenado o no en el circuito para distinguir entre fallos latentes y silenciosos, por lo que la clasificación se reduce a diferenciar entre aquellos fallos que causan averías y los que no.

Adicionalmente a la clasificación de fallos, también denominada comúnmente en la literatura como diccionario de fallos, pueden medirse datos sobre la evolución del error y su propagación a través del circuito. Estos datos permiten medir parámetros de confiabilidad como la cobertura de fallos o las latencias de propagación de errores.

Una vez descritos los atributos que caracterizan la problemática de los métodos de inyección de fallos falta describir el sistema de inyección necesario para implementar esta solución, lo que se conoce como entorno de inyección.

En [Hsue97] se presenta un esquema general con los componentes básicos del entorno de inyección, véase la Figura 20. El *módulo de inyección* selecciona un fallo de la lista de todos los posibles fallos a inyectar y lo inserta en el circuito a evaluar. El *banco de pruebas* almacena los vectores de entradas que definen la actividad del

circuito que se desea probar. El *módulo de observación* monitoriza el funcionamiento del diseño y realiza la lectura de datos. Finalmente, estos datos son procesados en el bloque *analizador de resultados*. Todo el proceso de inyección es dirigido por el bloque *controlador* que se encarga de aplicar las entradas correspondientes a cada instante de tiempo al circuito, activar la inyección en función del tiempo de inyección seleccionado e iniciar la lectura de resultados.

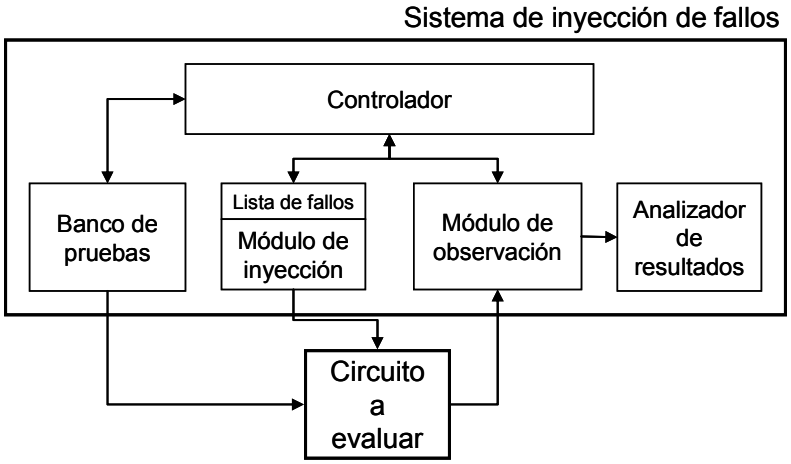


Figura 20. Entorno básico de un sistema de inyección

Dependiendo de cómo se implementa el entorno de inyección, las distintas técnicas se pueden agrupar en tres categorías diferentes [Bens03]:

- Técnicas basadas en simulación. Mediante una herramienta de simulación se insertan fallos en un modelo del circuito a estudiar.
- Técnicas basadas en inyección *hardware*. Estas técnicas se basan en la inserción de fallos en una implementación real del circuito o prototipo. Los fallos insertados son fallos físicos que producen pulsos de tensión reales en nodos del circuito.
- Técnicas basadas en inyección *software*. Se basan en modificar el *software* a ejecutar en el circuito bajo análisis, normalmente un procesador, para proporcionar la capacidad de modificar el estado del sistema de acuerdo con el modelo de fallo dado.

Esta clasificación es utilizada tradicionalmente en la literatura, entrando en desuso la clasificación que inicialmente se presentó en [Prad96], basada en la fase del ciclo de diseño en la que se aplicaba la inyección (fase de diseño o fase de prototipado).

Los métodos basados en simulación pueden aplicarse a cualquier nivel de abstracción. Estas técnicas se utilizan durante la fase de diseño, lo cual facilita el diseño de circuitos tolerantes a fallos, y permiten una controlabilidad y observabilidad total de las señales del circuito. Su principal limitación es que son técnicas muy lentas,

especialmente en el caso de circuitos complejos, y requieren un alto coste computacional.

Por otro lado, las técnicas implementadas en *software* o en *hardware* se aplican en prototipos o implementaciones reales, por lo que no es necesario disponer de un modelo del circuito. Estas técnicas realizan la inyección de fallos en tiempo real, siendo en general más rápidas que las técnicas basadas en simulación. Sin embargo, las técnicas de inyección *hardware* requieren de equipos especiales para la generación de los fallos y la mayoría de dichas técnicas proporcionan una controlabilidad y observabilidad de los fallos y sus efectos limitada. Las técnicas de inyección *software* sólo permiten la inyección de fallos en aquellas partes del circuito accesibles por *software* y son técnicas intrusivas porque implican la modificación del código original.

Algunas de las técnicas de inyección desarrolladas en los últimos años no se ajustan realmente a ninguna de las categorías que se acaban de describir y que se utilizan tradicionalmente. Las técnicas de emulación en FPGAs consisten en prototipar una descripción del diseño en una FPGA y realizar la inyección sobre dicho prototipo (la emulación de fallos se describe con detalle en el apartado 3.3.2). En la literatura la emulación suele considerarse una técnica para acelerar la simulación. Sin embargo, la inyección se realiza sobre un soporte *hardware* que, además de acelerar el proceso de inyección, permite obtener información adicional a la proporcionada por la simulación. La emulación incluye detalles de implementación y de las condiciones reales de funcionamiento, permitiendo estudiar el comportamiento del circuito bajo fallos cuando interacciona con otros circuitos en condiciones muy similares a las reales. Esta característica es muy interesante en el caso de evaluar componentes de un SoC pero, obtener dicha información mediante simulación sería muy costoso. Por lo tanto, no parece adecuado clasificar las técnicas basadas emulación dentro de la categoría de simulación. Tampoco se considera adecuado clasificar la emulación dentro de las técnicas de inyección *hardware* porque la emulación está orientada a la inyección en diseños de ASICs y, por lo tanto, el prototipado del circuito en la FPGA no es un ejemplo real del circuito que finalmente será fabricado (el circuito final no es implementado en un FPGA). En consecuencia, proponemos una agrupación de técnicas distinta, en función de si la inyección se realiza sobre una descripción del circuito, como ocurre en los métodos de simulación y emulación, o sobre una implementación *hardware* real, bien sobre un prototipo o sobre un componente comercial.

3.3 Técnicas de inyección de fallos sobre una descripción del circuito

Las técnicas de inyección de fallos aplicadas sobre una descripción del circuito pueden utilizarse durante cualquier etapa del ciclo de diseño. Lo más habitual es aplicarlas en la etapa de diseño por dos razones:

- Disponer de la descripción del circuito permite tener acceso completo a todas las partes del circuito, pudiendo inyectarse en cualquier localización y observar la evolución del efecto del fallo con el máximo detalle posible. Esta información es fundamental para evaluar las funciones de los mecanismos de tolerancia a fallos y es la mejor solución para detectar las zonas críticas y menos robustas del circuito donde es necesario mejorar la tolerancia a fallos.
- El rediseño del circuito es más rápido y el coste es mínimo en esta etapa del ciclo de diseño. Por lo tanto, evaluar la tolerancia a fallos de un circuito durante la etapa de diseño facilita y reduce el coste de posibles modificaciones necesarias cuando la tolerancia a fallos no es adecuada, es insuficiente o se ha sobredimensionado.

Dentro de estas técnicas se pueden distinguir dos tipos, las técnicas basadas en simulación y las técnicas basadas en emulación. Las técnicas de inyección de fallos basadas en simulación realizan la inyección mediante *software*, utilizando únicamente herramientas CAD (*Computer Aided Design*) para realizar todas las tareas de la inyección y pueden aplicarse a distintos niveles de abstracción. Por otro lado, en los métodos basados en emulación se recurre a un soporte *hardware*, una FPGA, para implementar al menos el circuito bajo estudio.

3.3.1 Simulación

Las técnicas de inyección de fallos basadas en simulación se implementan con herramientas CAD. Para aplicar dichas técnicas es necesario disponer de una descripción del circuito y de un modelo de los fallos a inyectar. Estos se pueden describir en distintos niveles de abstracción:

- Nivel físico. Es el nivel de abstracción en el que el circuito se describe en función de su *layout*, es decir, en función de las máscaras de difusión utilizadas en el proceso de fabricación del circuito.
- Nivel eléctrico o de transistor, cuando se describe el circuito en función del modelo del transistor, considerando sus parámetros eléctricos.
- Nivel lógico o de puerta. El circuito se describe mediante la interconexión de bloques básicos, como puertas lógicas y biestables.

- Nivel de transferencia de registros (RT). Los elementos básicos en este nivel son registros, memorias, operadores, unidades lógicas y aritméticas y buses.
- Nivel de sistema. El diseño se describe como el conjunto de los componentes que lo forman, como pueden ser microprocesadores, módulos de memoria, dispositivos de entrada/salida, etc.

La precisión del modelo de circuito y del modelo de fallo utilizados condiciona la validez de los resultados obtenidos. Los niveles de abstracción más bajos permiten estudiar con más precisión los fenómenos físicos reales que generan los fallos y sus efectos. Las simulaciones a nivel físico proporcionan información sobre la sensibilidad de una tecnología dada a los efectos de la radiación. En [Lamb05] se analiza la sensibilidad de tecnología SOI a SEUs, estudiando las interacciones de los neutrones con el silicio y el oxigeno mediante una herramienta de simulación denominada MC-DASIE. Dicha herramienta utiliza un método de Monte Carlo para la evaluación de la tasa de *soft error* (SER). En el nivel eléctrico los fallos transitorios se simulan como pulsos de corrientes o de tensión. Estas simulaciones tienen en cuenta los parámetros de la tecnología y los retardos causados por las transiciones en los transistores, lo que permite obtener información sobre la duración de los pulsos transitorios que se generan y los efectos a los que da lugar. Para realizar simulaciones de fallos a nivel eléctrico se utilizan simuladores comerciales como SPICE.

Los trabajos desarrollados para simulación en los niveles de abstracción físico o eléctrico son necesarios para la comprensión de los mecanismos de fallos y de cómo afectan en los distintos niveles de abstracción. Sin embargo, simular un circuito física o eléctricamente es una tarea muy lenta, de forma que el análisis de la confiabilidad de un diseño complejo, con una gran cantidad de nodos a considerar, mediante simulación a niveles bajos de abstracción es muy costoso o incluso inviable. [Choi92] describe el entorno de inyección *FOCUS* que simula fallos a nivel eléctrico para analizar su propagación e impacto a más altos niveles de abstracción. La inyección de 2.100 fallos con esta técnica duró tres semanas. Este dato ilustra el alto coste computacional que suponen estas técnicas de simulación debido a la gran cantidad de información y detalle de los modelos.

A nivel de transistor se pude utilizar una de las pocas herramientas comerciales disponibles para la evaluación de SEUs. La empresa iRoC Technologies Inc. ha desarrollado la herramienta TFIT™ que permite evaluar la tasa de *soft error* (SER) durante el diseño de un circuito a nivel de transistor. La simulación se realiza utilizando un simulador SPICE, Figura 21.

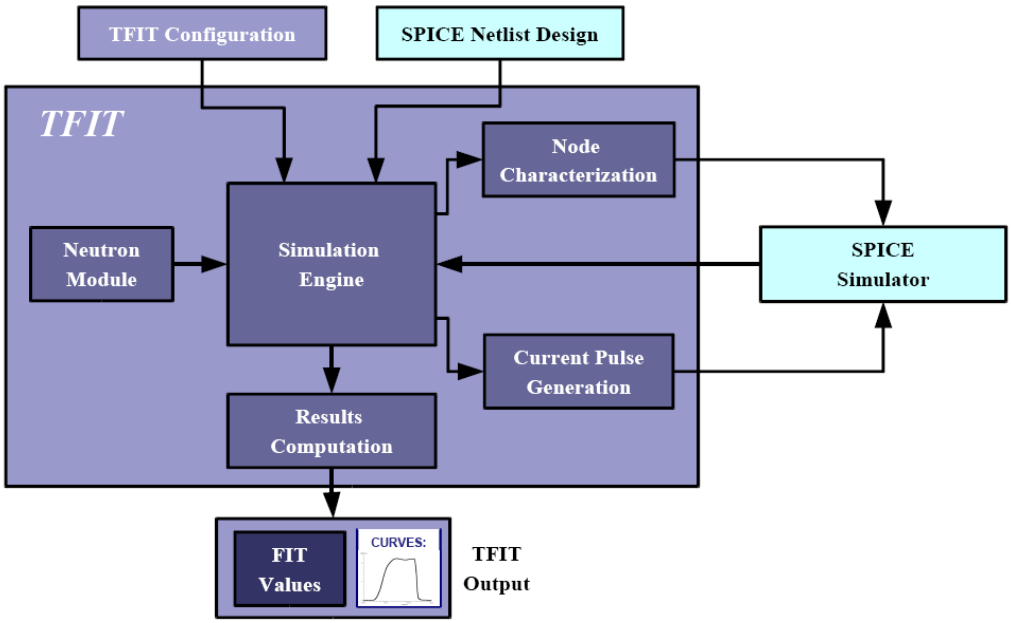


Figura 21. Flujo de simulación de TFIT™ [iRoC]

Cuando el objetivo es la simulación de circuitos VLSI complejos las descripciones se realizan en los niveles altos de abstracción, como el nivel de puerta, el de transferencia de registros (RT) o el nivel de sistema, lo que disminuye la complejidad de las simulaciones y consume menos tiempo. El nivel de sistema se usa para estudiar redes de procesadores ya que es la mejor manera de describir el comportamiento *hardware* y *software* de sistemas complejos con precisión. Sin embargo, existen ciertos aspectos que dificultan el desarrollo de herramientas de simulación a este nivel de abstracción:

- La gran variedad de componentes que pueden constituir un sistema complican el desarrollo de herramientas de simulación de propósito general.
- No es usual disponer de un modelo de simulación del sistema completo, por lo que es necesario desarrollarlo para poder aplicar la simulación funcional.

[Gosw97] presenta la herramienta de simulación DEPEND, donde el modelo de simulación del sistema se describe en el lenguaje de programación orientado a objetos C++. Con objeto de facilitar el desarrollo del modelo del sistema, la herramienta dispone de una biblioteca de objetos que describen algunas de las funciones más habituales en las arquitecturas tolerantes a fallos, como redundancia NMR o votadores. DEPEND realiza una simulación jerárquica para acelerar el proceso de simulación. Analiza los submódulos individualmente y se combinan sus resultados para obtener el resultado global correspondiente al sistema completo.

SOCFIT™ es una herramienta comercial desarrollada por iRoC Technologies Inc. [iRoC] que evalúa la sensibilidad de un SoC durante la fase de diseño y mide la tasa de averías (FIT) a nivel de sistema.

Los niveles lógicos y RT son más apropiados para simular componentes individuales y estudiar sus mecanismos de tolerancia a fallos. En estos niveles de abstracción los lenguajes de descripción *hardware* (HDL, *Hardware Description Language*), como VHDL (*VHSIC Hardware Description Level*) o Verilog, son ampliamente utilizados en la descripción de circuitos integrados digitales por los diseñadores. En consecuencia, las herramientas de simulación basadas en estos lenguajes de descripción son de gran interés en la evaluación de la tolerancia a fallos durante la etapa de diseño de un CI. A continuación, se analiza este tipo de técnicas y se presentan las herramientas más relevantes desarrolladas hasta el momento.

3.3.1.1 Inyección de fallos mediante simulación de descripciones HDL

En la descripción de un circuito juegan un papel fundamental los lenguajes de alto nivel de descripción *hardware* ya que permiten diseñar en altos niveles de abstracción simplificándose el diseño de circuitos integrados complejos. Por esta razón se usan comúnmente en la descripción de circuitos lo que ha llevado, como veremos en este apartado, al desarrollo de diversas herramientas para la evaluación de la tolerancia a fallos. Este método permite integrar la tarea de evaluación en el proceso de diseño usando el mismo entorno.

Mayoritariamente las herramientas desarrolladas hasta el momento se basan en el uso de VHDL, aunque existen algunas que utilizan modelos Verilog. En [Zara03] se presenta una herramienta de simulación, denominada INJECT, para la inyección de fallos transitorios y permanentes en modelos Verilog[10]. La herramienta INJECT soporta la inyección de fallos en circuitos descritos a nivel de conmutación, es decir, como la red de transistores que lo conforma, considerando cada transistor como un conmutador ideal. El objetivo de la herramienta es estudiar los efectos de los fallos en distintos niveles de abstracción incluyendo los niveles bajos para modelar de forma fiable los fenómenos importantes de circuitos CMOS.

3.3.1.1.1 Técnicas y herramientas existentes basadas en simulación de descripciones VHDL

La inyección de fallos mediante simulación de descripciones VHDL puede implementarse de tres maneras diferentes:

- Modificando la descripción VHDL del circuito.
- Utilizando comandos de un simulador VHDL comercial.
- Modificando las herramientas de simulación.

[10] Verilog es más potente en niveles bajos de abstracción que VHDL.

Dentro de la categoría de las **técnicas que modifican el modelo VHDL** se pueden distinguir a su vez dos tipos de soluciones:

- **Saboteadores** (o perturbadores). Solución basada en la adición de componentes dedicados a la inyección de fallos y denominados saboteadores [Boue98][Card02]. Estos componentes alteran el valor o las características temporales de una o varias señales cuando se activa el fallo. Durante la operación normal permanecen inactivos y se activan sólo para inyectar el fallo. Se distinguen dos estructuras, en serie y en paralelo (Figura 22). Los saboteadores en serie se interponen entre una o varias salidas (emisor de la señal) y su correspondiente entrada o entradas (receptor). Los saboteadores en paralelo se implementan como una fuente más de la señal de forma que cuando se active modifique el resultado de la función de resolución. Un método equivalente sería modificar la propia función de resolución [Delo96].

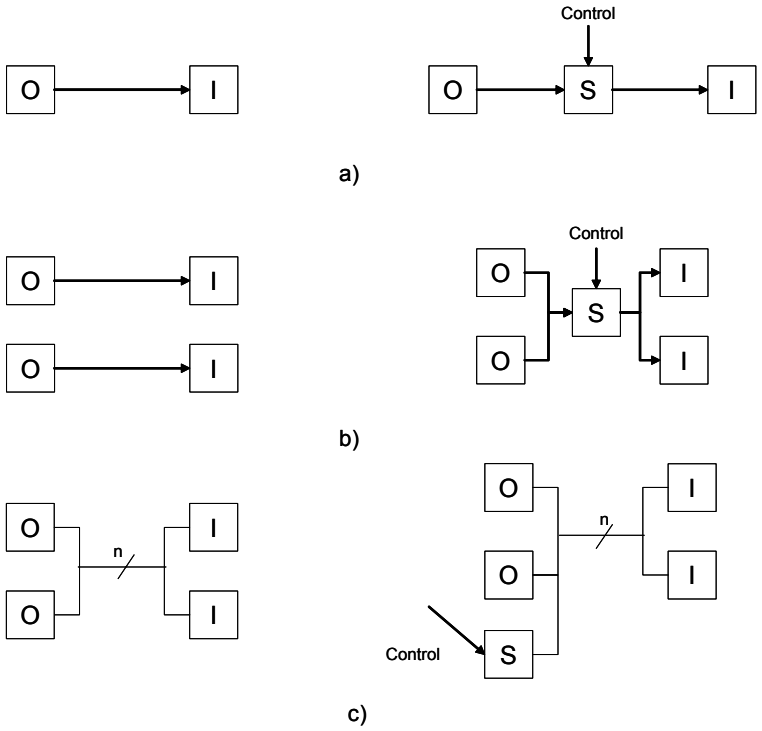


Figura 22. Distintas estructuras de saboteadores (derecha) y el circuito original en el que se insertan (izquierda) [Jenn94]. a) Saboteador en serie simple. b) Saboteador en serie complejo. c) Saboteador en paralelo

- **Mutantes.** Son técnicas basadas en la modificación de descripciones de componentes ya existentes [Jenn94]. Un mutante es una descripción de un componente que reemplaza otra descripción. Cuando está inactivo se comporta como el componente al que sustituye. La mutación puede llevarse a cabo de varias formas: reemplazando componentes en las descripciones estructurales (por ejemplo, reemplazar una puerta NAND por una NOR) o modificando

manualmente sentencias en descripciones comportamentales, por ejemplo, generando operadores erróneos o cambiando identificadores de variables.

Por otro lado, están las técnicas que usan **órdenes del simulador** en vez de modificar la descripción del diseño [Berr02a][Jenn94]. Permiten la lectura y escritura de todas las señales y variables de la descripción del circuito. Sin embargo, al no existir simuladores comerciales específicos para la inyección de fallos, es preciso usar simuladores de propósito general y las funciones que estos proporcionan. Estas técnicas implican detener la simulación y lanzarla de nuevo para cada fallo que se inyecta, lo que supone un tiempo excesivamente alto cuando el número de fallos es grande.

El método que proporciona unas mayores prestaciones es la **modificación de las herramientas de simulación** con el objetivo de que el propio simulador disponga de las capacidades de inyección y observación de resultados sin la necesidad de módulos adicionales. [Sieh97] presenta la herramienta VERIFY que utiliza señales especiales para especificar la duración media de los fallos, el intervalo entre fallos y el tipo de fallo. Se implementan en las descripciones comportamentales de los componentes básicos. Por lo tanto es necesario el uso de un compilador específico que maneje dichas extensiones y un simulador para realizar los experimentos de inyección. Es el propio simulador el que determina la localización y el instante del fallo, sin ninguna actuación del usuario, puesto que esa información está contenida en las nuevas señales.

Algunas herramientas implementan distintas técnicas de inyección, como VFIT [Bara05] que implementa inyección mediante órdenes del simulador, mutantes y saboteadores y puede inyectar fallos transitorios, permanentes e intermitentes.

Las técnicas basadas en comandos del simulador son los métodos que menos esfuerzo de implementación requieren puesto que no es necesario modificar el código VHDL. Sin embargo, su mayor desventaja es que es necesario detener la simulación para realizar la inyección y clasificación de cada fallo, y que estas técnicas generalmente dependen del simulador utilizado. Por otro lado, en las técnicas basadas en modificación del diseño es necesario generar los saboteadores o mutantes, insertarlos en el código y recompilar el modelo VHDL, lo que las hace muy lentas. Además, mientras que los saboteadores son componentes y por lo tanto se pueden reutilizar con facilidad, los mutantes se generan específicamente en cada caso, aunque luego se incluyen en el modelo con más facilidad que los saboteadores, gracias al mecanismo de configuración que ofrece el VHDL. La modificación de las herramientas de simulación es la solución más rápida y la que proporciona mejores prestaciones, como se ha dicho anteriormente, pero sólo puede utilizarse cuando el código de las herramientas de simulación está disponible lo que no es habitual, o desarrollando un simulador específico para fallos lo que es muy costoso.

En resumen, las técnicas basadas en simulación de descripciones VHDL permiten evaluar la tolerancia a fallos de un circuito durante la etapa de diseño, integrándose en el proceso de diseño del circuito endurecido. La simulación de fallos facilita un posible rediseño y proporciona una alta flexibilidad, siendo posible aplicar distintos modelos de fallos en cualquier parte del circuito. Sin embargo, el mayor inconveniente que presentan las técnicas basadas en simulación es el tiempo necesario para realizar una campaña de inyección, mayor cuanto mayor es la complejidad del circuito. Los circuitos actuales, de gran tamaño y complejidad, precisan de métodos y técnicas que aceleren el proceso de evaluación con respecto a las técnicas de simulación.

3.3.1.2 Técnicas para acelerar la inyección de fallos mediante simulación

El mayor inconveniente de las técnicas de simulación de fallos es el tiempo de CPU requerido, lo que las hace muy lentas. Se han desarrollado soluciones con el objetivo principal de acelerar las campañas de inyección mediante simulación. En [Parr00] y [Berr02a] se proponen un conjunto de técnicas que aceleran la inyección de fallos SEU en la descripción VHDL de circuitos digitales. En ambos estudios la inyección se realiza a través de comandos estándar de los simuladores comerciales, por lo que el método no depende de un simulador particular y puede ser fácilmente integrado en cualquier flujo de diseño. Las técnicas que se proponen para minimizar el tiempo de simulación consisten en reducir la lista de fallos a inyectar y disminuir el tiempo empleado en la simulación de cada fallo.

3.3.1.2.1 Reducción de la lista de fallos

La reducción de la lista de fallos, denominada en la literatura colapsación de la lista de fallos consiste en detectar posibles equivalencias entre fallos para clasificarlos según su efecto sin necesidad de realizar su inyección en el circuito, de forma que se reduce en número de fallos a simular y, por lo tanto, se acelera el proceso de evaluación de la tolerancia a fallos. En [Berr02a] se distingue entre las técnicas de colapsación estática de fallos, cuando se aplican antes de realizar la campaña de inyección, y las técnicas de colapsación dinámica si se aplican durante el proceso de inyección.

La **colapsación estática** de fallos puede realizarse en función de la topología del circuito a evaluar (colapsación de fallos independiente de la carga de trabajo) o en base a la actividad realizada (colapsación de fallos dependiente de la carga de trabajo). El estudio de la topología del circuito proporciona información sobre equivalencias entre fallos:

- Todos los fallos que se inyectan en los biestables que están conectados a una salida directamente (salidas registradas) se clasifican como averías.
- Todos los biestables que estén conectados directamente a otro biestable provocarán el mismo efecto que este último y por lo tanto los fallos inyectados en uno u otro serán del mismo tipo.
- En un registro de n bits basta con analizar los fallos en uno de los bits si sobre todos los bits actúan las mismas operaciones.

Las técnicas de colapsación dependientes de la carga de trabajo requieren del análisis del comportamiento del circuito libre de fallos. Es necesario almacenar todas las operaciones de escritura o lectura de cada elemento de memoria del circuito. Las reglas propuestas en [Berr02a] son las siguientes:

- No es necesario inyectar fallos en un registro si éste no va a ser leído después del instante de inyección (fallo latente) o si el registro va a escribirse con un nuevo valor tras el instante de inyección (fallo silencioso).
- Los fallos inyectados entre una operación de lectura o escritura y la subsiguiente operación de lectura son equivalentes.

Por otro lado, a diferencia de las técnicas de colapsación estática, las técnicas de **colapsación dinámica** no suponen un coste adicional en tiempo, puesto que se aplican durante la inyección de fallos. [Berr02a] presenta una técnica de colapsación dinámica basada en el análisis de la propagación del efecto del fallo durante su simulación. Una vez inyectado un fallo (f_i, t_p), donde f_i es el elemento de memoria afectado y t_p es el instante de inyección, se compara el estado del circuito con el estado esperado sin fallos. Si en un instante $t_q > t_p$ el estado del circuito coincide con el estado esperado excepto en un único elemento de memoria f_j, los fallos (f_i, t_p) y (f_j, t_q) son equivalentes. Para aplicar esta técnica es necesario disponer del valor de todos los elementos de memoria para distintos instantes de tiempo, correspondientes a la simulación sin fallos del circuito. Almacenar la información necesaria implica el uso de grandes cantidades de memoria y espacio en disco.

Los resultados presentados en [Berr02a] muestran que la aplicación del análisis de la topología de un diseño reduce la lista de fallos en un 10%, el análisis de la actividad supone un 74% de reducción y la colapsación dinámica evita la simulación de casi un 5% de los fallos durante la campaña de inyección. Estos datos son orientativos puesto que la mejora proporcionada dependerá del circuito bajo estudio y su carga de trabajo.

3.3.1.2.2 Reducción del tiempo empleado en la simulación de cada fallo

[Berr02a][Parr00] presentan técnicas para acelerar la simulación de cada fallo. Se propone almacenar el estado del circuito (contenido de todos los biestables) en determinados instantes denominados puntos de comprobación para reducir el tiempo empleado en la simulación previa al instante de inyección y acelerar la clasificación de cada fallo:

- La simulación de un nuevo fallo se inicia cargando el estado del circuito correspondiente al punto de comprobación más cercano al instante de inyección.
- Tras la inyección del fallo se compara periódicamente el estado del circuito con el estado esperado para detectar el caso en que el efecto del fallo desaparece. Entonces el fallo se clasifica como silencioso. Esta técnica implica un coste en tiempo y en memoria que debe ser minimizado disminuyendo el número de comparaciones a realizar necesarias. En [Berr02a] se indica que la periodicidad de las comparaciones realizadas entre estados del circuito se incrementa de forma exponencial. De forma que la primera comparación se realiza un ciclo de reloj después de la inyección, la segunda tras dos ciclos, después tras cuatro ciclos y así sucesivamente.

Esta técnica está limitada por los requisitos de memoria en disco, utilizada para almacenar toda la información referente a los puntos de comprobación, y el tiempo empleado en almacenar los datos y realizar las comparaciones entre estados. Además dicha limitación es más importante cuanto mayor es el tamaño y complejidad del circuito bajo estudio.

3.3.2 Emulación

Las técnicas para la evaluación de la tolerancia a fallos de un circuito basadas en emulación consisten en realizar la inyección de fallos sobre un prototipo del circuito que se implementa en un dispositivo lógico reprogramable.

Al igual que los métodos de inyección basados en simulación de descripciones HDL, la emulación proporciona información sobre el funcionamiento del circuito en presencia de fallos y permite evaluar, durante el ciclo de diseño, el comportamiento de los mecanismos de tolerancia a fallos integrados en el circuito. Sin embargo, en las técnicas basadas en emulación, los fallos se insertan sobre una plataforma *hardware* de emulación, una FPGA, lo que permite realizar la campaña de inyección a velocidad *hardware* y paralelizar tareas de inyección. En la Figura 23 se representa el esquema general de un entorno de inyección de fallos basado en emulación. El circuito bajo estudio se prototipa en el emulador *hardware* mientras que el control de inyección y el resto de tareas relacionadas con el proceso de evaluación, como la aplicación de los

estímulos de entrada o el análisis de resultados, se ejecutan desde el PC. Como ya se ha indicado, este esquema es general, y puede diferir entre las distintas técnicas de emulación existentes. Las posibles particularidades propuestas por diferentes autores se detallan en el apartado 3.3.2.1.

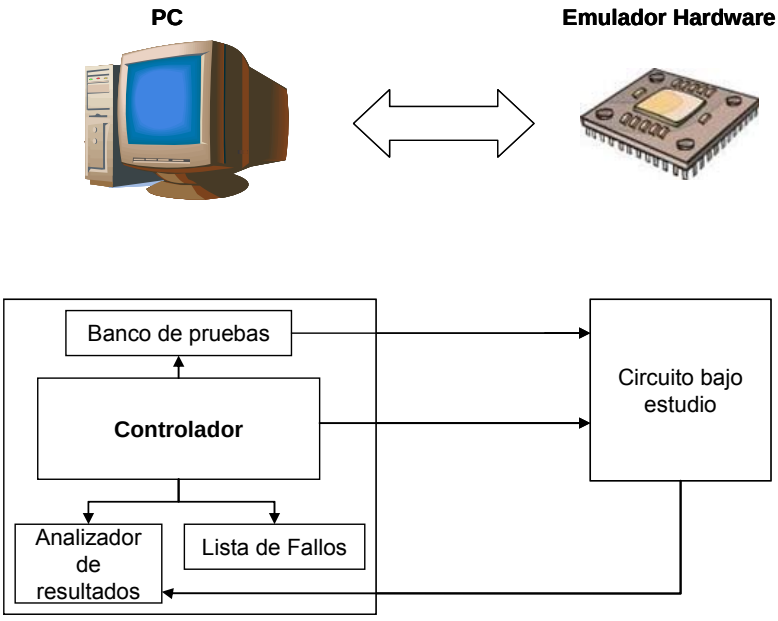


Figura 23. Entorno de inyección de fallos basado en emulación

En general, el PC se encarga de aplicar sobre el circuito los estímulos correspondientes a cada ciclo de ejecución, de controlar el proceso de inyección activando la inserción del fallo en el instante adecuado en función de la lista de fallos que se desean evaluar, y de observar el comportamiento del circuito tras el fallo para analizar y clasificar los fallos. Mientras que la FPGA se encarga de emular la ejecución del circuito bajo estudio.

La emulación permite disminuir notablemente el tiempo empleado en la evaluación de la tolerancia a fallos de un circuito con respecto a las técnicas de simulación, superando el principal inconveniente que plantean estas últimas. Además, las técnicas basadas en emulación con FPGAs permiten al diseñador comprobar el comportamiento real en el entorno de aplicación, considerando las interacciones con otros dispositivos en la fase de diseño.

3.3.2.1 Técnicas y herramientas existentes basadas en emulación con FPGAs

Las técnicas de inyección existentes basadas en emulación parten de una descripción del circuito a estudiar, generalmente escrita en un lenguaje de alto nivel (VHDL o Verilog), que se prototipa en una FPGA. La inyección de fallos se realiza mediante la modificación del dato dónde se desea insertar el fallo. En función de cómo

se implementa dicha modificación, es decir, en función del mecanismo de inyección utilizado, las técnicas de emulación de fallos existentes pueden clasificarse en dos categorías distintas:

- Inyección mediante reconfiguración de la FPGA, modificando el valor del contenido del elemento de memoria a evaluar cada vez que se inyecta un fallo Figura 24.
- Inyección controlada por lógica adicional insertada en el circuito original. Es necesario, por tanto, modificar o *instrumentar*[11] la descripción original.

Las técnicas basadas en reconfiguración no implican un incremento del área utilizada ni la modificación de la descripción original del circuito, pero son más lentas que las basadas en la inserción de lógica adicional puesto que requieren la reconfiguración parcial o total de la FPGA para cada fallo. Ambas clases de técnicas presentan ventajas e inconvenientes que se detallan a lo largo de este apartado. A continuación se describen algunas de las técnicas de emulación de fallos más representativas.

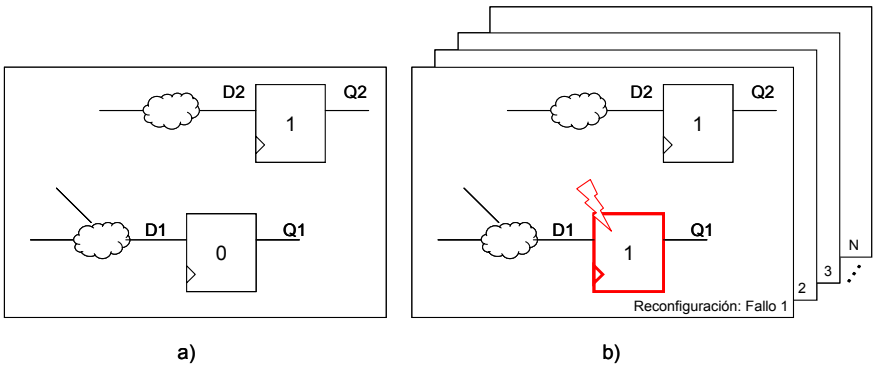
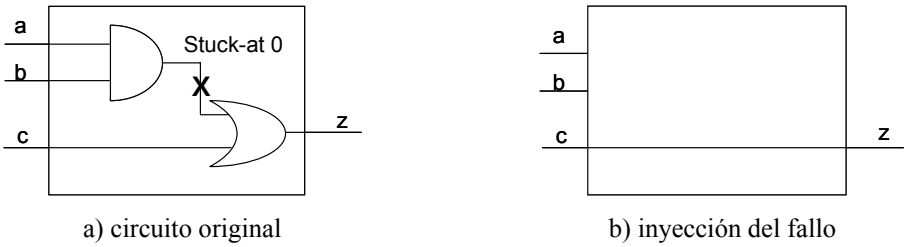


Figura 24. Reconfiguración del circuito para emulación de fallos en *hardware*. a) Circuito original. b) Inserción de un *bit-flip* mediante la reconfiguración del circuito

Inicialmente la emulación de fallos basada en FPGAs se utilizó en el ámbito del test de fabricación, inyectando fallos *stuck-at* en circuitos digitales, con el objetivo de acelerar la generación de patrones de test [Chen95][Hong96]. En [Chen95] se propone el uso de las FPGAs para emular fallos mediante la reconfiguración de la FPGA. En este caso la inyección de un fallo consiste en modificar el circuito conectando una señal a un valor lógico constante, lo que simula un fallo *stuck-at* (Figura 25). Por lo tanto, es necesario sintetizar y reconfigurar un gran número de veces. Con el objetivo de reducir ese número se aplican técnicas que permiten inyectar varios fallos a la vez aunque el tiempo empleado en las distintas reconfiguraciones continua siendo mayor que el tiempo del proceso de emulación. Con este método se consigue un incremento en la

[11] La tarea de modificación de la descripción del circuito, para añadir lógica encargada de controlar la inyección de fallos, se conoce en la literatura como instrumentación del circuito [Cive01a]

velocidad del proceso de inyección de fallos de cerca del doble con respecto a la simulación.



a) circuito original b) inyección del fallo

Figura 25. Inyección de fallos *stuck-at* mediante reconfiguración en una FPGA [Chen95]

[Hong96] presenta una técnica de inyección de fallos también en el ámbito del test de fabricación, pero en este caso la inyección se realiza mediante lógica adicional, para evitar la limitación en la ejecución que supone realizar una reconfiguración de la FPGA por fallo. En este caso se inyectan fallos *stuck-at* añadiendo un bloque *hardware* que se muestra en la Figura 26. Estos bloques contienen biestables que almacenan la señal encargada de controlar la inyección de los fallos. Dichos biestables se conectan en una cadena, de forma que para inyectar un nuevo fallo tan sólo es necesario desplazar el su contenido. De esta forma es posible emular tanto el circuito original como versiones del circuito con fallos, sin reconfigurar.

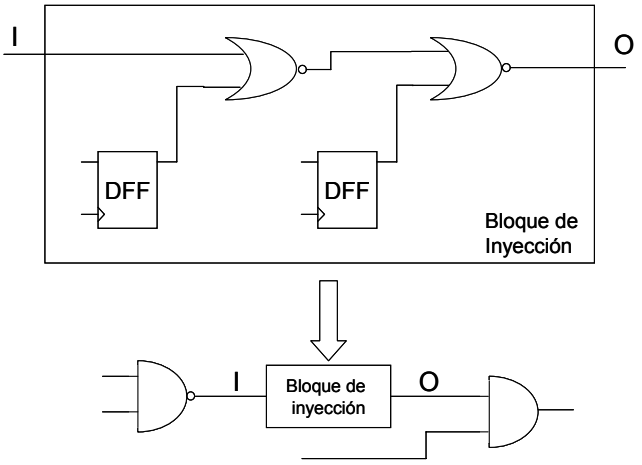


Figura 26. Elemento de inyección de fallos *stuck-at* descrito en [Hong96]

Posteriormente, la emulación de fallos se extendió a la evaluación de la tolerancia a fallos para acelerar el proceso de inyección. [Leve00] presenta una de las primeras propuestas para la aplicación de la emulación de fallos como método de evaluación de la tolerancia frente a SEUs en circuitos digitales, aunque no se llega a implementar en una FPGA. La técnica descrita en [Leve00] es una extensión de los métodos de simulación basados en el uso de *mutantes*, descritos anteriormente (apartado 3.3.1.1.1). Esta solución está orientada a la inserción de fallos en máquinas de estados

partiendo de una descripción funcional y sintetizable del circuito en VHDL. Se propone insertar un SEU en una máquina de estados como una transición errónea entre estados. Para ello se introduce un mutante que consiste en modificar la sentencia de asignación de estados.

En [Cive01a] se presenta una técnica de emulación basada en la instrumentación del circuito, es decir, en la adición de *hardware* adicional encargado de insertar fallos en los biestables del circuito. Los autores defienden la inyección de fallos mediante la instrumentación del circuito frente a la reconfiguración porque acelera el proceso de inyección, al evitar una reconfiguración (parcial o total) por fallo. En concreto, la propuesta consiste en modificar la descripción del circuito a nivel lógico, sustituyendo los biestables del circuito por otros bloques, capaces de inyectar fallos transitorios de tipo *bit-flip*. En la Figura 27 se muestra el esquema del biestable modificado, o instrumento, que se propone en [Cive01a].

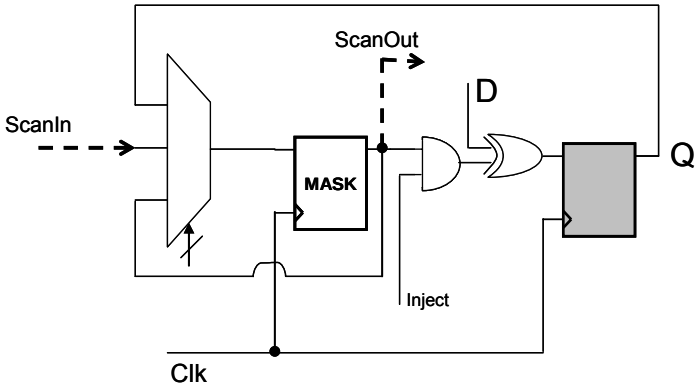


Figura 27. Elemento de inyección de fallos transitorios descrito en [Cive01a]

El elemento de inyección en [Cive01a] consta de un biestable adicional, cuyo contenido determina si el biestable asociado del circuito se ve afectado por un fallo o no cuando se activa la inyección mediante ciertas señales de control (*Inject*). Los biestables de máscara de cada instrumento se conectan en serie formando la cadena de máscara de fallo. Dicha máscara se carga al inicio de la inyección de un fallo como un registro de desplazamiento, para permitir la inyección tanto de fallos simples como múltiples. Además, mediante la cadena de máscara es posible acceder al contenido de todos los biestables y leer el estado del circuito vía serie. Esta capacidad proporciona una observabilidad de los efectos que el fallo provoca en el circuito, más eficiente que las técnicas basadas en reconfiguración, en especial para la detección de los fallos latentes. Con esta solución se consiguen aumentos en la velocidad del proceso de evaluación de cuatro órdenes de magnitud con respecto a las técnicas de inyección basadas en simulación de descripciones VHDL. Esta técnica proporciona tasas de 100 µs/fallo para una frecuencia de reloj de la FPGA de 20 MHz y un banco de pruebas corto (100 vectores). El tiempo empleado en el proceso de inyección aumenta significativamente al

aumentar la longitud del banco de pruebas (~1ms/fallo para un banco de pruebas de 100.000 vectores). A pesar de la mejora conseguida, la velocidad del proceso de inyección está limitada por el tiempo de comunicación entre el PC y la FPGA, y por lo tanto depende del protocolo de comunicación utilizado. [Cive01b] presenta una extensión de la técnica de inyección anterior insertando fallos también en bloques de memoria. Sin embargo, no se propone ninguna solución para observar la evolución de los errores en la memoria.

También en [Lima01a] se propone un sistema de inyección basado en la instrumentación del circuito. Sin embargo, en este caso se propone una estructura novedosa del entorno de inyección, trasladando algunas de las tareas de inyección que tradicionalmente se ejecutaban en el PC a la FPGA (Figura 28).

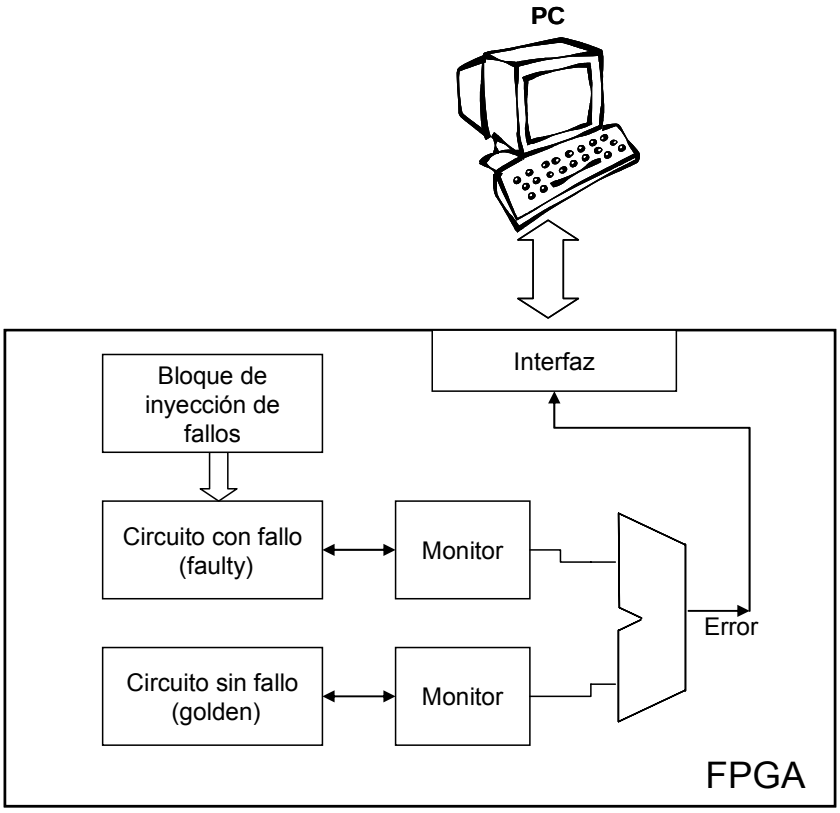


Figura 28. Entorno de emulación de fallos propuesto en [Lima01a]

La solución descrita en [Lima01a] está orientada a la evaluación de procesadores, inyectando fallos tanto en biestables como en memorias. El componente *hardware* del entorno de inyección propuesto consta de dos instancias del circuito, para realizar la ejecución con fallo (*faulty*) y sin fallo (*golden*) en paralelo, un bloque de observación encargado de monitorizar la salida de ambos circuitos para poder detectar averías y un bloque de inyección encargado de generar los parámetros que caracterizan cada fallo (instante de inyección y localización). Se asume que la salida del circuito se almacena en la memoria interna, de forma que, para clasificar un fallo, el bloque de

observación (monitor) lee el contenido de cada palabra de memoria al finalizar la ejecución de la aplicación, comparando los datos *faulty* y *golden*. Por lo tanto, la inyección de cada fallo requiere la ejecución completa de la aplicación.

En los registros, la inyección de fallos presentada en [Lima01a] se realiza de forma similar a la presentada en [Cive01a], sustituyendo cada biestable por un instrumento que permite la inserción de un *bit-flip* mediante señales de control, sin embargo, y a diferencia de [Cive01a], en este caso dichas señales se generan en la FPGA, por el bloque de inyección. Para inyectar en los bloques de memoria interna del circuito, se propone sustituir cada módulo de memoria por un bloque de memoria de doble puerto, en la que uno de los puertos se utiliza para la ejecución normal y el otro para la inyección de fallos. La inyección de un fallo en memoria requiere de tres pasos, leer el dato almacenado, invertir el bit en el que se quiere insertar el fallo y escribir el dato modificado en la dirección indicada. Las memorias deben inicializarse antes de realizar un experimento de inyección. El proceso de inicialización de las memorias y el de la lectura de datos requiere un coste temporal, que depende del tamaño de la memoria y puede suponer un porcentaje considerable con respecto al tiempo total empleado en el proceso de evaluación.

En [Anto02] se presenta una técnica basada en reconfiguración parcial de la FPGA para realizar la inyección de fallos *bit-flip* en los biestables del circuito. En esta propuesta la inserción de un fallo consiste en modificar directamente la cadena de bits que configuran la FPGA, denominada *bitstream*. Para ello se utiliza JBits, que es una aplicación basada en Java y desarrollada por Xilinx®, para permitir a los diseñadores modificar el *bitstream* de sus FPGAs. El contenido del biestable en el que se quiere insertar el fallo se modifica mediante las señales de inicialización asíncrona (*set, reset*), invirtiendo el contenido del biestable en el instante de inyección (Figura 29). Para cada fallo a inyectar se inicia la ejecución de la aplicación hasta el momento de inyección, entonces se lee el estado de los biestables, se activa la señal de inicialización necesaria para invertir el contenido actual y se continúa con la ejecución de la aplicación.

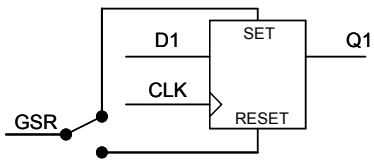


Figura 29. Inyección de *bit-flip* en un biestable mediante el uso de las señales de inicialización asíncrona

La clasificación de los fallos en [Anto02] se realiza leyendo el contenido de los biestables periódicamente y comparándolo con el contenido esperado que fue almacenado durante la ejecución sin fallo. El proceso de lectura de la lógica programada en un dispositivo FPGA desde el PC se denomina *readback* y es posible únicamente en

FPGAs de Xilinx®. Mediante *readback* se puede acceder a los estados almacenados en los elementos de memoria de la FPGA [Xilinx].

Los resultados presentados obtenidos con el método anterior [Anto02], muestran que el tiempo de reconfiguración y el alto número de reconfiguraciones necesarias para la inyección de fallos retardan el proceso de emulación. Esta técnica obtiene unas tasas de inyección de entre 100 ms/fallo y 3,5 s/fallo, en función de la FPGA utilizada.

Otra técnica de inyección de fallos SEU basada en reconfiguración parcial de la FPGA ha sido propuesta más recientemente por la Universidad de Sevilla [Agui04][Lope07b]. En [Agui04] se presenta una herramienta denominada FT_UNSHADES, utilizada por la ESA, que ha sido desarrollada para una FPGA Virtex-II de Xilinx®, explotando la capacidad de reconfiguración parcial y de *readback* de estos dispositivos. Con dicha herramienta se insertan *bit-flips* en cualquier biestable del circuito bajo test, en cualquier instante de tiempo, mediante el mecanismo de inyección descrito en [Anto02]. Como en el caso de la solución propuesta en [Lima01a], en este caso el entorno de inyección descrito difiere del entorno de emulación general, trasladando algunas de las tareas de inyección del PC a la FPGA. El esquema del sistema propuesto se presenta en la Figura 30.

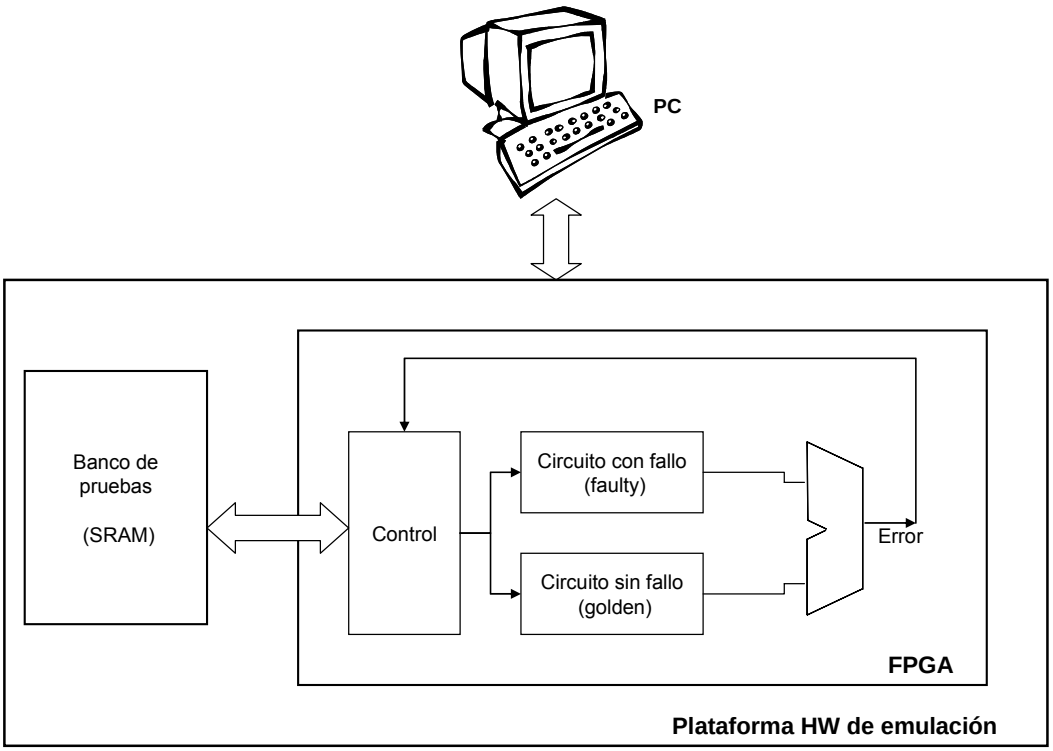


Figura 30. Esquema del entorno de inyección propuesto en [Agui04]

Dos instancias del circuito se ejecutan en paralelo en la FPGA, con y sin fallo, para poder comparar las salidas en cada ciclo de reloj, y así detectar las posibles averías que se produzcan como consecuencia de la inyección de un fallo. Cuando se detecta una

avería, el bloque de control implementado en la FPGA finaliza la emulación del fallo correspondiente, por lo que en ese caso no es necesario ejecutar hasta el final la aplicación. En caso contrario, la emulación del fallo se detiene al finalizar la ejecución del banco de pruebas, entonces se compara el estado del circuito con el esperado (resultado de la ejecución sin fallo) para clasificar los fallos latentes. Los estímulos de entrada se almacenan en una memoria SRAM incluida en la plataforma *hardware* de emulación, para disminuir la comunicación necesaria con el PC y acelerar el proceso de inyección. Una herramienta *software* se encarga de generar los parámetros que caracterizan cada fallo (instante de inyección y localización) y de almacenar la clasificación de los fallos. Esta solución proporciona tasas de inyección del orden de 100ms/fallo [Lope07b].

Recientemente, en [Ejla05] se propone una modificación del entorno de inyección propuesto en [Cive01a], para mejorar la observabilidad del comportamiento del circuito de forma rápida, reduciendo la comunicación con el PC. La solución propuesta consiste en realizar la clasificación de los fallos en la FPGA, introduciendo para ello el módulo de observación representado en la Figura 31. Los biestables del circuito se replican, compartiéndose la lógica combinacional, un biestable almacena los valores de la ejecución con fallos (*faulty*) y otro se utiliza durante la ejecución sin fallos (*golden*). Ambas ejecuciones se alternan en el tiempo comparándose continuamente el contenido de ambos biestables. Cada vez que se detecta una diferencia entre un registro *faulty* y el correspondiente registro de referencia *golden*, se interrumpe al PC el cual almacena qué biestables contienen un error. De esta forma se observa la propagación del fallo a través del circuito.

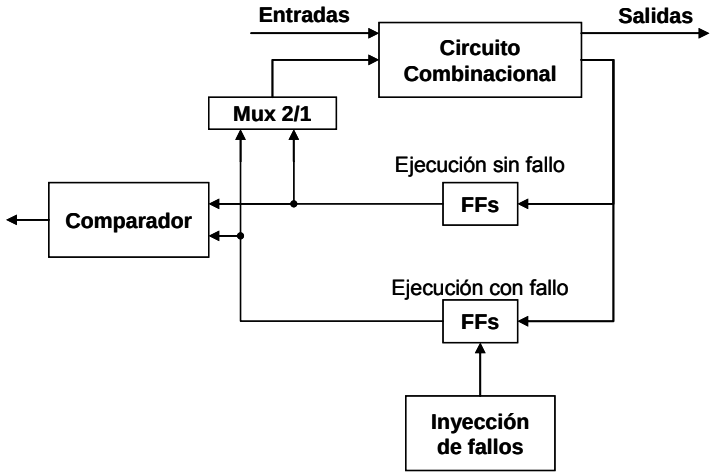


Figura 31. Modulo de observación propuesto en [Ejla05] para acelerar el proceso de emulación de fallos

Todas las técnicas de emulación de fallos descritas obtienen una mejora en la velocidad del proceso de evaluación con respecto a la simulación. La aceleración conseguida en los trabajos descritos (con respecto a la simulación) es de entre uno y

cuatro órdenes de magnitud. En definitiva, la comunidad científica acepta, tras los trabajos desarrollados por diferentes autores, que las técnicas de inyección de fallos basadas en emulación con FPGA proporcionan un aumento de la velocidad del proceso de evaluación con respecto a la simulación. Esta mejora en la velocidad hace de las técnicas de emulación una solución efectiva para la evaluación de la tolerancia a fallos de los CIs, frente a la simulación.

Los resultados obtenidos en los distintos trabajos realizados hasta el momento muestran que las técnicas de inyección basadas en la instrumentación del circuito son más eficaces en cuestión de velocidad que las técnicas basadas en reconfiguración, aunque requieren la modificación del circuito y la adición de lógica extra, esto es, mayores requisitos en área. Debido al aumento de la densidad de integración en los CIs es posible fabricar FPGAs con millones de puertas, incrementándose la capacidad de procesado de las FPGAs. Esto ha propiciado el avance en el estudio de los métodos basados en emulación, siendo ahora suficiente una placa FPGA para circuitos relativamente grandes. En la actualidad, existen en el mercado plataformas de desarrollo FPGA con la posibilidad de conectarse directamente al computador a través del bus PCI lo que facilita y agiliza la transferencia de información entre el PC y la placa emuladora en comparación con otros sistemas de comunicación. Sin embargo, y a pesar de los protocolos de comunicación existentes, las técnicas basadas en emulación propuestas en la literatura por otros autores requieren una intensa comunicación entre el PC y la FPGA, lo que limita la velocidad del proceso de inyección al protocolo de comunicación, siempre más lento que el sistema de emulación en la FPGA [Cive01a].

En este trabajo de tesis se presenta un entorno de emulación que minimiza la comunicación necesaria entre PC y FPGA, para lo cual se propone una nueva arquitectura del sistema de emulación [Port04][Garc04b][Lope05b][Lope05c] [Lope07a][Garc04b]. En esta misma dirección, algunas de las soluciones más recientes presentes en la literatura [Agui04][Ejla05][Lima01a], descritas anteriormente, proponen técnicas para disminuir la comunicación PC-FPGA, lo que muestra el interés de la comunidad científica por acelerar el proceso de inyección.

3.4 Técnicas de inyección de fallos sobre un componente comercial

La inyección de fallos sobre un componente comercial se aplica para analizar la confiabilidad de diseños finales, para medir el comportamiento de las técnicas de mitigación de fallos frente a fallos reales o cuando no se dispone de una descripción del circuito como sucede con los diseños IP (*Intellectual Property*). Este método proporciona resultados más realistas que en el caso de aplicar la inyección sobre una descripción del circuito y es, en algunos casos, el único método que permite medir con

precisión parámetros como la latencia o la cobertura de fallos. Estas técnicas se aplican sobre el circuito mientras esté funciona en modo normal, por lo que son muy apropiadas para evaluar aplicaciones en tiempo real. A continuación se describen las diferentes técnicas existentes y las propuestas que han realizado diferentes autores.

3.4.1 Inserción de fallos físicos

La inyección de fallos reales consiste en someter el circuito bajo estudio a una perturbación externa que produzca fallos, sin necesidad de modificar el comportamiento del circuito a excepción de la propia inserción del fallo. Son por lo tanto métodos no intrusivos que no introducen ningún aumento en el tiempo de ejecución con respecto a la duración normal de la ejecución del circuito. Como mayores inconvenientes, presentan una observabilidad y controlabilidad limitadas y la necesidad de un equipo de test adicional para realizar la inyección y observar el resultado. Dentro de estas técnicas se pueden distinguir distintos tipos, inyección mediante radiación de iones pesados, mediante láser o técnicas de inyección en los conectores (conocidas en la literatura como técnicas de inyección a nivel de pin).

3.4.1.1 Radiación con iones pesados

La principal fuente de fallos SEU son las partículas que conforman la radiación ionizante proveniente del espacio. Por lo tanto, la forma más precisa y realista de probar experimentalmente el comportamiento del circuito bajo fallos es someterlo a dicha radiación. Estos experimentos, denominados test de radiación, consisten en bombardear el circuito de estudio con un haz de partículas, normalmente iones pesados, pero también pueden ser protones o neutrones. Para realizar un test de radiación, es necesario que el dispositivo haya sido desprovisto de su encapsulado, y que el test se realice en vacío para evitar colisiones del haz de iones con otras partículas, como las del aire, antes de alcanzar el dispositivo a radiar [Karl94]. El comportamiento del circuito bajo radiación se compara con un circuito libre de fallos para detectar los efectos de los fallos. Los fallos transitorios, que se producen como consecuencia de radiar un CI, pueden afectar a cualquier elemento del circuito, aunque la posición afectada y el instante de tiempo en el que se genera el fallo son impredecibles y no se pueden controlar.

El principal objetivo del test de radiación es medir los parámetros de confiabilidad de la aplicación real, considerando el conjunto de estímulos que se darán en la operación normal. El test de radiación puede aplicarse sobre el circuito en modo estático o en modo dinámico si éste está realizando una actividad determinada [Vela00]. Sin embargo, la aplicación final suele no estar disponible en el momento del test o puede ser modificada posteriormente. La metodología habitual consiste en realizar un test estático con radiación de partículas y realizar el estudio dinámico mediante otra

técnica de inyección [Vela00]. Los datos obtenidos permiten medir la sección eficaz de un circuito, σ, frente a SEUs, proporcionando información sobre la sensibilidad de dicho dispositivo a ese tipo de fallos. Además, conociendo el flujo de partículas del haz, es posible predecir la tasa de fallos SEU que se producirá cuando el circuito esté funcionando normalmente.

Este método se utiliza también para comparar la sensibilidad de distintas tecnologías y estudiar los factores que pueden influir en la tasa de fallos, como por ejemplo la frecuencia de funcionamiento. En [Irom04] se realizan experimentos de radiación para analizar la dependencia de un circuito (un microprocesador PowerPC) con el aumento de la frecuencia de reloj y cómo dicho aumento puede afectar a la sensibilidad del circuito frente a SEUs. Los resultados muestran que la sección eficaz de los registros crece cuando aumenta la frecuencia de reloj. Este dato permite hacer una previsión de cómo evoluciona la sensibilidad de los circuitos, cada vez más rápidos gracias al desarrollo de la tecnología. Debido a la precisión y fiabilidad del método, el test de radiación proporciona información de referencia para validar otras técnicas de inyección de fallos.

Los experimentos de radiación precisan de infraestructuras y equipos especiales, muy costosos, que sólo están disponibles en centros de aceleradores de partículas. Además, requieren *hardware* dedicado para realizar los experimentos. Algunos autores han desarrollado plataformas *hardware* para la implementación de experimentos de radiación [Faur02][Karl94]. Estos sistemas deben proporcionar soporte para realizar la comparación de las salidas del circuito, la aplicación de los estímulos de entrada, la comunicación con el exterior, etc. Por otra parte, el test de radiación, al igual que otros métodos de inyección de fallos físicos, puede provocar un fallo que dañe el circuito. Por ejemplo, un problema a considerar cuando se realiza un test de radiación es el efecto de *latch-up*, SEL. Este efecto, descrito en el apartado 2.2.1, da lugar a grandes corrientes y a una excesiva disipación de calor que puede dañar el circuito. Puesto que los fallos generados mediante radiación no son controlables, hay que incluir un mecanismo de protección frente a un posible SEL. Este mecanismo consiste en monitorizar la corriente del circuito para detectar cualquier incremento. En caso de observar un valor de corriente superior a cierto umbral el dispositivo debe apagarse.

En definitiva, la radiación de un circuito con iones pesados es un método de inyección de fallos que proporciona resultados precisos de la sensibilidad de un circuito, considerando tanto la tecnología como las estructuras de tolerancia a fallos incluidas. Sin embargo, un posible rediseño implica la modificación del diseño y la fabricación de un nuevo prototipo, lo que implica un incremento del tiempo y coste del proceso de diseño. Esto, junto con el alto coste de los equipos necesarios para la realización de un test de fabricación, y la baja capacidad de análisis que proporciona esta técnica hacen

que sea necesario aplicar otras técnicas de inyección durante la etapa de diseño, de forma previa.

3.4.1.2 Inyección láser

La inyección de fallos transitorios mediante un haz láser consiste en bombardear el circuito con dicho haz, de forma que la energía de la radiación láser genera pares electrón-hueco, dando lugar a fallos transitorios. En primer lugar, la inyección láser se aplicó para generar fallos permanentes [Vela92_a], pero actualmente es una técnica muy utilizada para la inserción de fallos transitorios SEEs.

La inyección láser presenta ciertas ventajas frente a la radiación con iones:

- La inyección láser, a diferencia de la radiación con iones, permite seleccionar, mediante técnicas de microscopía, el elemento del circuito en el que se inserta el fallo. La elección de la localización en las últimas técnicas propuestas es muy exacta. Para localizar la zona es necesario utilizar una herramienta CAD de *layout* y disponer del diseño físico del circuito a evaluar. A continuación, la localización se traslada a coordenadas X-Y de la tabla láser [Poug04] utilizando un microscopio para focalizar el haz en la zona elegida. Entonces, se provoca un pulso corto de suficiente potencia para inducir un fallo transitorio, pero sin dañar el componente.
- Los equipos necesarios para la realización de un experimento de inyección con radiación son caros y sólo están disponibles en centros de aceleradores, mientras que las instalaciones y el equipamiento necesarios para realizar un experimento de inyección con láser son más accesibles.

Estas características de la inyección de fallos mediante láser han llevado a algunos autores a estudiar la posibilidad de utilizar el láser como método alternativo al test de radiación. En [Mill04] se presenta un estudio comparativo entre ambas técnicas. Una de las principales diferencias encontradas entre ambas técnicas es el ancho del haz [Mill04], mayor en el caso del haz láser que en el haz de partículas ionizantes. Esto puede generar que un pulso láser afecte a varias zonas a la vez si el ancho del haz es mayor que la tecnología estudiada.

En resumen, la inyección láser es un método efectivo para investigar los SEEs en circuitos digitales. Sin embargo, el proceso de inyección es más complejo que en el caso de la inyección mediante radiación puesto que necesita el uso de herramientas para el control tanto del instante de inyección como de la localización del fallo.

3.4.1.3 Inyección a nivel de pin

La inyección de fallos a nivel de pin consiste en introducir una perturbación en el CI a través de sus pines. Se usa para insertar fallos permanentes [Powe95] [Prad96].

Esta técnica de inyección ofrece la capacidad de inyectar una gran cantidad de fallos automáticamente en un corto período de tiempo. Es una técnica genérica, de fácil aplicación a distintos circuitos, aunque el acceso a los pines debe estudiarse y solucionarse para cada circuito particular. Los circuitos diseñados para ser testados, de forma que tienen todas sus entradas y salidas conectadas a pines externos, se evalúan fácilmente a través de los pines, en caso contrario, el acceso a las zonas del circuito dónde inyectar fallos es un limitación importante que debe resolverse en cada caso particular [Bens03]. Esta técnica presenta los siguientes inconvenientes:

- La asignación forzada de un valor en un pin puede dañar el circuito.
- La mayor limitación de esta técnica es que las localizaciones del fallo están limitadas a las señales de entrada y salida conectadas a los pines del circuito.
- Además la observabilidad también está muy restringida por lo que no es posible observar cómo la modificación de la señal en un pin afecta al resto del circuito para detectar las zonas más críticas de éste.
- No es posible utilizar esta técnica cuando el objetivo principal es la evaluación del circuito frente a SEUs, porque los elementos de memoria del diseño son partes internas.

Así pues, la inyección de fallos a través de los pines del circuito, era una técnica muy utilizada para evaluar la tolerancia a fallos en la década pasada, pero actualmente debido a las limitaciones que presenta y al aumento de la densidad de integración y de las frecuencias de funcionamiento son más habituales la radiación con iones pesados o la inyección láser como métodos de inyección de fallos físicos.

Un ejemplo de las herramientas desarrolladas para implementar este tipo de solución es MESSALINE [Arla90], que permite inyectar distintos tipos de fallos y controlar la duración y frecuencia de los mismos.

3.4.1.4 Otras técnicas

En [Varg05] se propone una técnica de inyección basada en interferencias electromagnéticas (EMI, *ElectroMagnetic Interferences*). En este caso se asume que los fallos generados son *bit-flip* y fallos de retardo (*delay*). Esta técnica no permite controlar la posición y el instante de tiempo en el que se inyecta el fallo. No existen muchos estudios sobre el uso de EMI para la evaluación de la tolerancia a fallos SEU en circuitos digitales. Esta técnica todavía está en desarrollo y por lo tanto hoy en día no se usa comúnmente.

Una variante de la inyección a nivel de pin es la inyección mediante la alteración de la tensión de alimentación. En [Bakh05] se utilizan las perturbaciones en la tensión

de alimentación para evaluar el efecto de fallos transitorios en FPGAs de SRAM. Estas perturbaciones se introducen a través de un transistor MOS colocado entre la tensión de alimentación y las líneas correspondientes en la FPGA y un transistor de potencia entre la masa y las líneas de alimentación. Cuando el transistor MOS no conduce, la capacidad de la FPGA se descarga a través del transistor de potencia y se genera una caída de la tensión de unos pocos microsegundos de duración.

3.4.2 Inserción de fallos lógicos

La inyección física de fallos implica el uso de equipos especiales para la generación de los fallos, caros, especialmente para las técnicas de radiación con iones. En general, son técnicas con limitaciones en el control de las posiciones dónde se inyectan los fallos y en la observabilidad de sus efectos. Un método de superar estas limitaciones cuando la inyección debe hacerse en un componente comercial consiste en realizar la inyección de fallos lógicos aprovechando los recursos propios del sistema a estudiar, como por ejemplo:

- La lógica presente en los circuitos para realizar tareas de test, disponible en muchos de los circuitos VLSI actuales.
- Funciones *software* en el caso de circuitos o sistemas microprocesadores.

En algunos sistemas como las FPGA y los microprocesadores que tienen estructuras y funciones propias que los caracterizan, se han propuesto métodos de inyección específicos que aprovechan dichos mecanismos para realizar la inyección. Esas técnicas se describen en los siguientes apartados.

3.4.2.1 Inyección de fallos en FPGAs

La estructura interna de las FPGAs causa efectos particulares en estos dispositivos en presencia de SEEs, como se vio en el capítulo 2. Los SEEs en FPGAs deben evaluarse cuando la implementación final del diseño es precisamente una FPGA. Los fallos SEU en dispositivos FPGA afectan a los biestables, a los bloques de memoria interna y a los bits de la memoria de configuración que controlan la lógica implementada y las interconexiones. Por lo tanto, para evaluar sus propiedades de confiabilidad es necesario inyectar fallos en todas estas posiciones. Un SEU en un bit de configuración puede cambiar la lógica implementada. En la Figura 32 se presenta un ejemplo de cómo se modifica la lógica del circuito de forma permanente cuando un SEU afecta a una LUT (*Look-Up Table*). Una LUT de *n*-bits se utiliza para codificar cualquier función lógica *booleana* de *n* entradas como una tabla de verdad. En función del bit afectado un SEU puede provocar un cambio en la lógica combinacional implementada o un fallo de tipo *stuck-at*.

La inyección de fallos en la memoria de configuración es un problema propio de las FPGAs. La complejidad de los dispositivos reprogramables hace que la inyección de fallos mediante simulación sea un proceso excesivamente lento. A este hecho hay que añadir que normalmente los fabricantes de estos dispositivos no publican los modelos o descripciones del circuito. Por lo tanto, los métodos de evaluación de la tolerancia a fallos de FPGAs comúnmente utilizados están basados en la inserción de fallos en un componente comercial mediante una de las siguientes técnicas:

1. Inserción de fallos físicos aplicando alguna de las soluciones descritas en el apartado 3.4.1.
2. Introduciendo modificaciones, *bit-flip*, en la memoria de configuración de la FPGA a evaluar.

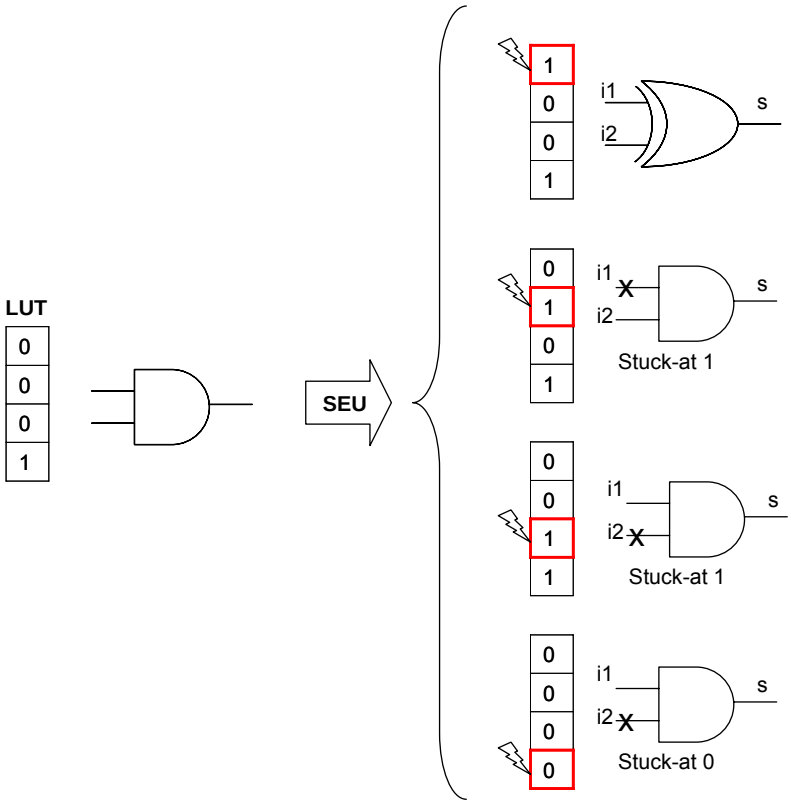


Figura 32. Ejemplo de cómo un SEU afecta a un bit de configuración modificando la lógica implementada

Para modificar un bit de la memoria de configuración es necesario leerla y volverla a escribir con el valor del bit en cuestión invertido mediante reconfiguración en tiempo de ejecución. Los entornos de inyección, desarrollados hasta el momento, dirigidos a las FPGAs se han realizado sobre dispositivos de Xilinx® [Alde03][Bern04][Kent06] porque este fabricante proporciona la capacidad de leer la memoria de configuración o *bitstream* en cualquier momento (*readback*), así como el

software necesario para manipularlo (JBits[12]). En [Kent06] y [Lima01b] se presentan soluciones basadas en reconfiguración mediante JBits para introducir *bit-flips* en los biestables del diseño y en los bits de configuración de las LUTs de FPGAs Virtex®.

El trabajo presentado en [Alde03] propone una herramienta para inyectar fallos SEUs, también en una Virtex®, sin recurrir a ningún *software* comercial para la modificación del *bitstream*. Dicha herramienta es capaz de inyectar fallos en los puntos de interconexión programables (PIP, *Programmable Interconnection Point*, que no son accesibles mediante JBits), y en los registros de control de la configuración de la FPGA. El sistema de inyección se implementa en una FPGA que se conecta, por un lado al PC a través de un puerto USB y por otro a la FPGA bajo estudio.

[Bern04] presenta un sistema de inyección en el que se inyecta únicamente en los recursos de la FPGA que se utilizan. Para determinar cuales son estos recursos se usa una base de datos construida a partir de la decodificación del *bitstream* de la FPGA. Cada familia de FPGAs puede tener una estructura de *bitstream* diferente por lo que el uso se esta técnica se restringe a la familia para la que se ha estudiado la estructura de la memoria de configuración. En [Bern04] el método se aplica sobre una FPGA de Xilinx®.

3.4.2.2 Inyección de fallos en microprocesadores

Los microprocesadores son componentes muy comunes en los sistemas digitales actuales. Los fallos SEU en estos sistemas pueden afectar a los registros internos y a las memorias implementadas en SRAM, normalmente utilizadas para almacenar datos porque el código suele almacenarse en otro tipo de memoria, como memoria *flash*, menos sensible a los efectos de SEEs. Cuando no se dispone de una descripción del circuito, es necesario realizar la inyección de fallos en el componente comercial, que es el caso más habitual. La inyección de fallos en los recursos internos de un microprocesador es una tarea complicada, debido a la falta de controlabilidad y observabilidad sobre dichos recursos y a la limitada accesibilidad. Existen propuestas, descritas más adelante en este mismo apartado, que proponen hacer uso de los recursos *hardware* disponibles en el propio microprocesador para realizar la inyección de fallos. Por ejemplo, [Ferr01] propone inyectar fallos utilizando acceso directo a memoria (DMA). La mayoría de las técnicas propuestas por los investigadores se basan en mecanismos de inyección *software* o en usar las capacidades de depuración incluidas en los procesadores actuales.

[12] Como se dijo en el apartado 3.3.2.1, JBits es un conjunto de clases en el lenguaje de programación Java que proporcionan una interfaz (API, *Application Programming Interface*) para el acceso al *bitstream* de las FPGAs de las familias Virtex® y Virtex-II® de Xilinx®.

3.4.2.2.1 Técnicas software

Las técnicas de inyección de fallos SEU en microprocesadores basadas en *software* consisten en introducir una rutina *software* para modificar el valor del elemento de memoria en el que se desea inyectar un SEU. Muchos fallos físicos, como por ejemplo errores en buses o memorias, afectan a la ejecución del *software*, por lo que se pueden modelar fallos físicos modificando el código. Así pues las técnicas de inyección basadas en *software* permiten emular, tanto fallos *hardware*, como fallos *software*. Estas técnicas son rápidas porque se ejecutan a la velocidad normal del procesador y no requieren de *hardware* adicional para realizar la inyección. Sin embargo, presentan ciertas limitaciones. Sólo se pueden inyectar fallos en posiciones accesibles por *software*. Son técnicas intrusivas porque conllevan una modificación del código a ejecutar por el microprocesador y son difíciles de generalizar a diferentes sistemas debido a que los métodos se desarrollan generalmente en base a las características y las capacidades del sistema al que se aplica.

La inserción de un fallo puede implementarse en tiempo de compilación o durante la ejecución de la aplicación [Hsue97]. Las técnicas de inyección en tiempo de compilación requieren la modificación del programa fuente o el código ensamblador cada vez que se realiza una inyección y son más adecuados para fallos permanentes. Por otro lado, las técnicas de inyección en tiempo de ejecución se utilizan para emular fallos transitorios. Las instrucciones necesarias para emular un fallo transitorio son:

- Leer el contenido del elemento de memoria que se quiere modificar.
- Invertir dicho valor.
- Escribir el dato erróneo de nuevo en el registro o palabra de memoria afectado.

Para activar la inserción del fallo es necesario añadir algún mecanismo. Este mecanismo puede ser:

- Un temporizador. Es la técnica más sencilla. Cuando el temporizador termina la cuenta se genera una interrupción que realiza la inyección del fallo. Este es el mecanismo que implementa la herramienta FERRARI presentada en [Kana95].
- Uso de excepciones. Cuando una excepción *hardware*[13] o *software*[14] se ejecuta, se genera una interrupción que transfiere el control de la ejecución al bloque *software* de inyección encargado de insertar el fallo.

[13] Interrupción asíncrona, como por ejemplo una interrupción generada por puertos de entrada/salida.

[14] Interrupción síncrona que tiene lugar después de ejecutar una instrucción, como por ejemplo ocurre cuando el procesador intenta realizar una división entre cero.

- Inserción de código. Se añaden instrucciones, al programa ejecutado, encargadas de inyectar un fallo antes de que se ejecute cierta instrucción. Estas técnicas no modifican las instrucciones originales como ocurre en las técnicas de inyección en tiempo de compilación sino que añaden nuevas instrucciones.

Los dos primeros mecanismos de activación, temporizador y excepciones, son los más comunes entre las distintas propuestas existentes hasta el momento. En ambos casos la inyección se realiza mediante rutinas de interrupción. Cuando la interrupción se activa, el procesador finaliza la instrucción que estaba siendo ejecutada y se almacena el valor del contador de programa automáticamente. Por lo tanto, la resolución temporal que ofrecen estos métodos no es un ciclo de reloj sino que está limitada a una instrucción. La mayoría de las técnicas *software* propuestas inyecta fallos mediante la activación de una interrupción como las presentadas en [Bens98][Carr98] y [Vela00].

En [Bens98] se presenta una solución que ejecuta la aplicación en modo paso a paso. Tras la ejecución de cada instrucción se activa una excepción. La rutina de interrupción comprueba si se ha alcanzado el instante de inyección en cuyo caso inserta el fallo, en caso contrario se reanuda la ejecución de la siguiente instrucción. Un computador, conectado al procesador bajo estudio, se encarga de almacenar la lista de fallos y de observar los efectos del fallo inyectado, con objeto de minimizar las modificaciones a realizar en el sistema objetivo. Al realizar una ejecución en modo paso a paso se introduce un retardo en la ejecución normal de la aplicación.

La técnica propuesta en [Carr98], denominada Xception, utiliza los mecanismos de excepción en depuración para activar la inyección de un fallo. Los eventos que disparan la excepción en este caso están relacionados con la ejecución de instrucciones, manipulación de datos y características temporales. La inserción del fallo es muy rápida (~μs), por lo que puede aplicarse para evaluar sistemas en tiempo real. Su generalización a otros procesadores no es directa porque las excepciones disponibles para cada procesador son diferentes. También en este caso el control de la inyección, la lista de fallos y el análisis de resultados se realizan desde un computador.

Por otro lado, en [Vela00] se describe una solución basada en rutinas de interrupción activadas asíncronamente denominada *C.E.U.* (*Code Emulated Upset*), donde el control de la inyección se implementa en *hardware*. Es una solución independiente de la aplicación y permite inyectar un alto número de fallos puesto que cada fallo tan sólo requiere unas pocas decenas de ciclos de reloj. La implementación automática se realiza sobre una arquitectura *hardware*, denominada *THESIC* (*Testbed for Harsh Environment Studies on Integrated Circuits*), que se encarga de generar las interrupciones que activan la inyección, las localizaciones del fallo, y de leer los resultados.

3.4.2.2.2 Técnicas basadas en las infraestructuras para depuración de los microprocesadores

La mayoría de los procesadores actuales contienen lógica dedicada para tareas de depuración. Esta lógica, conocida como OCD (*On-Chip Debugger*), permite al diseñador el acceso a los recursos internos a través de herramientas *software*. Estas capacidades se pueden utilizar para acceder a los contenidos de la memoria y los registros, proporcionando un mecanismo sencillo para la inyección de fallos en circuitos microprocesadores comerciales, sin necesidad de modificar el *software* implementado.

[Folk98] presenta la herramienta de inyección FIMBUL (*Fault Injection and Monitoring using BUilt in Logic*). La inserción de fallos se realiza a través del puerto de acceso para test (TAP, *Test Access Port*) compatible con el estándar IEEE 1149.1 para rastreo periférico, o su término en inglés *boundary scan* (utilizado en adelante) [JTAG]. Esta herramienta está orientada a la evaluación de fallos SEU en microprocesadores por lo que el modelo de fallo utilizado es el *bit-flip*. El tiempo de inyección se fija mediante el uso de puntos de ruptura, que se configuran mediante la programación del registro de depuración al que se accede por el TAP. La herramienta FIMBUL está implementada en *software* y se ejecuta desde una estación de trabajo UNIX a 50MHz. La tasa de inyección obtenida con esta herramienta es de 0,25 fallos/s.

En [Reba99] se propone una solución que utiliza el modo de depuración BDM (*Background Debugging Mode*) disponible en los microprocesadores de Motorola (Figura 33). Cuando este modo de operación está activado se puede controlar externamente la unidad del microcontrolador y los accesos a memoria y a dispositivos de entrada/salida a través de una interfaz serie. El entorno de inyección se implementa por completo en *software* y está limitado a procesadores de Motorola.

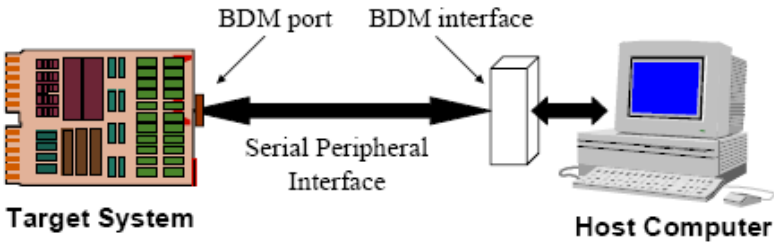


Figura 33. Entorno de inyección propuesto en [Reba99] basado en el uso del modo de operación BDM

Los trabajos presentados en [Fida06][Peng06] proponen soluciones más generales basadas en el estándar de depuración Nexus [Nexus]. El estándar Nexus proporciona un interfaz de propósito general para la depuración en procesadores empotrados y el desarrollo de herramientas de depuración. Las características definidas en el estándar se pueden clasificar en cuatro clases. La clase 4 es el conjunto más completo de propiedades y permite la depuración en tiempo real, siendo las otras clases

un subconjunto de esta. Ambas propuestas se basan en las propiedades de la clase 4, por lo que están orientadas a microprocesadores que soportan un depurador Nexus de altas prestaciones.

[Peng06] propone un entorno de inyección implementado en el PC que activa la inserción de un fallo con un temporizador. La solución propuesta en [Fida06], se basa en modificar la infraestructura de depuración añadiendo un bloque para realizar la inyección de fallo y evitar el tener que detener la ejecución del sistema para insertar el fallo (Figura 34). Esta técnica puede aplicarse a sistemas en tiempo real. La campaña de inyección se controla desde un PC que envía mensajes al OCD a través de un puerto USB o Ethernet.

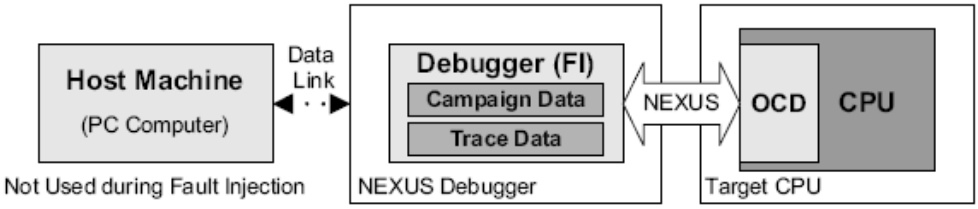


Figura 34. Entorno de inyección de fallos propuesto en [Fida06]

Nótese, que algunas de las propuestas descritas son muy recientes, [Fida06][Peng06], lo que ilustra el interés actual de los investigadores en este tipo de técnicas. Esto se debe, a dos razones fundamentales:

- Por una parte, se debe a la tendencia actual de diseño de SoC y circuitos embebidos, lo que requiere de una técnica de evaluación de la tolerancia a fallos SEU en componentes comerciales efectiva de bajo coste para las primeras medidas de la confiabilidad.
- La evolución y mejora de los microprocesadores actuales conlleva un desarrollo de las capacidades de depuración. La mayoría de los microprocesadores actuales incluyen potentes infraestructuras de depuración que permiten el acceso a los recursos internos, como registros y memorias, las zonas sensibles a fallos SEU.

En definitiva, las técnicas de inyección basadas en el uso de las infraestructuras de depuración integradas en los microprocesadores permiten el acceso a los recursos internos del procesador, lo que posibilita la inyección de fallos y la observación de sus efectos. Además, esta técnica aprovecha los recursos lógicos disponibles y no requiere el uso de plataformas *hardware* adicionales específicas para realizar la inyección. Sin embargo, hay que tener en cuenta que el acceso a los recursos internos está limitado a las capacidades del depurador.

Las técnicas de inyección basadas en OCDs propuestas por otros autores consisten en realizar el control y observación de la inyección desde el PC, lo que retarda

el proceso de inyección con respecto a la velocidad del *hardware*. Una excepción es la propuesta presentada en [Fida06] donde algunas tareas de inyección se realizan desde el OCD modificado para acelerar el proceso de inyección. Sin embargo, esta solución sólo puede aplicarse cuando se dispone de la descripción del OCD y además éste es compatible con el estándar Nexus clase 4.

3.5 Resumen y conclusiones

Las técnicas para la inyección de fallos transitorios en CIs digitales, propuestas por otros autores, presentan diferentes ventajas e inconvenientes. Las propiedades que caracterizan una técnica de inyección son la fiabilidad, el coste económico, la intrusividad, la velocidad, el esfuerzo de implementación y la eficiencia para controlar la inyección y observar sus efectos. Se han realizado numerosos estudios [Arla90][Mill04][Bens99][Folk98][Faur03] en los que se comparan distintas técnicas de inyección. Estos estudios persiguen alguno de los siguientes objetivos:

- Analizar si alguna de las técnicas a comparar es mejor y más adecuada que las demás o por el contrario los resultados que obtienen son complementarios.
- Validar una nueva propuesta de inyección.

Las conclusiones obtenidas muestran que en general la técnica de inyección más adecuada depende de:

- Los objetivos que se quieren cubrir como qué parámetros de confiabilidad se quieren medir o si se persigue la detección de zonas críticas, etc.
- Los recursos disponibles
- Los requisitos principales que debe cubrir el método de evaluación, como por ejemplo la **velocidad** del proceso, la **fiabilidad** de los resultados, el **coste de diseño** o una mínima modificación del circuito a estudiar (**baja intrusividad**).

Las técnicas de inyección se pueden agrupar en dos grandes categorías en función de si se aplican sobre una descripción del circuito o sobre un componente comercial. La inyección en una descripción es muy útil en el proceso de diseño de un circuito tolerante a fallos. Permite detectar las zonas críticas del circuito que requieren de algún mecanismo de endurecimiento y reducir en lo posible el tiempo necesario para la comercialización del circuito, lo que es un objetivo fundamental para los diseñadores. Realizar ese estudio sobre una implementación comercial supondría un incremento considerable en el coste de diseño, ya que una modificación en el circuito requeriría la fabricación de un nuevo prototipo. Otra ventaja que presenta la inyección en descripciones es la flexibilidad de las técnicas, permitiendo el acceso a todos los recursos del circuito. Sin embargo, su precisión depende de la precisión de los modelos

del circuito y del fallo utilizados, mientras que en el caso de la inyección en componentes comerciales la fiabilidad es mayor. Además, en los sistemas actuales es muy habitual el uso de circuitos IP o comerciales de los cuales no se dispone de una descripción del diseño.

Dentro de las diferentes técnicas de inyección de fallos en la descripción del circuito se distinguen las técnicas basadas en simulación y las basadas en emulación con FPGAs. La complejidad de los circuitos actuales hace que las técnicas basadas en simulación requieran un gran coste en términos de tiempo de CPU y de recursos de memoria RAM. Así, la principal desventaja de estos métodos es el tiempo empleado en el proceso de evaluación. Para acelerar este proceso se propone la emulación *hardware* en FPGAs que es una técnica de reciente aplicación en el campo de la evaluación de la tolerancia a fallos. Existen algunas propuestas que ilustran la potencia de este tipo de soluciones. Sin embargo, todas las técnicas de emulación presentadas proponen controlar la campaña de inyección desde el PC lo que implica una comunicación intensa entre la FPGA y el PC que limita la velocidad del proceso.

La inyección de fallos SEU más precisa es la radiación con iones pesados sobre un componente comercial o prototipo, además esta técnica permite estudiar no sólo el comportamiento de los mecanismos de mitigación de fallos implementados en el sistema sino también la sensibilidad de la tecnología. Sin embargo el coste del equipamiento necesario es muy alto. Este factor, unido a las limitaciones en observabilidad y controlabilidad del proceso de inyección hacen que la aplicación de esta técnica se reduzca a medidas finales sobre confiabilidad, aplicándose previamente otras técnicas de inyección para obtener resultados preliminares de la tolerancia a fallos del circuito.

Cuando la descripción de un circuito no está disponible y se desea realizar una evaluación preliminar de la confiabilidad del sistema es necesario utilizar técnicas de inyección en componentes comerciales y es muy interesante que dichas técnicas sean de bajo coste. En el caso de los microprocesadores, las capacidades de depuración que los procesadores actuales tienen integradas proporcionan un mecanismo de bajo coste para la inserción de fallos en los recursos internos sin modificar la aplicación a ejecutar.

El estudio del estado de la técnica realizado permite concluir que las técnicas de inyección existentes, aplicables durante la etapa de diseño del circuito tolerante a fallos, están limitadas en velocidad porque la mayor parte de las tareas de inyección, como el control o la observación de los efectos, se realizan desde un PC. Debido a la complejidad de los diseños actuales, esas técnicas proporcionan una velocidad insuficiente y es necesario proponer nuevas técnicas de inyección que aceleren el proceso de evaluación. Además, puesto que la tendencia actual en el diseño de CIs es la

integración de distintos componentes, SoCs y sistemas empotrados, se requieren técnicas que permitan la evaluación de distintos tipos de circuitos. En esta tesis se proponen técnicas de inyección basadas en emulación de fallos con FPGA. Una solución está orientada a la evaluación de un circuito digital cuando su descripción en un lenguaje de descripción *hardware* está disponible, y otra está orientada a la evaluación de microprocesadores, componentes esenciales en los sistemas digitales actuales.

En el capítulo 4 se describe un sistema de inyección de fallos SEU basado en emulación con FPGAs con el que es posible acelerar el proceso de evaluación en hasta dos órdenes de magnitud con respecto a las técnicas de emulación de fallos propuestas por otros autores.

En el capítulo 5 se presenta una solución para evaluar la tolerancia a fallos de microprocesadores basada en el uso de las capacidades de depuración donde el objetivo es alcanzar un compromiso entre generalidad, velocidad, fácil implementación y que no requiera modificar ningún componente *hardware* o *software* del circuito.

CAPÍTULO 4

EMULACIÓN AUTÓNOMA DE FALLOS TRANSITORIOS

4. Emulación Autónoma de Fallos Transitorios

En este capítulo se propone una técnica de inyección de fallos transitorios SEU en circuitos digitales, basada en emulación con FPGAs y denominada Emulación Autónoma. La arquitectura del sistema de inyección propuesto implementa un conjunto de optimizaciones del proceso de inyección que aceleran considerablemente su ejecución con respecto a las técnicas existentes. La solución presentada permite la evaluación de cualquier circuito digital cuando se dispone de su descripción en un lenguaje de alto nivel. Se estudia en particular la evaluación en circuitos con memorias empotradas, lo que hasta ahora suponía un proceso lento, con respecto a la inyección en biestables, por su menor observabilidad y controlabilidad. La Emulación Autónoma es una aportación original de este trabajo de tesis doctoral.

4.1 Introducción

La emulación de fallos es una solución para la evaluación de la tolerancia a fallos que consiste en realizar la inyección sobre un prototipo del circuito implementado en una FPGA. Tradicionalmente, el dispositivo FPGA se utiliza únicamente para emular el circuito, mientras que el resto de tareas relacionadas con la inyección, como el control de la inyección o el análisis de resultados, se realizan en un computador externo, normalmente un PC. El objetivo de esta técnica es evaluar el comportamiento de los mecanismos de tolerancia a fallos del circuito durante la etapa de diseño. El prototipado del circuito a estudiar en un dispositivo programable proporciona un circuito funcionalmente idéntico a su versión final, lo que permite un análisis[15] adecuado de su tolerancia frente a SEUs. Junto con la descripción del circuito a estudiar, es necesario disponer de un conjunto de estímulos o banco de pruebas representativo del funcionamiento típico que realiza dicho circuito, puesto que la cobertura de fallos de un circuito depende de las partes que se ejercitan durante el proceso de inyección y por lo tanto del banco de pruebas usado.

Los trabajos realizados por otros autores, en relación con técnicas de inyección basadas en emulación con FPGAs (capítulo 3), prueban que la emulación de fallos es un método eficiente para evaluar la tolerancia a SEUs en circuitos digitales, y que acelera considerablemente el proceso de inyección con respecto a las técnicas basadas en simulación.

La velocidad del proceso de inyección es un factor fundamental especialmente cuando se evalúa la tolerancia a fallos de circuitos reales, dónde el número de posibles

[15] Como ya se dijo en el capítulo 2, la tolerancia a fallos SEU de la tecnología SRAM de la FPGA no afecta a este estudio y debe realizarse con otros métodos cuando el dispositivo final va a ser una FPGA.

fallos es elevado (del orden de cientos de millones[16]). Generalmente, para los circuitos actuales, es inviable realizar en un tiempo razonable una evaluación exhaustiva o de al menos un porcentaje significativo de fallos. Con el objetivo de realizar el proceso de evaluación de la tolerancia a fallos en un tiempo razonable, los métodos de evaluación existentes inyectan un subconjunto de fallos, generado según una distribución de probabilidad determinada, para obtener un resultado estadístico de la sensibilidad del circuito estudiado. Típicamente, dicho subconjunto consta de un número de fallos del orden de decenas de miles (o incluso menos), lo que supone menos de un 1% de todos los posibles SEUs.

Los métodos basados en emulación realizan la ejecución del circuito a velocidad *hardware*, acelerando el proceso de inyección significativamente. Las técnicas de emulación propuestas por otros autores permiten inyectar cientos de miles de fallos en varias horas. Sin embargo, cientos de miles de fallos pueden no ser un número significativo para evaluar los circuitos actuales. Acelerar el proceso de emulación permitiría realizar campañas de inyección con un número mayor de fallos evaluados. Este es uno de los principales objetivos de este trabajo de tesis doctoral.

Según el estudio del estado de la técnica realizado (capítulo 3), se puede concluir que la limitación en velocidad más importante en las técnicas de emulación de fallos es el protocolo de comunicación entre el PC y la FPGA. Las técnicas propuestas hasta el momento utilizan el computador para el control de la campaña de inyección, lo que implica una comunicación intensiva entre el PC y la FPGA. La transmisión de información necesaria para realizar las tareas de inyección en el PC limita la velocidad máxima que podría alcanzar la emulación de fallos, gracias a las prestaciones que proporcionan las FPGAs actuales.

En este capítulo se presenta un nuevo sistema de inyección, que es una aportación original de este trabajo de tesis, cuyo principal objetivo es acelerar el proceso de emulación. La arquitectura propuesta implementa un sistema de emulación autónomo que no requiere la comunicación con el PC durante el proceso de inyección. Está basado en la modificación del circuito añadiendo lógica adicional para la inserción de fallos. Además, incluye una serie de optimizaciones en la clasificación de fallos y en el protocolo de emulación que reducen el tiempo empleado por cada fallo, proporcionando un sistema de inyección capaz de realizar campañas exhaustivas o bien con conjuntos de fallos mucho mayores que los que se podían considerar hasta ahora.

[16] En un circuito con un número F de elementos de memoria y un banco de pruebas con una longitud de C ciclos de reloj, existen $C \cdot F$ posibles fallos *bit-flip*. Suponiendo un circuito medio con 5,000 elementos de memoria y un banco de pruebas con 100,000 ciclos, el número de posibles fallos asciende a $5 \cdot 10^8$.

En primer lugar se presenta la arquitectura del sistema de inyección propuesto, denominado sistema de Emulación Autónoma. En el apartado 4.3, se describen las posibles optimizaciones aplicables a la emulación de fallos, se proponen soluciones para su implementación dentro de la arquitectura de Emulación Autónoma, así como los recursos que son necesarios. A continuación, en el apartado 4.4, se proponen tres posibles implementaciones distintas del sistema de inyección, y se analizan sus diferencias. El estudio de *bit-flips* en biestables no presenta problemas de acceso, sin embargo, el caso en el que el circuito contiene memorias empotradas plantea nuevas dificultades de acceso, tanto para la inyección como para la observación de los efectos. Dichas dificultades se describen en el apartado 4.5, junto con las soluciones propuestas que constituyen una aportación original. Finalmente, se presentan las conclusiones del capítulo.

4.2 Arquitectura propuesta

Las FPGAs actuales son dispositivos muy potentes y proporcionan altas prestaciones, por lo que permiten emular grandes circuitos a una alta velocidad. Sin embargo, los sistemas de emulación propuestos hasta la actualidad presentan ciertas limitaciones que impiden obtener el máximo rendimiento de las prestaciones que ofrecen dichos dispositivos. En estos sistemas de emulación de fallos el control de la campaña de inyección se realiza desde el PC. Esto implica una intensa comunicación entre el *software* y la FPGA para controlar el proceso de inyección, aplicar los estímulos de entrada y observar el funcionamiento del circuito para clasificar los fallos.

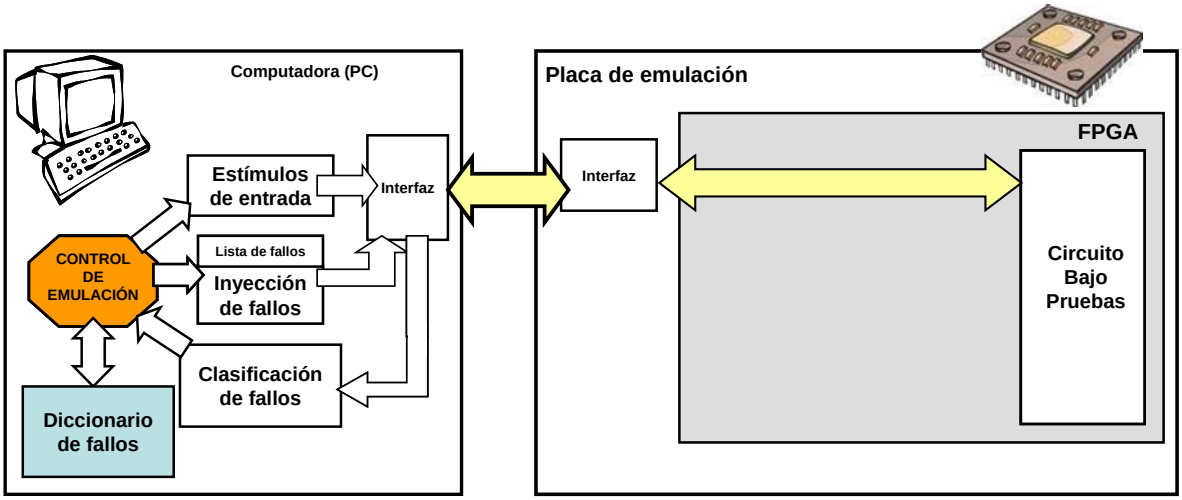
La transmisión de la información necesaria entre PC y FPGA para realizar las tareas de inyección es lenta con respecto a la velocidad de ejecución de la FPGA, especialmente, el estudio del comportamiento del circuito, por la limitada observabilidad. Esto conlleva interrupciones en la ejecución *hardware* cada vez que se requiere una interacción con el PC, e impone una limitación importante en la velocidad del proceso de evaluación. Por lo tanto, es necesario minimizar dicha comunicación todo lo posible.

En esta tesis doctoral, se presenta un entorno de inyección de fallos *bit-flip* basado en emulación, que no requiere la interacción con el PC durante el proceso de evaluación. El sistema propuesto se denomina *Sistema de* ***Emulación Autónoma*** y consiste en implementar el sistema completo de inyección en la FPGA. De esta forma, el acceso por parte del sistema de inyección a cualquier elemento del circuito es directo, y se reduce considerablemente el tiempo empleado en realizar las distintas tareas del proceso de evaluación. Para su aplicación se requiere una descripción del circuito en un lenguaje de algo nivel, como VHDL, y el banco de pruebas correspondiente a la

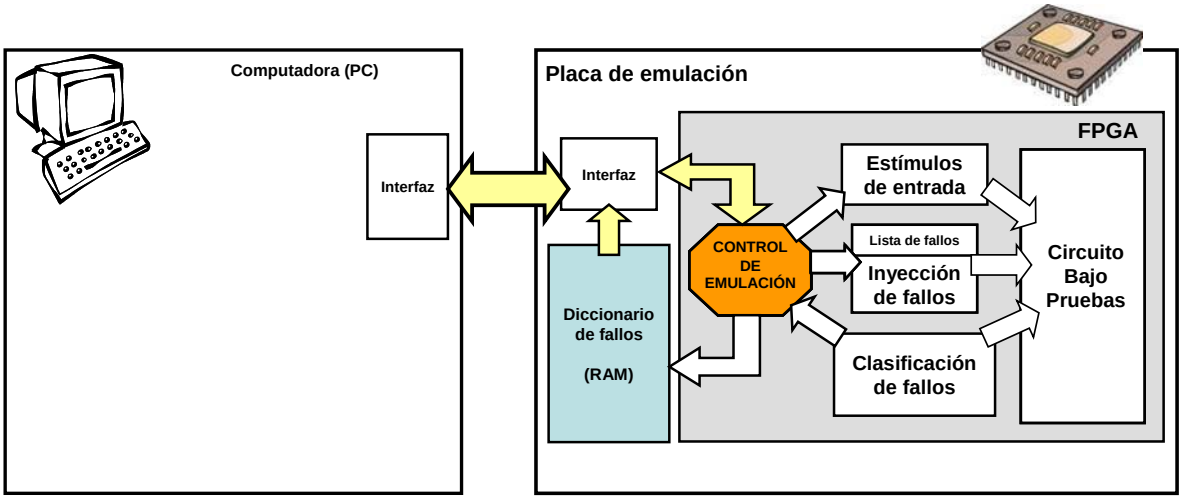
funcionalidad que se quiere evaluar. En el sistema de Emulación Autónoma, la FPGA se encarga de las siguientes tareas:

- Control del proceso completo de inyección.
- Aplicación de los estímulos de entrada al circuito.
- Activación de la inyección del fallo.
- Observación del comportamiento del circuito y clasificación de los efectos provocados a consecuencia de la inyección del fallo.

En la Figura 35 se muestran tanto la estructura general de un sistema de emulación (Figura 35.a), como la arquitectura de Emulación Autónoma propuesta (Figura 35.b). La comparación entre ambos esquemas ilustra de manera gráfica las mejoras que se proponen en el nuevo sistema de emulación.



a) Estructura tradicional de un sistema de inyección basado en emulación



b)Estructura del sistema de Emulación Autónoma

Figura 35. Arquitectura propuesta para el sistema de inyección autónomo

El entorno de inyección propuesto hace uso de los recursos disponibles en las plataformas FPGA actuales, como los bloques de memoria RAM, para minimizar las tareas que se realizan desde el PC. Por ejemplo, en la Figura 35.b se apuntan algunos detalles de implementación como el almacenamiento de los resultados de la emulación en la memoria RAM disponible en la plataforma de emulación, hasta el final de la campaña, de forma que sólo se descargan al PC una vez finalizada la evaluación.

Al realizarse en *hardware* todas las tareas de inyección y, en concreto, aquellas que requieren una comunicación intensiva con el circuito bajo estudio, se obtienen los siguientes beneficios:

1. La comunicación entre la FPGA y el PC requerida se minimiza, estableciéndose únicamente en dos ocasiones, al comienzo de la campaña de inyección para configurar la plataforma de emulación desde el PC, y al finalizar dicha campaña, para recoger los resultados obtenidos, normalmente el diccionario o clasificación de los fallos.
2. El acceso a los distintos elementos del circuito es directo, mejorando y facilitando su controlabilidad y observabilidad. Generalmente, en las técnicas de emulación propuestas hasta el momento, se busca un compromiso entre la observabilidad y la velocidad de inyección de fallos, puesto que una mayor observabilidad en esas soluciones implica la lectura de información desde el PC y por lo tanto, un mayor tiempo empleado en el proceso de evaluación. Con el sistema de Emulación Autónoma, se obtiene una alta observabilidad sin perjuicio en la velocidad de inyección, puesto que el sistema de inyección y el circuito bajo estudio están implementados en el mismo dispositivo.
3. La ejecución de las tareas de inyección en *hardware* es más rápida que en *software*, puesto que pueden paralelizarse, especialmente las labores de cálculo.

Por otra parte, el PC se encarga de las tareas de configuración del sistema de inyección. Las tareas realizadas por el PC no suponen un coste excesivo en tiempo puesto que se realizan una sola vez en cada campaña de inyección. Dichas tareas son las siguientes:

- Generar el fichero de configuración de la FPGA y programar el dispositivo.
- Activar la emulación para que comience la campaña de inyección.
- Descargar el diccionario de fallos obtenido al finalizar el proceso de emulación para su análisis.

El diagrama de bloques de la arquitectura propuesta, representada en la Figura 35.b, muestra los distintos componentes y funciones que en un sistema de Emulación Autónoma se implementan en *hardware*. Al igual que en el resto de entornos de

emulación, el circuito bajo estudio se implementa en la FPGA para ejecutar en *hardware* el funcionamiento en presencia de fallos. La novedad del sistema presentado reside principalmente en el bloque controlador. Al estar implementado también en la FPGA, las tareas que desempeña se ejecutan a velocidad *hardware* y se elimina el cuello de botella que se produce en la comunicación con el PC. Esta mejora lleva acompañada la implementación de otros módulos que permitan la realización de dichas tareas. A continuación se describen en detalle todos los módulos implementados en *hardware*:

- **Estímulos de entrada**. Los vectores de entrada al circuito deben estar disponibles en la FPGA para que su aplicación no ralentice la ejecución del circuito. Para ello, pueden almacenarse en una memoria interna o bien generarse concurrentemente, durante el funcionamiento del circuito. El almacenamiento del banco de pruebas en memoria consume gran cantidad de recursos internos y puede suponer una limitación cuando se trata de grandes cantidades de datos. En ese caso, se pueden aplicar técnicas de compresión para reducir la información a almacenar.

- **Clasificación de fallos.** Este bloque representa la función de observación y clasificación de los fallos. Al implementar esta tarea dentro de la FPGA el acceso a los recursos internos del circuito es directo, mejorándose la observabilidad y acelerándose el proceso de clasificación de fallos, que es uno de los más costosos de los sistemas de inyección. El *hardware* necesario para realizar esta tarea depende de la clasificación que se quiera realizar. Para la detección de averías es necesario observar las salidas del circuito y compararlas con las esperadas cuando el circuito se ejercita libre de fallos. Para un estudio más profundo de la tolerancia a fallos del circuito, se deben observar los elementos internos de memoria, para comprobar si los fallos son o no latentes. En cualquier caso, los resultados esperados del circuito sin fallos, deben estar disponibles también en el *hardware*, bien almacenados en memoria o bien deben generarse durante la campaña de inyección. En el apartado 4.4 se proponen distintas implementaciones del sistema y por lo tanto distintas soluciones para realizar esta función.

- **Inyección de fallos.** Este módulo realiza la inyección de fallos. En el capítulo 3, se explicó que en las soluciones basadas en emulación se distinguen dos tipos de mecanismos de inyección, los basados en reconfiguración y los basados en la modificación o instrumentación del circuito para añadirle la capacidad de auto-inyectarse fallos. Los resultados obtenidos por otros autores prueban que la técnica más rápida es la basada en la instrumentación del circuito puesto que la introducción de un *bit-flip* se realiza en un único ciclo de reloj, mientras que la

descarga de parte del *bitstream* es un proceso más lento. El principal objetivo de este trabajo de tesis doctoral es acelerar el proceso de inyección, por lo que en el sistema de Emulación Autónoma se utiliza la instrumentación del circuito como mecanismo de inyección. Además, las técnicas basadas en reconfiguración requieren un conocimiento de la estructura del *bitstream* para controlar la zona del circuito en la que se inyecta el fallo. Esto hace que la solución dependa de la tecnología de la FPGA para la cual se desarrolla la técnica. La implementación *hardware* del control de de la inyección mediante la instrumentación del circuito es más sencilla que en el caso de la reconfiguración. En definitiva, la técnica más adecuada, compatible con la arquitectura del sistema propuesto, es la instrumentación del circuito.

- **Circuito bajo estudio.** Consiste en el circuito a evaluar modificado con la lógica necesaria para inyectar fallos y observar su comportamiento. En el apartado 4.4 se presentan tres posibles modificaciones del circuito y un análisis comparativo que estudia las ventajas e inconvenientes de cada caso.
- **Control de emulación.** Este bloque consiste en una máquina de estados que se encarga de controlar el proceso completo de inyección, insertando cada fallo, decidiendo cuando la emulación de un fallo finaliza y es necesario continuar insertando más fallos o por el contrario la campaña ha terminado, etc. Para cada fallo, el control de la emulación se encarga de las siguientes tareas:
  1. Leer las características del fallo a insertar de la lista de fallos. Estas características en el caso de un SEU son el elemento de memoria afectado y el instante de tiempo en el que se produce dicho fallo, esto es, el instante de inyección.
  2. Inicializar el circuito para comenzar la ejecución desde un estado conocido.
  3. Controlar la ejecución del circuito, aplicando los estímulos de entrada correspondientes a cada ciclo de ejecución.
  4. Controlar el tiempo de ejecución y activar la inyección en el elemento de memoria que se quiere evaluar y en el instante de tiempo adecuado, en función de la lista de fallos.
  5. Habilitar la comparación entre el comportamiento del circuito con fallos y el comportamiento esperado para clasificar el fallo. Cuando se determina el efecto que el fallo ha provocado en el circuito, el control de emulación se encarga de almacenar la clasificación del fallo obtenida, utilizando por ejemplo la memoria RAM disponible en la plataforma de emulación.

6. Controlar la ejecución de cada fallo, deteniendo la emulación de un fallo cuando finaliza el banco de pruebas o se conoce el tipo de efecto causado y comenzando la emulación del siguiente hasta finalizar con la campaña completa cuando todos los fallos de la lista de fallos se han evaluado.

Como ya se ha dicho, con la arquitectura propuesta de Emulación Autónoma, la comunicación entre el emulador *hardware* y el PC se minimiza notablemente y se obtienen una gran controlabilidad y observabilidad de los elementos del circuito, que no proporcionan las técnicas existentes hasta el momento. Una vez reducida la comunicación con el PC, la manera de acelerar más el proceso de emulación consiste en reducir el tiempo empleado en la emulación de cada fallo. La mayor controlabilidad y observabilidad del circuito que proporciona la Emulación Autónoma (con respecto al estado de la técnica), permiten introducir optimizaciones del proceso de inyección, para reducir el tiempo necesario en emular un fallo. En el siguiente apartado se presentan una serie de optimizaciones con el objetivo de aumentar la velocidad del proceso de inyección.

4.3 Optimizaciones en el proceso de inyección

Las mejoras a introducir en el sistema de inyección con el objetivo de acelerar el proceso de emulación pueden ir orientadas en dos direcciones: reducir el tiempo necesario para evaluar cada fallo o reducir la campaña de inyección (lista de fallos).

Con respecto al tiempo de emulación de un fallo, las optimizaciones que se proponen se basan en reducir el tiempo empleado en los distintos pasos necesarios para la inyección de cada fallo. En general, para un banco de pruebas dado, los pasos necesarios para emular cada fallo son los siguientes, representados gráficamente en la Figura 36:

- Llevar el estado del circuito al instante en el que se quiere insertar el fallo. La forma más sencilla de hacerlo, y por eso es la que se usa de forma habitual, consiste en ejecutar el banco de pruebas desde el inicio hasta el instante de inyección.
- Realizar la inyección del fallo.
- Clasificar el fallo en base a su efecto sobre el circuito. Se continúa la aplicación del banco de pruebas desde el instante de inyección hasta el momento en el que se clasifica el fallo o finaliza la ejecución.

Por lo tanto, para emular un fallo puede ser necesario ejecutar el banco de pruebas completo cuando el fallo no se clasifica hasta el final del mismo. En ese caso, cada fallo supondría C ciclos de emulación, donde C es la longitud del banco de pruebas Para un número elevado de fallos, puede suponer un tiempo de emulación excesivo.

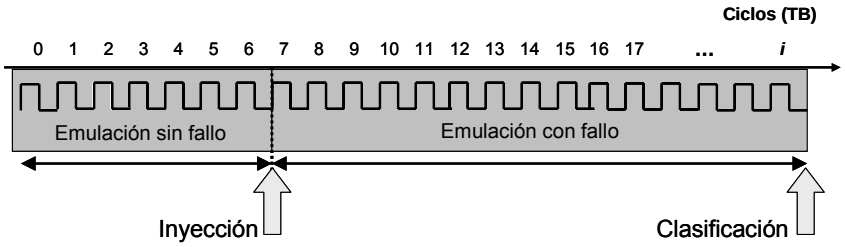


Figura 36. Tiempo de ejecución necesario para emular cada fallo.

Gracias al uso del mecanismo de inyección mediante instrumentación del circuito, la inyección del fallo se realiza mediante lógica durante el funcionamiento del circuito, sin necesidad de detenerlo. Por lo tanto, las optimizaciones en la emulación de un fallo se aplican en las otras dos tareas, llevar el estado del circuito al instante de inyección y clasificar el fallo. Para acelerar el proceso de alcanzar el estado del circuito en el instante de inyección se propone restaurar el circuito con el estado previo al instante de inyección, solución que se presenta en el apartado 4.3.1. Por otra parte, la clasificación del fallo se acelera averiguando el efecto provocado en el comportamiento del circuito lo antes posible. Esta última optimización se describe en el apartado 4.3.2. Ambas optimizaciones pueden aplicarse de forma eficiente por las características del sistema de Emulación Autónoma (alta controlabilidad y observabilidad).

Por otro lado, la reducción de la campaña de inyección consiste en disminuir la lista de fallos a inyectar, necesarios para obtener resultados significativos sobre la tolerancia a fallos del circuito. Para ello, se establecen mecanismos que permitan conocer el efecto de ciertos fallos sin necesidad de realizar su emulación, como ocurre, por ejemplo en el caso de insertar un *bit-flip* en una parte del circuito que nunca se ejercita. La técnica de colapsación de la lista de fallos que se propone se describe en el apartado 4.3.3.

A continuación, se detalla el conjunto de optimizaciones del proceso de emulación, propuestas y desarrolladas en esta tesis doctoral.

4.3.1 Restauración del estado previo al instante de inyección

El estado previo al instante de inyección puede alcanzarse de dos maneras:

- Emulando el banco de pruebas hasta el instante previo a la inyección.
- Cargando el estado correspondiente en el circuito. Para ello, es necesario disponer de los valores contenidos en todos los elementos del circuito en el instante de inyección.

Suponiendo que la carga del estado del circuito correspondiente al instante de inyección se realiza en un único ciclo de reloj, la restauración del estado evita la emulación del circuito sin fallo desde el inicio para cada fallo. Para realizar una

estimación del tiempo que permite ahorrar este mecanismo, supongamos una campaña de inyección en la que el banco de pruebas consta de *C* ciclos de reloj y donde el circuito a estudiar contiene *F* elementos de memoria sensibles a un SEU. Se considera que todos los elementos de memoria tienen la misma posibilidad de ser afectados por un fallo en cualquiera de los posibles instantes de inyección, por lo que el número de fallos posibles es $F \cdot C$. En el peor caso, serán necesarios *C* ciclos para emular cada fallo. Para un biestable determinado, si el fallo se inserta en el primer ciclo de reloj del banco de pruebas, se requiere un único ciclo de ejecución para alcanzar el instante de inyección; si el fallo se inyecta en el segundo ciclo del banco de pruebas, se requieren dos ciclos de ejecución para alcanzar el instante de inyección, y así sucesivamente. De esta forma, para un biestable determinado el tiempo total que se emplea en llevar al circuito al instante de inyección es de $(1+2+3+\ldots C)$[17]ciclos de reloj. Por lo tanto, teniendo en cuenta que el mecanismo de restauración del estado del circuito se realiza en un único ciclo de reloj, en la campaña de inyección completa con $F \cdot C$ fallos, dicho mecanismo evitaría la emulación de un número C_s de ciclos de ejecución, dónde:

$$C_s = \boldsymbol{C}\cdot\boldsymbol{F}\cdot(1+2+3+\ldots+\boldsymbol{C}\text{-}1) \Rightarrow \boxed{C_s = \boldsymbol{F}\cdot\frac{\boldsymbol{C}^2\cdot(\boldsymbol{C}-1)}{2}}$$

La restauración del estado es una optimización efectiva que permite acelerar el proceso de inyección de cada fallo, como se demuestran los resultados experimentales obtenidos, presentados en el capítulo 6. En el apartado 4.4 se presentan distintas soluciones para implementar esta optimización.

4.3.2 Clasificación de fallos

La emulación de un fallo finaliza cuando se alcanza el final del banco de pruebas o bien se clasifica el fallo inyectado dentro de alguna de las categorías establecidas por el diseñador. La clasificación más habitual consiste en distinguir entre averías, fallos silenciosos y fallos latentes. Como se explicó en el capítulo 3, las averías son un mal funcionamiento del circuito, siendo las salidas de éste distintas de las esperadas en ausencia de fallos. Un fallo es silencioso cuando sus efectos desaparecen, bien porque han sido enmascarados por un mecanismo de tolerancia a fallos o bien porque se ha sobrescrito debido al funcionamiento normal del circuito. Cuando el efecto del fallo permanece almacenado en el circuito sin provocar ningún funcionamiento erróneo antes del fin del banco de pruebas, el efecto se clasifica como latente. Estos tipos de efectos son los más habituales, pero pueden existir otros dependiendo del circuito y los

[17] $\sum_{i=1}^{n} \mathrm{i} = \frac{\mathrm{n}\cdot(\mathrm{n}+1)}{2}$

requisitos de tolerancia a fallos. Por ejemplo, para circuitos microprocesadores, suele considerarse una pérdida de secuencia como un efecto a distinguir.

Puesto que tras la inyección de un fallo la emulación del circuito está dirigida a estudiar y averiguar el efecto que aquél provoca, es importante reducir el tiempo empleado en realizar dicha clasificación para acelerar el proceso de evaluación. Generalmente, las técnicas de inyección de fallos detienen la emulación de un fallo tan pronto como detectan que se ha producido una avería. Sin embargo, cuando el fallo es silencioso o latente el efecto se determina una vez finalizado el banco de pruebas, momento en el cual se lee el estado del circuito para comprobar si el estado es correcto o no. Los fallos latentes requieren una ejecución completa, pero los silenciosos pueden detectarse tan pronto como desaparezca el efecto del fallo y el estado del circuito coincida de nuevo con el estado esperado en ausencia de fallos.

Acelerar la clasificación de los fallos latentes y silenciosos requiere el acceso a todos los elementos de memoria del circuito de forma rápida y continua, comparando su contenido con el estado esperado para el circuito libre de fallos (Figura 37). Esto es posible en el entorno de inyección propuesto gracias a que el sistema completo está implementado en el mismo dispositivo *hardware*.

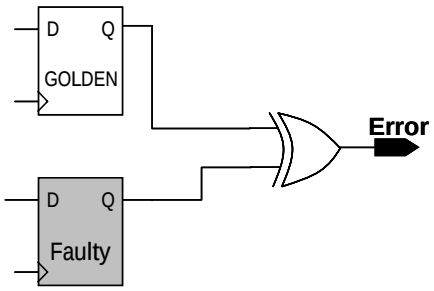


Figura 37. Comparación del estado del circuito con el valor esperado en ausencia de fallos para la detección de fallos silenciosos.

Optimizar la detección de fallos silenciosos proporciona una mejora considerable, sobre todo en el caso de circuitos tolerantes a fallos que contengan estructuras para enmascarar fallos o corregirlos. En consecuencia, aplicando esta optimización, junto con la restauración del estado previo a la inyección descrito en el punto anterior, el tiempo empleado en cada fallo se ve drásticamente reducido. La emulación de un fallo se reduce a unos pocos ciclos de ejecución.

La Figura 38 muestra gráficamente el tiempo empleado en cada fallo cuando se aplica la detección temprana de fallos silenciosos y la restauración del estado previo a la inyección, ilustrando la disminución del tiempo de ejecución necesario que estas optimizaciones proporcionan.

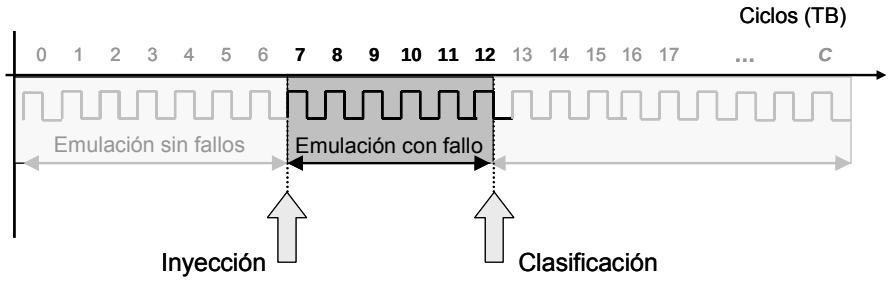


Figura 38. Tiempo empleado para la emulación de un fallo cuando se aplican las optimizaciones propuestas en 4.3.1 y 4.3.2.

4.3.3 Colapsación dinámica de fallos

La campaña de inyección se puede reducir cuando es posible conocer el efecto de los fallos sin necesidad de inyectarlos en el circuito. Para ello se estudian las posibles equivalencias entre distintos fallos. En el capítulo 3 se describen las técnicas de colapsación de fallos propuestas en la literatura con el objetivo de acelerar la inyección de fallos basada en simulación [Berr02a]. A continuación se presenta un nuevo método de colapsación de fallos aplicable durante la emulación de fallos, gracias a la mayor observabilidad que proporciona la Emulación Autónoma.

El sistema de Emulación Autónoma descrito en este capítulo permite acceder a cualquier parte del circuito de forma directa al estar el sistema de inyección implementado en el mismo dispositivo *hardware*. El mecanismo propuesto para realizar la detección temprana de fallos silenciosos implica la comparación continua entre los valores almacenados en los elementos de memoria del circuito con fallo, con los valores correctos esperados (Figura 37). Esta comparación proporciona al bloque de control de la emulación información sobre qué elementos de memoria almacenan valores erróneos, generando una señal de error por cada biestable.

Sea $E_{t0} = (e_0, e_1,\ldots, e_i,\ldots, e_{F-1})_{t0}$ el vector que representa las señales de error generadas en el instante de ejecución *t0* para cada uno de los biestables f_i del circuito, siendo F el número total de biestables. Es decir, el componente e_i del vector E_{t0} es el valor de la señal de error correspondiente al biestable f_i en *t0*. Antes de realizar la inyección de un fallo ningún biestable contiene un valor distinto al esperado por lo que $e_i = 0, \forall i$. En el instante de inyección t_{inj}:

$$E_{tinj}\begin{cases} e_{inj} = 1 \\ e_i = 0, \ \forall i \neq inj \end{cases}$$

donde el sub-índice *inj* se corresponde con el biestable afectado por el *bit-flip*.

Tras la inyección de un fallo, mientras no se realice un acceso al elemento de memoria afectado, bien sea de escritura o bien de lectura, se tiene que $E_t = E_{tinj.}$ En el momento de la ejecución *t'* en el que el biestable a evaluar se ejercite $E_{t'} \neq E_{tinj.}$ Todos

los fallos inyectados en el biestable f_{inj} en los instantes de tiempo comprendidos en el intervalo [t_{inj}, t') provocarán el mismo efecto en el comportamiento del circuito, es decir, son fallos equivalentes. Por lo tanto, sólo es necesario inyectar uno de esos fallos para poder clasificarlos.

La reducción de la lista de fallos a evaluar se utiliza fundamentalmente en técnicas de inyección mediante simulación, con el objetivo de reducir el coste computacional necesario para realizar el proceso de evaluación, véase el capítulo 3. Las técnicas de reducción o colapsación dinámica de la lista de fallos que se encuentran descritas en la literatura requieren el uso de grandes cantidades de memoria y espacio en disco. La técnica de colapsación dinámica descrita en [Berr02a] consiste en comparar el contenido de todos los elementos de memoria del circuito con los datos esperados, tras la inyección del fallo y durante la simulación. Por lo tanto, el estado esperado del circuito debe estar almacenado para poder realizar la comparación.

La mejora que se describe en este apartado no requiere almacenar información adicional a la que se genera durante la propia emulación del fallo. De forma que, partiendo de la lista completa de posibles fallos simples, el control del emulador es capaz de encontrar equivalencias entre fallos durante el proceso de inyección, y decida qué fallos son los que hay que inyectar. Esta optimización es especialmente notable en el caso de fallos latentes, puesto que requieren la ejecución del banco de pruebas completo y son los fallos cuya evaluación consume más tiempo.

La inclusión de este mecanismo en el sistema de emulación evita tener que disponer de una lista de fallos. Se asume que durante la campaña de inyección se insertarán todos los posibles fallos que pueden afectar a un conjunto de biestables dado [f_a, f_b], correspondientes a la parte del circuito que se desea evaluar, y en un intervalo de tiempo determinado [t_p, t_q). La inyección de fallos debe ser ordenada. A continuación se describe el protocolo a seguir:

1. Para cada biestable $f_i \in [f_a, f_b]$:
    1. $t_j = t_p$
    2. Se inyecta un *bit-flip* en el biestable f_i en el instante t_j.
    3. Se observa y almacena el instante de tiempo de la ejecución del banco de pruebas, *t'*, en el cual se accede al biestable afectado ($E_{t'} \neq E_{tinj}$), dónde $t' \in (t_j, t_q]$.
    4. Se observa el efecto que provoca el fallo y se clasifica dentro de alguna de las categorías consideradas.

5. Todos los fallos producidos en f_i en $t \in (t_j, t')$ son equivalentes al anterior y producen el mismo efecto, por lo que no es necesario inyectarlos para evaluarlos.
6. $t_j = t'$
7. Si $t_j = t_q$ se ha alcanzado el final del intervalo de tiempo en el cual se quieren inyectar fallos para el biestable f_i. En caso contrario, ($t_j < t_q$), volver al paso 2.

El bloque de control, Figura 35, se encarga de dirigir este proceso y decidir de forma dinámica los fallos que deben inyectarse, sin introducir para ello retrasos en el proceso de evaluación de la tolerancia a fallos.

4.4 Implementaciones de un sistema de Emulación Autónoma

Basándose en el concepto de Emulación Autónoma Existen distintas posibles implementaciones prácticas en función de cómo se diseñe el *hardware* de inyección o de las optimizaciones del proceso de inyección que se apliquen. En este apartado se describen tres implementaciones del sistema de Emulación Autónoma propuesto en este trabajo de tesis. Cada una de las implementaciones o técnicas de Emulación Autónoma propuestas incluye distintos niveles de optimización. El apartado 4.4.1 presenta la implementación más completa, mientras que 4.4.2 y 4.4.3 describen sistemas simplificados, reduciendo las optimizaciones incluidas en el emulador y por lo tanto los recursos *hardware* necesarios. Las tres implementaciones son acordes al sistema de emulación autónomo e inyectan fallos mediante la modificación del circuito bajo estudio. Por lo tanto la arquitectura del sistema en los tres casos es la presentada en la Figura 35.

La modificación del circuito consiste en reemplazar cada biestable del circuito por una estructura, que proporciona capacidad para modificar el contenido del biestable original, es decir, para auto-inyectarse fallos, y realizar las diferentes optimizaciones. Dependiendo de cómo se implemente la inyección o de las mejoras aplicadas dichas estructuras o instrumentos (término propuesto en [Cive01a]) difieren.

El estudio de las tres técnicas permite analizar la mejora obtenida con las distintas optimizaciones y estimar los requisitos en área para su implementación. El sistema de Emulación Autónoma y las optimizaciones del proceso de inyección propuestas implican un consumo de recursos de la FPGA, ya que la mejora en tiempo de ejecución se logra usando recursos *hardware* adicionales. Las FPGAs actuales contienen una gran cantidad de recursos, pero cuando el circuito a evaluar es muy grande puede ser necesario recurrir a una FPGA de mayor capacidad y cuantos más recursos tiene una FPGA mayor es su coste. Es por lo tanto, interesante estudiar la

relación existente entre la mejora de la velocidad del proceso obtenida con cada optimización y los recursos necesarios para ello. Se busca un compromiso que proporcione un método de inyección rápido, eficaz y de bajo coste. Dicho análisis permite conocer qué método es más adecuado en función de las necesidades del diseñador y los recursos disponibles.

4.4.1 Técnica basada en multiplexación en el tiempo: *Time-Multiplexed*

Esta técnica se corresponde con la implementación del sistema de Emulación Autónoma que incluye el mayor número de optimizaciones (punto 4.3), incluyendo la restauración del estado del circuito previo a la inyección y la clasificación temprana de fallos silenciosos. Con el objetivo de justificar la estructura o instrumento que se propone, se describe la lógica necesaria para implementar cada función u optimización que se desea incluir en el sistema de inyección.

4.4.1.1 Inyección de fallos

La instrumentación del circuito permite la inyección en un biestable del circuito mediante el control de ciertas señales. Para implementar la inserción de un *bit-flip* se utiliza el mecanismo propuesto en [Cive01a]. Esta estructura se describe en el capítulo 3 (Figura 27 en la página 63). Consiste en añadir un biestable adicional por cada uno de los propios del circuito para almacenar una máscara de fallo que indica el lugar en el que ha de inyectarse un *bit-flip.* Cuando el biestable de máscara almacena un '1' lógico se introducirá un *bit-flip* en el elemento de memoria asociado en el instante en el que ciertas señales de control se activen. Esta máscara también permite inyectar fallos múltiples en el circuito.

4.4.1.2 Optimizaciones implementadas

Las optimizaciones que requieren modificaciones en el circuito para su implementación son la detección rápida de fallos silenciosos y la restauración del estado.

1. Detección rápida de fallos silenciosos. Es necesario comparar de forma continua el estado del circuito con fallo con respecto al estado esperado. Se propone duplicar cada biestable, dedicando uno para la ejecución libre de fallos, biestable *golden*, y otro a la ejecución del circuito en presencia de SEUs, biestable *faulty*. La duplicación completa del circuito requiere un incremento considerable de área necesaria por lo se propone duplicar únicamente los biestables compartiendo la lógica combinacional. De este modo, es necesario, ejecutar alternativamente la emulación del circuito con y sin fallo, de forma multiplexada en el tiempo. En caso de disponer de

suficientes recursos *hardware*, la duplicación del circuito completo sería la solución más rápida.

2. Restauración del estado. Es necesario disponer del estado previo a la inyección en la plataforma de emulación. Almacenar todos los posibles estados del circuito consume muchos recursos lógicos. Se propone añadir un biestable adicional que almacene dicho estado durante la emulación. La efectividad de este mecanismo se basa en una inyección ordenada de fallos y en la disponibilidad de dos biestables, uno para ejecuciones con fallos y otro para la ejecución correcta. La lista de fallos se ordena en función del instante de tiempo en el cual se realiza la inyección. Se ejecuta el circuito hasta alcanzar el primer instante de inyección y se almacena el estado del biestable *golden* en el biestable de estado. Cuando el fallo se clasifica, se puede recargar dicho valor en los biestables *golden* y *faulty* y evitar así la ejecución desde el inicio del banco de pruebas.

4.4.1.3 Estructura propuesta

La estructura que se propone está representada en la Figura 39. Con dicho instrumento el sistema de inyección controla la inyección y la restauración del estado, y detiene la emulación de un fallo cuando su efecto desaparece.

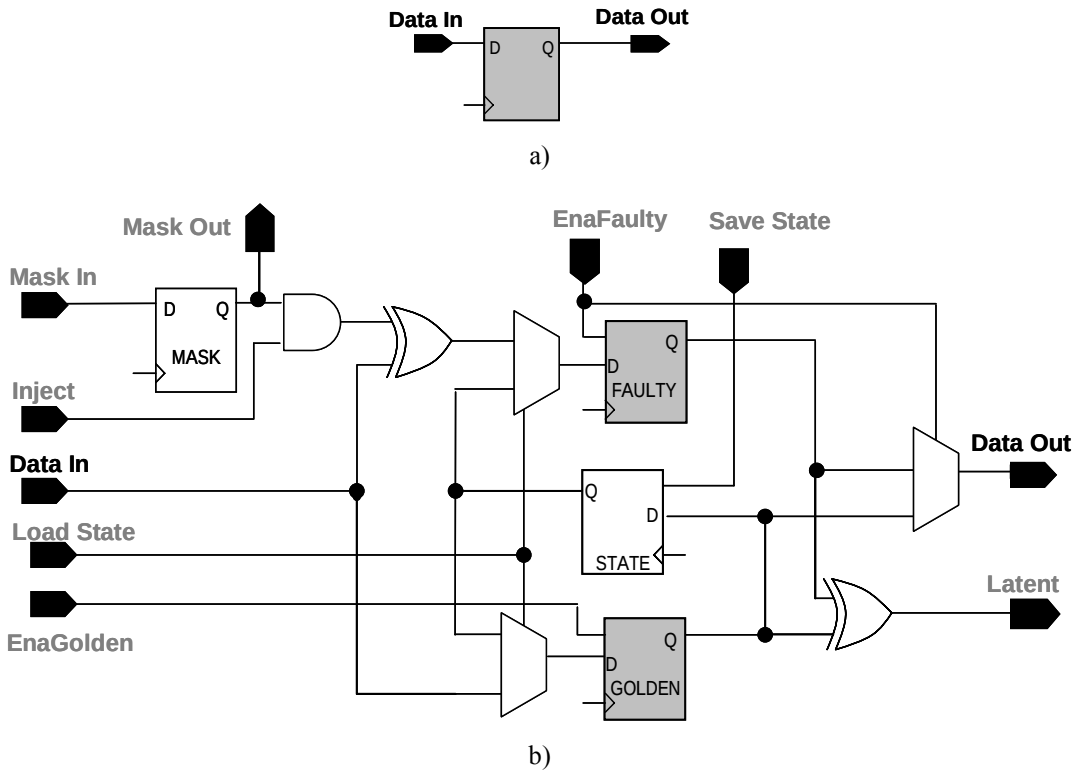


Figura 39. a) biestable original del circuito a modificar. b) lógica utilizada para reemplazar cada biestable original del circuito para la técnica de emulación *Time-Multiplexed*

Los biestables *faulty* y *golden* se utilizan para ejecutar el circuito con y sin fallo y generar así los resultados esperados y el estado interno del circuito durante la propia emulación del fallo. El biestable denominado *state* en la figura almacena el estado del biestable *golden* en un instante determinado, que posteriormente se utiliza para restaurar el estado previo a la inyección o un estado cercano, en función de los fallos a inyectar. El biestable *Mask* es el biestable que almacena la máscara de fallo y que permite introducir un *bit-flip* en el biestable *faulty*. Todas las señales de carga o selección del instrumento son asignadas por el control del emulador. Instrumentar un circuito de acuerdo con esta técnica implica, por tanto, multiplicar cada biestable por cuatro.

Los biestables de la máscara de fallos están conectados entre sí en serie, formando una cadena (Figura 40). Si la campaña de fallos a realizar es exhaustiva, es decir, se consideran todos los posibles fallos simples, la inyección de un fallo se puede realizar en un único ciclo de reloj. Se comienza la inyección por el primer biestable de la máscara de fallos por lo que cargar la máscara sólo requiere un ciclo de reloj. Se evalúan todos los fallos que pueden afectar a ese elemento, de forma que para esos fallos no es necesario modificar el valor de la máscara de fallo. Para inyectar fallos en el siguiente biestable tan sólo es necesario desplazar el valor que había almacenado en el biestable *Mask* una posición anterior en la cadena, lo cual consume un único ciclo de nuevo. De esta forma se reduce al máximo el tiempo empleado en la inserción del fallo, evitando cargar la máscara completa para cada fallo.

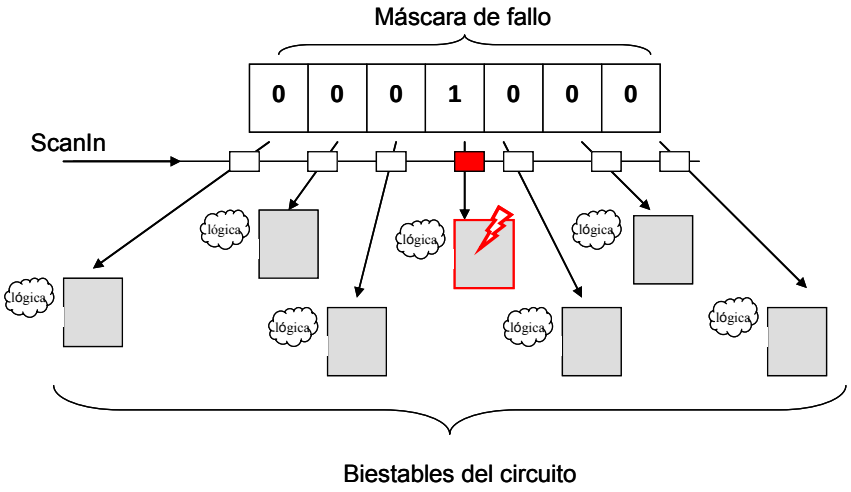


Figura 40. Esquema de la cadena formada por los biestables de la máscara para la inserción de fallos

Con la estructura propuesta cada biestable se cuadruplica, por lo que el área necesaria tiene un incremento notable. Por esta razón se ha optado por evitar duplicar el circuito completo y compartir la lógica combinacional. Puesto que la ejecución está multiplexada en el tiempo, se requiere el doble de ciclos emulados. Este hecho tiene una influencia importante en el caso de tener fallos latentes. Sin embargo, se espera que el diccionario de fallos obtenido de una campaña de evaluación contenga

mayoritariamente fallos silenciosos o averías y, en el caso de circuitos tolerantes a fallos, principalmente fallos silenciosos o detectados por algún sistema de tolerancia a fallos. En esos casos, el inconveniente impuesto por la multiplexación en el tiempo está contrarrestado por la mejora que proporciona una rápida detección de fallos silenciosos. Medidas experimentales sobre cómo afectan al proceso de evaluación ambos factores se presentan en el capítulo 6.

La Figura 41 muestra parte de la secuencia de ejecución de una campaña de inyección realizada aplicando la técnica *Time-Multiplexed*.

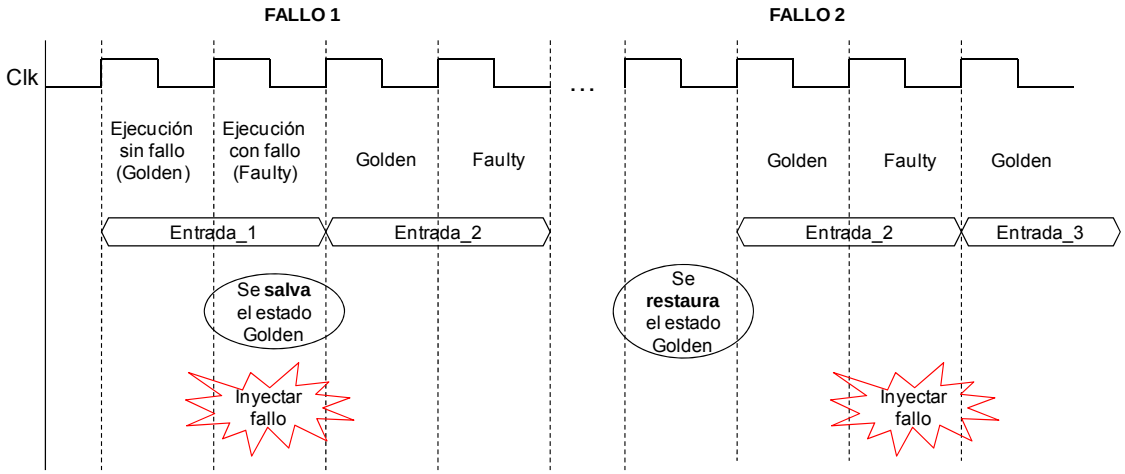


Figura 41. Secuencia de ejecución de una campaña de inyección de fallos aplicando la técnica *Time-Multiplexed*

En primer lugar, se carga la máscara de fallo en función del biestable que se quiere evaluar. En caso de tratarse de una inyección exhaustiva la máscara consiste en una cadena de ceros con un '1' lógico al final, de forma que se activa únicamente el primer biestable de la cadena.

Una vez cargada la máscara comienza la emulación del primer fallo. Primero se habilita el biestable que emula sin fallo (*GOLDEN*). En el siguiente ciclo de reloj, se deshabilitan los biestables *GOLDEN* para habilitar aquellos que emulan con fallo (*FAULTY*). En ese mismo ciclo se almacena el estado sin fallo en el biestable adicional *STATE* y se activa la inyección. El siguiente ciclo se ejecuta sin fallo de nuevo, aplicando el vector de entradas correspondiente al siguiente ciclo del banco de pruebas. La ejecución continúa alternando ciclos *golden* y *faulty* hasta que el fallo se clasifica o se alcanza el final del banco de pruebas. Entonces, se comienza la inyección del segundo fallo. Para ello se carga el estado del sistema que había sido almacenado anteriormente y que corresponde al estado anterior a la inyección del fallo. Se repiten los pasos realizados para el primer fallo hasta que se inyecten todos los fallos simples en el primer biestable. Después, se desplaza la máscara de fallo un lugar para que la inyección se produzca en el segundo biestable y se vuelve a repetir el proceso. Cuando

se han inyectado todos los fallos se leen desde el PC los resultados de la clasificación almacenados en la memoria RAM (Figura 35.b).

4.4.2 Técnica de inyección mediante scan-path: *State-Scan*

La técnica de inyección mediante *scan-path* o técnica *State-Scan* es una simplificación de la técnica *Time-Multiplexed* que permite reducir el área necesaria para su implementación.

El *scan-path* es una técnica proveniente del test de fabricación que utiliza un registro de desplazamiento dentro del camino lógico que siguen las señales dentro del circuito integrado, para permitir la inserción de señales de test a través de un único pin, el pin de entrada de datos de test, y extraer las respuestas de las señales a través de otro único pin, el pin de salida de datos de test.

En este trabajo se propone adoptar esta técnica para su aplicación en el entorno de la emulación con FPGAs. Así, en esta implementación se modifica la descripción en VHDL del circuito original para conectar los biestables del circuito formando una cadena de *scan*. De este modo es posible volcar el estado de los biestables del circuito con fallo en el instante de inyección.

4.4.2.1 Inyección de fallos

Los biestables originales del circuito se conectan en serie formando una cadena. Esto permite configurar el circuito con cualquiera de sus posibles estados mediante dicha cadena de *scan-path.*

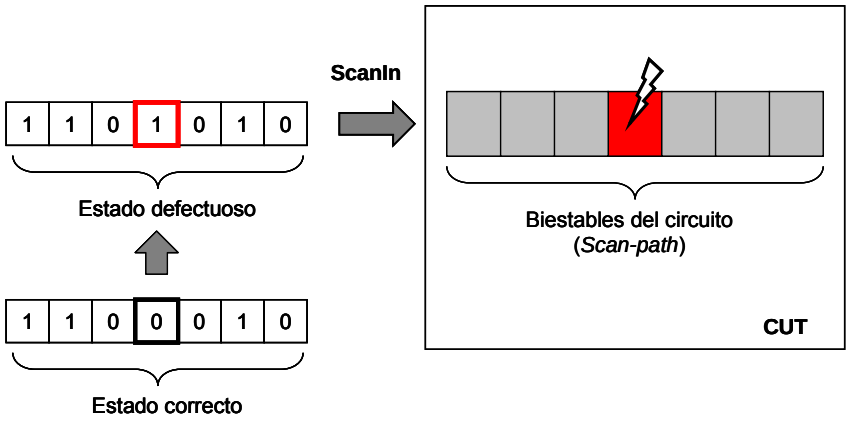


Figura 42. Mecanismo de inyección utilizado en la técnica de inyección *Scan-State*

Partiendo de los estados del circuito correspondientes a cada ciclo de ejecución sin fallo del banco de pruebas, la inyección se realiza descargando directamente el estado con un bit invertido, *bit-flip*, Figura 42. En este caso, no es necesario disponer de una máscara de fallo.

Los estados del circuito deben estar disponibles. Con el objetivo de que la ejecución de la emulación no requiera la comunicación entre el PC y la FPGA durante el proceso de inyección, los estados se almacenan en un bloque de memoria RAM de la plataforma FPGA utilizada para implementar el sistema de inyección. El control del emulador se encarga de la inserción de estos estados en el circuito, asignando los valores adecuados a las señales de control de la cadena de *scan-path*.

4.4.2.2 Optimizaciones implementadas

Con el objetivo de reducir la cantidad de lógica necesaria para la implementación del sistema de inyección y estudiar la influencia de las distintas optimizaciones sobre el proceso de evaluación, se reducen y simplifican las mejoras implementadas. Para la técnica *State-Scan* sólo se aplica la mejora relacionada con la restauración del estado:

1. Restauración del estado. El mecanismo de inyección utilizado en la técnica *State-Scan* carga el estado de inyección en el circuito, por lo que se evita emular los ciclos previos al instante de inyección, sin necesidad de añadir un biestable adicional encargado de almacenar el estado.
2. Detección rápida de fallos silenciosos. En esta técnica, esta optimización no se implementa. Para distinguir entre fallos latentes y fallos silenciosos sin necesidad de duplicar el circuito o la parte secuencial, como se propone para la técnica *Time-Multiplexed*, se compara el estado únicamente al final de la ejecución. El estado final del circuito se almacena en un biestable extra al final de la ejecución *golden*.

4.4.2.3 Estructura propuesta

La estructura que se propone está representada en la Figura 43.b.

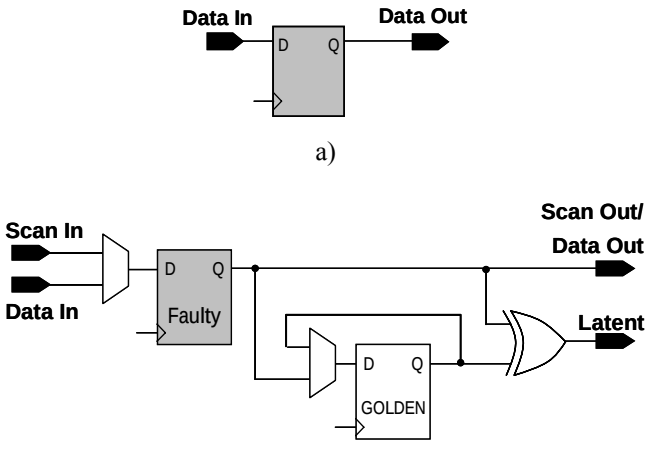


Figura 43. a) biestable original del circuito. b) estructura que reemplaza cada biestable para implementar la técnica de emulación *State-Scan*.

Instrumentar un circuito de acuerdo con esta técnica implica multiplicar cada biestable por dos, de forma que se reducen a la mitad los recursos necesarios para su implementación con respecto al caso de la técnica *Time-Multiplexed*. Por lo tanto, en términos de biestables el incremento de área necesario es del 100% mientras que aplicar la técnica anterior implica un incremento del 300%. Sin embargo, para poder observar diferencias en las salidas del circuito, clasificando el efecto del fallo como avería, las salidas esperadas deben estar almacenadas en una memoria. De esta forma se puede realizar la comparación con las salidas del circuito durante la emulación de cada fallo. Por lo tanto, la técnica *State-Scan* requiere una mayor cantidad de memoria disponible que la *Time-Multiplexed,* para almacenar tanto los estados del circuito con fallo como las salidas esperadas.

El biestable *Faulty* de la Figura 43.b almacena el estado del circuito durante la ejecución del banco de pruebas. Los biestables *Faulty* de todo el circuito se conectan en serie formando una cadena. Por otro lado, el biestable *Golden*, almacena el estado final del circuito para poder clasificar los fallos latentes y distinguirlos de los silenciosos. El instrumento contiene también lógica combinacional destinada a conectar los biestables *Faulty* y cierta lógica para realizar la comparación entre el estado del circuito y el estado final esperado. Todas las señales de carga o selección del instrumento son asignadas por el bloque de control del emulador.

En esta técnica la ejecución no se multiplexa en el tiempo por lo que la emulación de cada vector del banco de pruebas se realiza en un ciclo de reloj, a diferencia de lo que ocurre si se aplica *Time-Multiplexed* (dos ciclos, *golden* y *faulty*). Sin embargo, cargar el estado completo del circuito correspondiente al instante de inyección requiere, con la técnica *State-Scan*, un número de ciclos igual al número de biestables del circuito. La Figura 44 muestra parte de la ejecución de una campaña de inyección de fallos aplicando *State-Scan*.

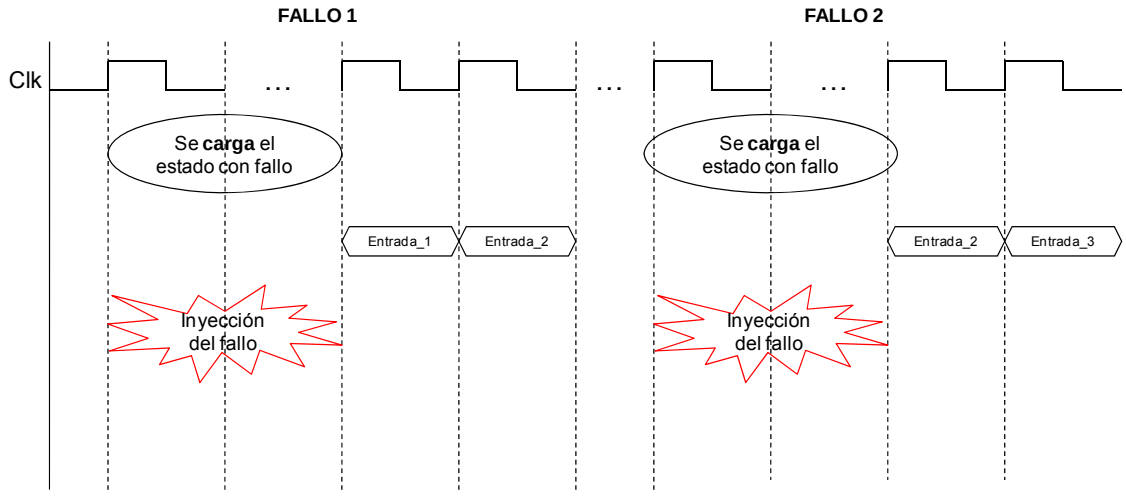


Figura 44. Secuencia de ejecución de una campaña de inyección de fallos aplicando la técnica *State-Scan*

En primer lugar, se realiza una ejecución sin fallo y el estado final se almacena en el biestable *Golden* del instrumento. Comienza la emulación del primer fallo cargando todos los biestables del circuito con el estado correspondiente al instante de inyección deseado a través de la cadena de *scan-path*. La emulación finaliza cuando se produce una avería, se detecta el fallo mediante alguno de los mecanismos de tolerancia a fallos disponibles en el circuito o el banco de pruebas finaliza. En este último caso, se realiza una comparación entre el estado almacenado en los biestables *Golden* y el almacenado en los *Faulty*, para distinguir entre fallos latentes y silenciosos. A continuación comienza la emulación del siguiente fallo, y así sucesivamente hasta haber evaluado todos los fallos de la lista.

4.4.3 Técnica basada en la instrumentación del circuito: *Mask-Scan*

Esta técnica es la más sencilla. Implementa el menor número de mejoras y optimizaciones propuestas pero por esta razón es la que menos requisitos impone con respecto a recursos necesarios.

4.4.3.1 Inyección de fallos

La inyección de fallos se implementa utilizando el instrumento propuesto en [Cive01a], de igual forma que en el caso de la técnica *Time-Multiplexed*.

Por lo tanto, el sistema de inyección consta de una máscara de fallos que indica dónde se introduce el fallo cuando las señales de control correspondientes activen la inyección.

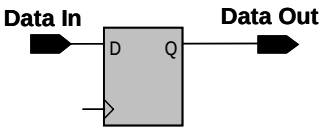
4.4.3.2 Optimizaciones implementadas

La técnica *Mask-Scan* es la más sencilla y no implementa la restauración del estado ni la detección rápida de fallos silenciosos.

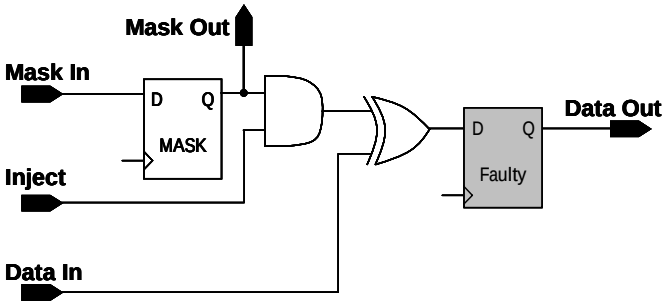
4.4.3.3 Estructura propuesta

La estructura que se propone está representada en la Figura 45.b. Puesto que no se implementa la detección rápida de fallos silenciosos ni la restauración del estado previo a la inyección, no son necesarios ni el biestable *State* ni el *Golden*. En este caso, la clasificación de fallos ofrece menos información puesto que el instrumento presentado en la Figura 45 no permite la detección de fallos latentes. El instrumento consta únicamente de un biestable adicional, que almacena el valor de máscara, y de lógica para inyectar el fallo si así está indicado en dicha máscara. Al instrumentar el circuito se produce un incremento del 100% de biestables utilizados, igual que en la técnica *State-Scan*. Las salidas esperadas están disponibles en una memoria para poder detectar una avería en el instante en el que se produzca. Por lo tanto, esta solución

requiere más memoria que la técnica *Time-Multiplexed*, pero menos que la basada en la cadena de *Scan*, al no almacenar los estados del circuito.



a)



b)

Figura 45. a) biestable original del circuito. b) estructura que reemplaza cada biestable para implementar la técnica de emulación *Mask-Scan*

El módulo de control genera todas las señales necesarias para ejecutar la inyección, así como las direcciones de memoria y las señales referentes al bloque de memoria RAM que almacena el diccionario de fallos, de la misma forma que ocurre en las otras dos técnicas propuestas.

Utilizando esta estructura, el banco de pruebas debe aplicarse desde el comienzo para cada fallo. Parte de la secuencia de ejecución de una campaña de inyección aplicando la técnica *Mask-Scan* se muestra en la Figura 46.

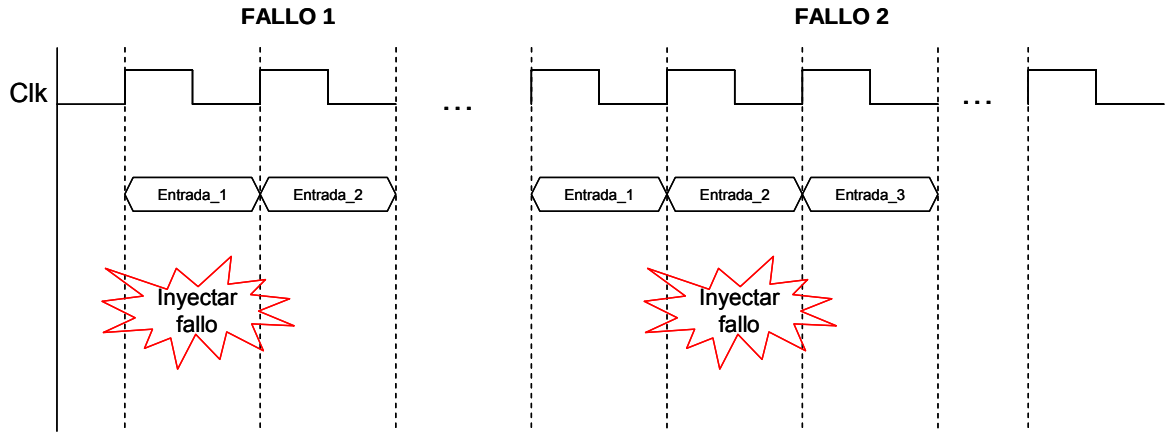


Figura 46. Secuencia de ejecución de una campaña de inyección de fallos aplicando la técnica *Mask-Scan*

La emulación de un fallo finaliza cuando ocurre una avería, un mecanismo de tolerancia a fallo detecta la presencia de un error o se alcanza el final del banco de pruebas. De esta forma, el proceso de evaluación es más lento que en el caso de aplicar *Time-Multiplexed*, puesto que no se aplica ninguna de las optimizaciones propuestas, pero se minimizan los recursos necesarios para su implementación.

4.4.4 Análisis comparativo

En este trabajo, se han propuesto tres soluciones para implementar un sistema de inyección de fallos transitorios basado en Emulación Autónoma en FPGAs. En un sistema de Emulación Autónoma la mayor parte de las funciones de inyección se realizan en *hardware*, en la plataforma de emulación, lo que permite ejecutar las tareas a mayor velocidad que su implementación *software*. Además, así se minimiza el intercambio de información necesario entre el PC y la FPGA puesto que esta comunicación es muy lenta y supone un cuello de botella en la velocidad del proceso.

Existen diferencias considerables entre las tres soluciones propuestas para implementar de forma práctica un sistema de Emulación Autónoma, dependiendo de cómo se resuelven los aspectos principales de la campaña de inyección de fallos, que son la lista de fallos, la inyección, la emulación y la clasificación de los fallos. Esto trae como consecuencia distintas implementaciones en *hardware* con distintos requisitos en términos de recursos necesarios. Así, por ejemplo, los datos que hay que almacenar o de los que hay que disponer para poder clasificar los fallos difieren en cada técnica. En la técnica *Mask-Scan* se almacenan las salidas esperadas en cada ciclo de reloj, en la técnica *State-Scan* son necesarias las salidas esperadas y el estado del circuito correspondiente al instante de inyección (el valor de cada biestable), y en la técnica *Time-Multiplexed* al realizar una ejecución con fallo y otra sin fallo estos valores se obtienen durante la propia emulación y no es necesario almacenarlos. Así los tamaños de memorias necesarias son distintos. Por lo tanto, los requisitos en área son diferentes en cada una de las técnicas propuestas.

Al incremento en área contribuyen las modificaciones realizadas en el circuito, el sistema de control necesario y las memorias. Así, aunque la técnica *State-Scan* es la propuesta en la que menos lógica hay que añadir al circuito, requiere una memoria RAM externa mucho mayor que el resto de las propuestas para almacenar los estados del circuito correspondientes al instante de inyección. En el capítulo 6 se analizan estos requisitos para varios circuitos.

Respecto al coste temporal de las distintas técnicas, se puede hacer una estimación. Esta estimación se contrasta en el capítulo 6 con los resultados experimentales obtenidos.

4.4.4.1 Estimación teórica del coste temporal

En esta descripción se usará ***F*** para representar el número de biestables que tiene el circuito, ***C*** para representar el número de vectores del banco de pruebas o lo que es equivalente el número de ciclos de reloj de la aplicación a ejecutar. El objetivo de este apartado es la estimación teórica del tiempo necesario para realizar la ejecución completa de la campaña de inyección de todos los fallos simples del circuito, es decir, se considera que la evaluación es exhaustiva. Este tiempo se divide, para su análisis, en tiempo de inyección del fallo y tiempo de emulación del circuito con fallo. El tiempo de inyección del fallo supone el tiempo necesario para situar al circuito en el instante de inyección del fallo. Mientras que el tiempo de emulación con fallo es el tiempo necesario desde la inyección hasta el final de la emulación, esto es, hasta que el fallo se clasifica o se alcanza el final del banco de pruebas.

En la Figura 47 se muestra gráficamente el número de ciclos necesarios para emular todos los fallos posibles en un determinado biestable cuando no se incluye ninguna optimización en el sistema de inyección. En abscisas se representan los ciclos de reloj del banco de pruebas, mientras que en ordenadas se muestran los distintos fallos a inyectar, uno por entrada del banco de pruebas para la inyección exhaustiva. En el caso de no aplicarse optimizaciones del proceso de inyección, deben ejecutarse todos los ciclos del banco de pruebas.

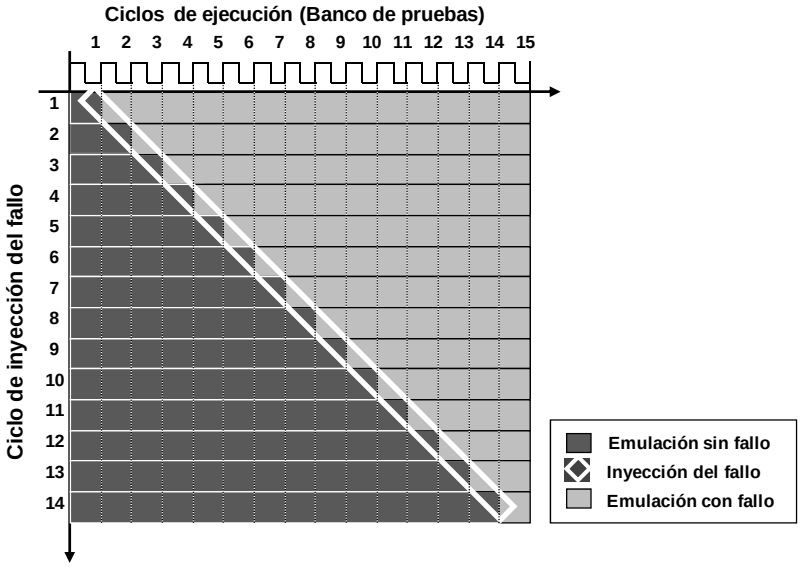


Figura 47. Emulación de todos los fallos posibles en un biestable determinado sin ninguna optimización

4.4.4.1.1 Mask-Scan

La inyección de los fallos se realiza de forma ordenada para optimizar la carga de la máscara de fallos. Se inyectan todos los fallos que pueden afectar a un biestable, después se desplaza la máscara de fallos para evaluar el siguiente biestable, y así

sucesivamente hasta finalizar el proceso de evaluación. En definitiva el tiempo de inyección total ($t_{inyección}$) es:

$$t_{inyección} = Ciclos_{inyección} \cdot T_{clock}$$

donde T_{clock} es el período de reloj del sistema de emulación.

Un biestable determinado puede verse afectado por un SEU en cualquiera de los instantes de ejecución, por lo que el tiempo de inyección relativo a la evaluación exhaustiva de un biestable es la suma de los ciclos necesarios para alcanzar el instante de inyección en cada fallo, esto es, (1+2+3+…+*C*). Para el circuito completo se tiene

$$\left.\begin{array}{l} Ciclos_{inyección} = \boldsymbol{F}\cdot(1+2+3+...+\boldsymbol{C}) \\ \sum_{i=0}^{n} i = \frac{n\cdot(n+1)}{2} \end{array}\right\} \Rightarrow Ciclos_{inyección} = \boldsymbol{F}\cdot\frac{\boldsymbol{C}\cdot(\boldsymbol{C}+1)}{2}$$

Por otro lado, el tiempo de emulación del fallo implica la emulación del circuito con el fallo insertado hasta que se detecta el fallo o hasta el final del banco de pruebas. Esto requiere, en el peor de los casos (si el fallo no ha sido detectado) $\boldsymbol{F}\cdot\boldsymbol{C}\cdot(\boldsymbol{C}+1)/2$ ciclos de reloj de la FPGA, en las mismas condiciones del párrafo anterior. En suma, los ciclos de reloj necesarios para emular el conjunto completo de fallos simples del circuito es:

$$Ciclos_{ejecución\_total} = F \cdot C\cdot(C+1)$$

En la Figura 48 se representa gráficamente el tiempo empleado en la emulación de los posibles fallos simples en un biestable determinado. Es necesario ejecutar el banco de pruebas completo para cada fallo a excepción de aquellos fallos que provocan una avería del circuito.

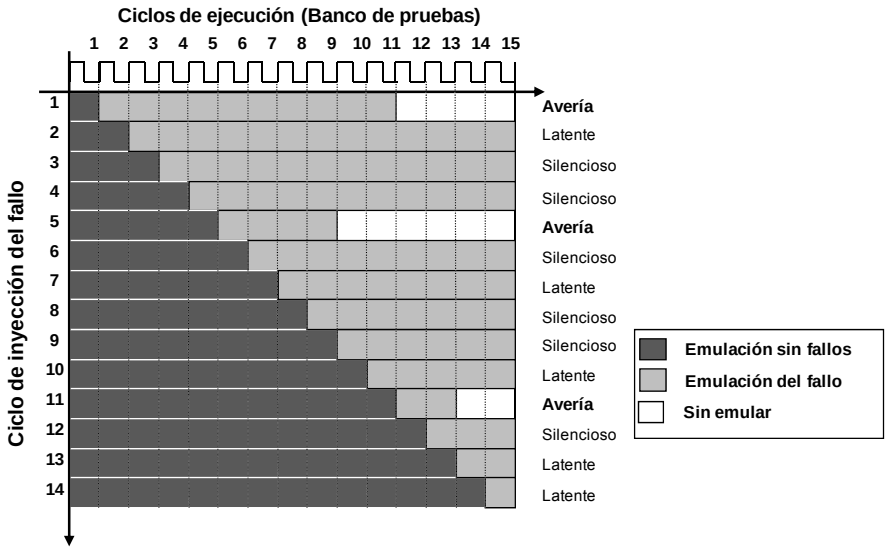


Figura 48. Emulación de los fallos posibles en un biestable determinado mediante la técnica *Mask-Scan*

Si se considerasen fallos múltiples, en el cálculo del tiempo total habría que añadir el tiempo empleado en introducir una máscara de fallos completa por la cadena de *scan*. Por ejemplo, para un número de fallos igual al anterior, *F·C*, con cada fallo se necesitan *C* ciclos para ejecutar el banco de pruebas y *F* ciclos para introducir la máscara de fallos completa, por lo que serían necesarios en el peor de los casos, *F·C·(C+F)* ciclos.

4.4.4.1.2 State-Scan

En este caso es necesario descargar, vía serie, el estado completo del circuito para cada fallo que se desee emular. Por tanto, los ciclos de reloj de la FPGA necesarios para completar la inyección de todo el conjunto de fallos simples del circuito es:

$$\text{Ciclos}_{\text{inyección}} = \#\text{fallos} \cdot \boldsymbol{F} = C \cdot \boldsymbol{F^2}$$

Con respecto al tiempo necesario para la emulación de fallos, es el mismo que el obtenido para la técnica anterior. Así, si el banco de pruebas puede aplicarse a la frecuencia del reloj de la FPGA (los estímulos están disponibles en la plataforma de emulación), en el peor de los casos se necesitarán F·C·(C+1)/2 ciclos de reloj para completar la emulación de todos los fallos.

$$\Rightarrow Ciclos_{ejecución\_total} = \boldsymbol{F \cdot C}\left(\boldsymbol{F} + \frac{\boldsymbol{C+1}}{2}\right)$$

El tiempo empleado en emular los fallos relativos a un biestable determinado se representan en la Figura 49. Se muestra una disminución del tiempo empleado en alcanzar el instante de inyección para los fallos inyectados en especial cuando el número de ciclos del banco de pruebas es mucho mayor que el número de biestables del circuito.

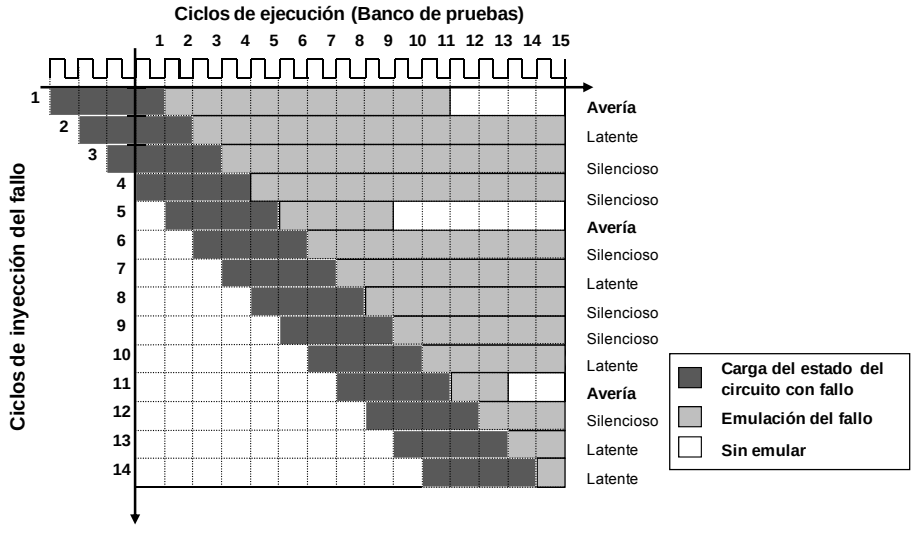


Figura 49. Emulación de los fallos posibles en un biestable determinado mediante la técnica State-Scan

4.4.4.1.3 Time-Multiplexed

La técnica *Time-Multiplexed* incluye optimizaciones tanto para llevar al circuito al instante de inyección como para acelerar la clasificación de fallos. Para estimar el efecto en el tiempo de emulación de un fallo que tiene una clasificación temprana de los fallos silenciosos, consideramos que de los $F \cdot C$ fallos a evaluar en una campaña de inyección exhaustiva S son fallos silenciosos. En la Figura 50 se muestra gráficamente el efecto de aplicar la técnica *Time-Multiplexed* en la ejecución del banco de pruebas. En este caso, el tiempo empleado en clasificar los fallos se reduce significativamente. Así como la restauración del estado, puesto que sólo requiere de un ciclo de reloj.

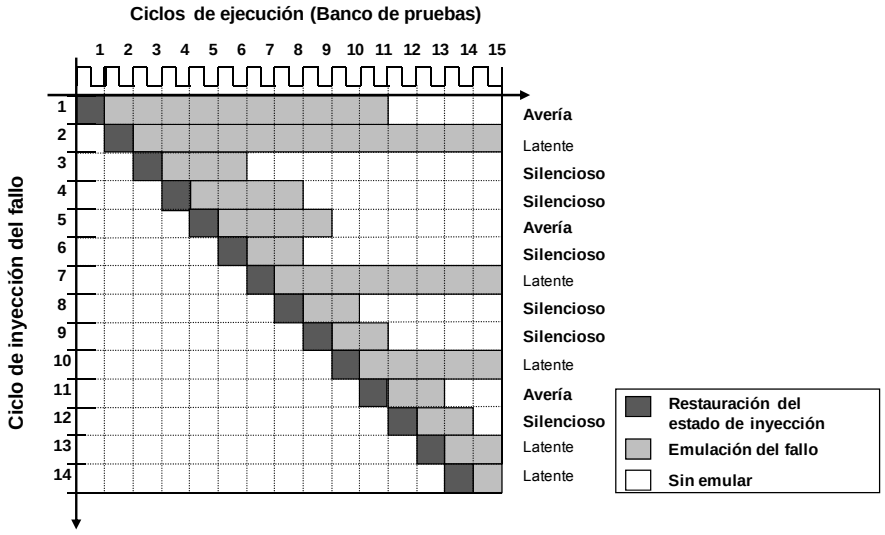


Figura 50. Emulación de los fallos posibles en un biestable determinado mediante la técnica *Time-Multiplexed*

Por lo tanto, respecto a la estimación del tiempo de inyección, tan sólo se consume un ciclo de reloj por fallo en la restauración del circuito:

$$Ciclos_{inyección} = (\#\ fallos) = F \cdot C$$

Por otro lado, puesto que se duplica la ejecución se duplica el tiempo de emulación requerido. En el peor de los casos (el fallo no se detecta):

$$Ciclos_{emulación} = 2 \cdot \boldsymbol{F} \frac{\boldsymbol{C}(\boldsymbol{C}+1)}{2} = \boldsymbol{F} \cdot C(\boldsymbol{C}+1)$$

Considerando que la detección temprana de los fallos silenciosos evita la emulación de K ciclos de ejecución en media, se tiene que el tiempo de emulación es el siguiente:

$$Ciclos_{emulación} = \boldsymbol{F} \cdot C(\boldsymbol{C}+1) - S \cdot K$$

En definitiva:

$$\Rightarrow Ciclos_{ejecución\_total} = \boldsymbol{F}\cdot C(\boldsymbol{C}+1) - S\cdot K + \boldsymbol{F}\cdot\boldsymbol{C} = \boldsymbol{F}\cdot\boldsymbol{C}\left(\boldsymbol{C}+\boldsymbol{2}\right) - S\cdot K$$

En un circuito tolerante a fallos se espera que un número considerable de fallos se corrijan o enmascaren rápidamente mediante los mecanismos de corrección de errores y las estructuras para enmascarar fallos presentes en el circuito. Por lo tanto, se estima que un porcentaje notable de los fallos inyectados son silenciosos y que su detección se realizará en unos pocos ciclos de reloj, por lo que el término *S·K* es de una magnitud notable.

4.4.4.1.4 Comparación

La Tabla 1 recoge el tiempo teórico empleado por cada técnica en la emulación de todos los fallos simples del circuito (medido en ciclos de reloj). Este tiempo aparece desglosado en ciclos de reloj para la inyección y para la emulación.

	Mask-Scan	***State-Scan***	***Time-Multiplexed***
$Ciclos_{inyección}$	$\boldsymbol{F}\cdot\frac{\boldsymbol{C}\cdot(\boldsymbol{C}+1)}{2}$	$C\cdot F^2$	$C\cdot F$
$Ciclos_{emulación}$	$\boldsymbol{F}\cdot\frac{\boldsymbol{C}\cdot(\boldsymbol{C}+1)}{2}$	$\boldsymbol{F}\cdot\frac{\boldsymbol{C}\cdot(\boldsymbol{C}+1)}{2}$	$C\cdot F(C+1\ ) - S\cdot K$
$Ciclos_{total}$	$F\cdot C\cdot(C+1)$ $\approx F\cdot C^2$	$\boldsymbol{F}\cdot\boldsymbol{C}\left(\boldsymbol{F}+\frac{\boldsymbol{C}+\boldsymbol{1}}{2}\right)$	$C\cdot F(C+2) - S\cdot K$ $\approx F\cdot C^2 - S\cdot K$

Tabla 1. Comparación teórica de los ciclos necesarios en la aplicación de cada técnica

Los tiempos empleados por las técnicas *Mask-Scan* y *Time-Multiplexed* se pueden simplificar cuando $\boldsymbol{C} >> 1$, condición que se cumple en cualquier circuito real. En ese caso, ambas técnicas implican un tiempo de ejecución total diferente únicamente en el término relativo a la detección de fallos silenciosos. Si el número de fallos silenciosos de un circuito fuese despreciable ambas técnicas emplearían el mismo tiempo en realizar una campaña de inyección. Por el contrario, si el número de fallos silenciosos es alto la técnica *Time-Multiplexed* proporciona una mayor aceleración al proceso de evaluación de la tolerancia a fallos.

En el caso de la técnica *State-Scan* la dependencia con el número de biestables del circuito $\boldsymbol{F}$ y con el número de ciclos de reloj del banco de pruebas $\boldsymbol{C}$ es distinta a las otras dos implementaciones. Para estudiar esta diferencia representamos gráficamente, Figura 51 y Figura 52, el tiempo de emulación empleado con la técnica *Mask-Scan* (igual al empleado por la solución *Time-Multiplexed* cuando no se considera el efecto de la detección rápida de los fallos silenciosos) y la técnica de *State-Scan.*

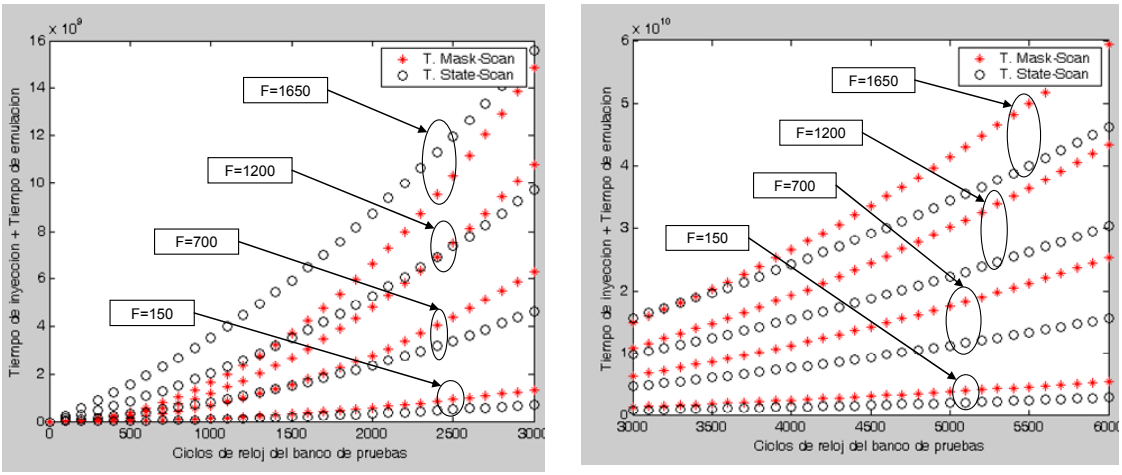


Figura 51. Representación gráfica de la dependencia del tiempo de emulación con ***C*** usando como parámetro el número de biestables, ***F***

Se observa como para un sistema dado, la técnica *State-Scan* es más rápida que la técnica *Mask-Scan* cuando el número de ciclos del banco de pruebas es grande. A medida que el banco de pruebas es más largo el tiempo de la técnica *Mask-Scan* crece, de tal forma que para ***C*** muy grande la técnica *State-Scan* es la más rápida.

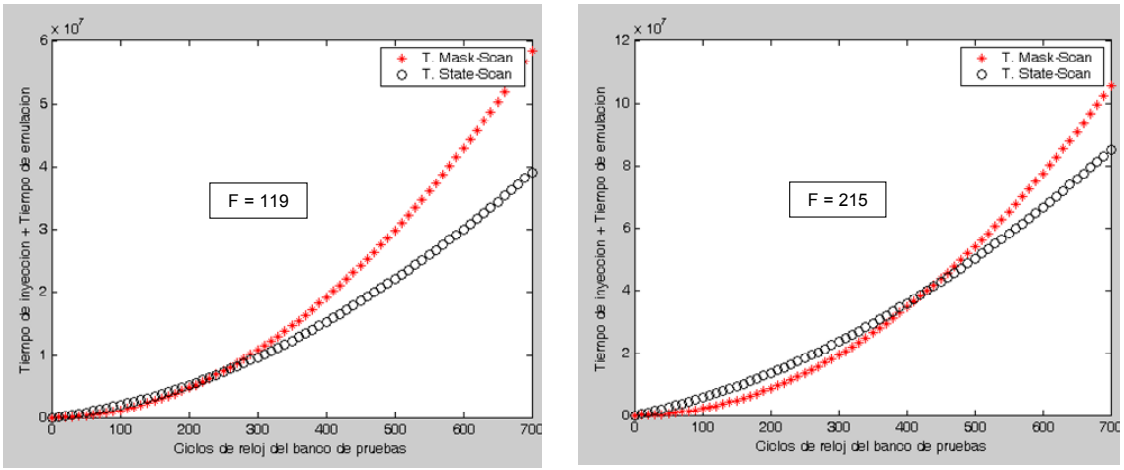


Figura 52. Comparación gráfica del tiempo de emulación empleado en distintas técnicas para dos sistemas particulares

Para un número dado de biestables los tiempos de ejecución de la campaña de inyección de fallos simples son iguales si se cumple la siguiente relación:

$$\text{Ciclos}_{\text{Tecnica Mask-Scan}} = F \cdot C^2$$

$$\text{Ciclos}_{\text{Tecnica State-Scan}} = F \cdot C \cdot \left(\frac{C+1}{2} + F \right)$$

$$\left.\begin{aligned} \text{Ciclos}_{\text{Tecnica de Mask-Scan}} &= \boldsymbol{F} \cdot \boldsymbol{C}^2 \\ \text{Ciclos}_{\text{Tecnica State-Scan}} &= \boldsymbol{F} \cdot \boldsymbol{C} \cdot \left(\frac{\boldsymbol{C}+1}{2} + \boldsymbol{F} \right) \end{aligned}\right\} \boldsymbol{F} \cdot \boldsymbol{C}^2 = \boldsymbol{F} \cdot \boldsymbol{C} \cdot \left(\frac{\boldsymbol{C}+1}{2} + \boldsymbol{F} \right) \Rightarrow \boldsymbol{C} = 2 \cdot \boldsymbol{F}$$

$$\Rightarrow \begin{cases} C < 2 \cdot F & \Rightarrow \text{Técnica de Mask} - \text{Scan es más rápida} \\ C > 2 \cdot F & \Rightarrow \text{Técnica de State} - \text{Scan es más rápida} \end{cases}$$

Por lo tanto, dependiendo del sistema y de su banco de pruebas será más adecuado aplicar una técnica u otra. Teniendo en cuenta el efecto de la detección de fallos silenciosos implementada en la técnica *Time-Multiplexed* puede obtenerse una gran mejora si el número de fallos silenciosos es alto. Este factor es de notable importancia, si tenemos en cuenta que experimentos realizados en circuitos reales muestran que hay un elevado porcentaje de fallos silenciosos. Este porcentaje puede ser de hasta el 50% de los fallos en circuitos sin protección, y mayor aún en circuitos tolerantes a fallos. La principal ventaja de la técnica de *Mask-Scan* frente a la *Time-Multiplexed* es que aquella requiere de una modificación del circuito menor y que el módulo de control es más sencillo.

4.5 Sistemas con memorias empotradas

Los circuitos digitales actuales incluyen habitualmente memorias integradas. Las memorias SRAM son sensibles a SEUs de igual forma que los biestables. Por lo tanto, debe evaluarse la tolerancia del circuito también cuando las memorias integradas se ven afectadas por un *bit-flip*. Los estudios en la literatura sobre inyección de fallos en memorias en emulación (capítulo 3) son escasos debido a la dificultad que conlleva [Cive01b] [Lima01a]. En [Cive01b] se manipulan las señales de control y los buses de datos y direcciones de la memoria, para inyectar fallos cuando corresponde, pero no se propone ninguna solución para analizar los fallos dentro de la memoria. Por otro lado, en [Lima01a] se presenta un modelo de memoria que consiste en una memoria de doble puerto. Uno de los puertos se utiliza para la ejecución normal mientras que el otro se usa para inyectar fallos. Sin embargo, y debido a la limitada observabilidad que proporcionan las memorias, el análisis de resultados consiste en leer todo el bloque de memoria, comparando los datos con los esperados sin fallo, lo que es un proceso muy lento.

La Emulación Autónoma acelera el proceso de inyección gracias a la minimización de la comunicación entre el PC y la FPGA durante la campaña de inyección, y a las optimizaciones propuestas que se incluyen. Para aplicar un entorno de Emulación Autónoma es necesario considerar y resolver el problema de la inyección de fallos y su emulación en memorias empotradas.

En el caso de los biestables es necesario un instrumento que sustituye a cada uno de los biestables originales del circuito y proporciona las capacidades de inyección y observación requeridas. De igual forma, es necesario proponer un modelo para instrumentar los bloques de memoria empotrados, proporcionando al circuito las capacidades necesarias para soportar la Emulación Autónoma de fallos. La memorias

podrían emularse si se fuerza al sintetizador a implementarlas como biestables. Sin embargo, para ello sería necesaria una cantidad de recursos que no está disponible en las FPGA comerciales. La mejor solución es proponer un modelo específico para bloques de memoria. Para ello, se hacen las siguientes asunciones:

- La memoria empotrada es síncrona, de forma que se puede prototipar en la FPGA con los bloques de memoria disponible en las FPGAs actuales.
- Sólo se inyectan fallos simples. El objetivo es la evaluación de la tolerancia a fallos de un circuito frente a SEUs, considerando que los fallos afectan a una única celda de memoria.
- El modelo debe soportar el sistema de Emulación Autónoma. Como se vio en el apartado 4.4 la técnica *Time-Multiplexed,* propuesta para implementar un sistema de Emulación Autónoma, integra el mayor número de mejoras y optimizaciones y proporciona una aceleración de la velocidad del proceso de inyección mayor. Las otras dos soluciones son simplificaciones de aquella con el objetivo de reducir el área necesaria para su implementación. Por tanto, el modelo se basa en la técnica *Time-Multiplexed*.
- La memoria no almacena información válida antes de comenzar la emulación. Todos los datos que se leen han sido previamente escritos durante la ejecución.

A continuación se describe el modelo de memoria propuesto y dos posibles implementaciones que contemplan distintos niveles de precisión y de recursos necesarios. El estudio de la inyección de fallos en memorias presentado es una aportación original de esta tesis doctoral.

4.5.1 Modelo de memoria propuesto y desarrollado

El sistema de Emulación Autónoma implementado con multiplexación en el tiempo requiere la modificación del circuito para que éste soporte las siguientes funcionalidades:

- Emulación tanto del estado con fallo, *faulty*, como del estado sin fallo o *golden*.
- Comparación de ambos estados en cada ciclo de reloj, activando una señal de error cuando el estado *faulty* difiere del *golden*.
- Restauración del estado previo al instante de inyección para evitar la emulación del banco de pruebas sin fallo.

La observabilidad y controlabilidad es total en los biestables. Sin embargo, en las memorias sólo es posible acceder a una palabra de memoria en cada ciclo de reloj por lo que su observabilidad y controlabilidad están limitadas, lo que dificulta la inyección de fallos y sobre todo la observación de los efectos que tiene el fallo insertado en el circuito.

La solución que se propone consiste en observar y controlar las señales de acceso a la memoria, en lugar de comprobar el contenido de la misma. De esta forma, se detecta cuando se introduce un fallo en la memoria y se registra, lo que permite al bloque de control de la inyección conocer en cada instante la presencia o no de errores en memoria sin necesidad de recorrerla para leer todos los datos.

La Figura 53 muestra el esquema del modelo propuesto para el instrumento de la memoria empotrada. Por simplicidad, se representa el caso de una memoria con un único puerto.

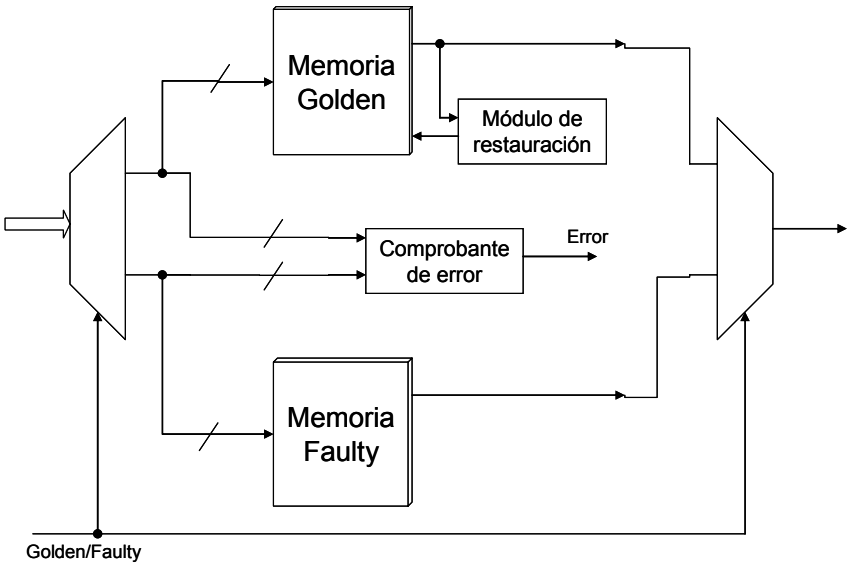


Figura 53. Modelo del instrumento para la memoria empotrada.

Los componentes que conforman el modelo son los siguientes:

- *Memoria Golden*. Es memoria que se corresponde con la ejecución correcta (sin fallo) del circuito.
- *Memoria Faulty*. Bloque utilizado para almacenar los datos erróneos durante la ejecución con fallo.
- *Comprobante de error*. Se encarga de comparar los accesos a memoria para detectar la escritura de un error en la memoria. Mientras los accesos en el ciclo *golden* coincidan con los accesos en el ciclo *faulty*, se considera que la memoria no está afectada por ningún error.

- *Módulo de restauración*. Se encarga de restaurar el contenido de la memoria antes del instante de inyección.

Se explica a continuación cómo se realizan las funciones necesarias en una campaña de inyección de fallos mediante *Time-Multiplexed*, utilizando el modelo de memoria propuesto. Como se ha dicho anteriormente en este apartado, dichas tareas son la emulación del circuito con y sin fallo, la comparación en cada instante de ambas ejecuciones y la restauración del estado previo a la inyección.

4.5.1.1 Emulación del circuito sin fallo

La ejecución *golden* se realiza con el bloque *Memoria Golden*. Este bloque de memoria almacena los valores correctos a lo largo de la emulación. Durante el ciclo de ejecución *golden* los accesos a memoria, tanto de lectura como de escritura, se realizan sobre esta memoria. Es decir, esta memoria coincide con el bloque de memoria del circuito antes de ser instrumentado y se comporta de la misma manera.

4.5.1.2 Emulación del circuito con fallo

La técnica de emulación considerada es la técnica *Time-Multiplexed*, de forma que se lleva a cabo un ciclo de ejecución sin fallo y a continuación el mismo ciclo de la carga de trabajo, pero con fallo. El bloque denominado *Memoria Faulty* se utiliza para almacenar los valores de la memoria afectados por el fallo. La forma más sencilla de implementar esta funcionalidad consiste en duplicar directamente el bloque de memoria, de manera, que durante el ciclo de ejecución *Faulty* todos los accesos a la memoria, tanto de lectura como de escritura, se realizan al bloque *Memoria Faulty*. Esta implementación se ha denominado modelo de memoria básico.

Otra implementación de la función de emulación con fallo consiste en almacenar únicamente las palabras de memoria que difieren del contenido *golden*. Entonces, sólo se accede al bloque *Memoria Faulty* para escribir palabras de memoria con errores o para leer posteriormente su contenido. Puesto que el acceso a dicha memoria debe realizarse en un único ciclo de reloj, la opción más adecuada es su implementación en una memoria CAM (*Content Addressable Memory*). El bloque de memoria *Faulty* debe disponer de la dirección de memoria que contiene el fallo así como del dato correspondiente. Cuando un nuevo error se detecta, la dirección y el dato se almacenan en uno de las posiciones vacías de *Memoria Faulty*. Debido al tipo de implementación este modelo se ha denominado modelo de memoria ECAM (*Error Content Addressable Memory*). En el ciclo de ejecución *Faulty* se accede a *Memoria Faulty* para comprobar si la palabra a la que se quiere acceder se encuentra almacenada en dicha memoria. Si el dato se encuentra en el bloque *Memoria Faulty* significa que el dato contiene un fallo. En caso contrario, el acceso debe realizarse al bloque *Memoria Golden*.

El tamaño de *Memoria Faulty* fijado determina el número máximo de errores que pueden considerarse. Se asume la hipótesis de que el circuito no puede recuperarse de un número de errores *N*, siendo *N* el número de palabras de la memoria *Faulty*. Por lo tanto, la elección de *N* debe realizarse alcanzando un compromiso entre los recursos necesarios y la precisión deseada. En la práctica, la probabilidad de que se cancelen por completo los errores de la memoria disminuye al aumentar el número de errores, y se estima que *N* es un valor relativamente pequeño en la práctica.

4.5.1.3 Clasificación de fallos

La observación de los efectos de un fallo en la memoria empotrada se basa en comparar los accesos a la memoria en el ciclo de ejecución *golden* con los correspondientes en el ciclo *faulty*, utilizando para ello el bloque *Comprobante de error*. Este bloque indica si se introduce un error en la memoria o por el contrario si un error ya existente se sobrescribe y por tanto se cancela. La comprobación se realiza en función de los valores del bus de dirección, el bus de datos y las señales de control de la memoria que habilitan la escritura, lectura o la propia memoria. Un error se produce en los siguientes casos:

- Sólo se escribe en la memoria en el ciclo *Golden* o sólo se escribe en la memoria en el ciclo *Faulty*. Se produce un nuevo error cuando la dirección a modificar no contenía fallo y el dato que se escribe difiere del ya almacenado.
- La memoria se escribe en ambos ciclos, *golden* y *faulty*, pero las direcciones de memoria a modificar son distintas. En este caso, pueden producirse dos errores, uno porque la dirección correcta no recibe el dato y otro porque otra posición de memoria modifica su contenido.
- La memoria se escribe en ambos ciclos, *golden* y *faulty*, pero el dato introducido difiere. Se introduce un nuevo error si la posición escrita no contenía ya un error.

Cuando, en el ciclo *faulty*, se escribe en una dirección de memoria que contiene un error, dicha escritura puede sobrescribir el fallo y por lo tanto cancelarlo. Esto sucede en los siguientes casos:

- Sólo se escribe en la memoria en el ciclo *Golden*. El error se cancela si la dirección contenía un error y el dato escrito coincide con el dato almacenado en la memoria *Faulty*.
- Sólo se escribe en la memoria en el ciclo *Faulty*. El error se cancela si la dirección contenía un error y el dato escrito coincide con el almacenado en la memoria *golden*.

- La memoria se escribe en ambos ciclos de ejecución, en distintas posiciones. Pueden cancelarse dos errores, el correspondiente a la dirección escrita en el ciclo *golden* y el debido a la escritura en el ciclo *faulty*.
- La memoria se escribe en ambos ciclos, en la misma dirección y con el mismo dato.

Para poder considerar la cancelación de un error en memoria se requiere conocer su posición. En el caso del modelo ECAM es sencillo, puesto que el bloque *Memoria Faulty* almacena tanto la dirección como el dato con fallo. Para el modelo básico, sería necesario añadir un registro de direcciones adicional, en caso contrario no es posible detectar la cancelación de un fallo en un único ciclo de reloj. Por lo tanto, la implementación elegida para la memoria con fallo determina la precisión obtenida en la clasificación de fallos. En el capítulo 6 se estudian las características de ambos modelos en función de los recursos necesarios y la precisión en los resultados ofrecida.

4.5.1.4 Restauración del estado

La restauración del estado para biestables se realiza en un único ciclo de reloj cuando la técnica utilizada es *Time-Multiplexed*. Esto supone un ahorro sustancial del tiempo empleado en la emulación de un fallo. Cuando el circuito contiene memorias, el proceso de restauración se complica debido a la limitada accesibilidad que presentan, ya que sólo puede escribirse una palabra por ciclo de reloj. La solución aplicada para biestables consiste en duplicar el elemento de memoria, de forma que un biestable adicional almacena el estado previo a la inyección y que será cargado en el biestable utilizado para la ejecución cuando sea necesario. En el caso de una memoria, serían necesarios tantos ciclos de reloj como palabras contiene dicho bloque para realizar la restauración. La memoria disponible en un circuito actual puede contener un número de palabras comparable o del orden del tamaño del ciclo de pruebas. Es evidente, que adoptar la solución aplicada a los biestables para el caso de memorias no resulta provechoso en el caso general.

En la arquitectura que se propone se realizan las siguientes hipótesis:

- Para una aplicación dada, en el caso general, se hace uso únicamente de una parte de la memoria disponible.
- Las escrituras realizadas en la memoria suponen un número de ciclos de reloj mucho menor que el número total de ciclos del banco de pruebas.

En el peor de los casos, se realizaría una escritura en memoria en cada ciclo de reloj. En ese caso, restaurar la memoria en el instante t_i requiere un tiempo t_i. Sin embargo, no restaurar el estado, supone la emulación del banco de pruebas previo al

instante de inyección. Teniendo en cuenta que se utiliza la técnica *Time-Multiplexed*, ese proceso consume $2 \cdot t_i$, ya que se realiza una ejecución con fallo y otra sin fallo.

Teniendo en cuenta las consideraciones anteriores, se propone restaurar el estado de las memorias escribiendo únicamente aquellas posiciones que fueron modificadas durante la emulación del fallo anterior en los instantes previos a la inyección del siguiente fallo. Durante la emulación de un fallo la *Memoria Golden* se va modificando en función de la ejecución del circuito sin fallo. Cuando un nuevo valor se escribe en dicha memoria el contenido previo a la escritura se almacena en el *Módulo de restauración*, así como el valor del vector de dirección. Si en una misma dirección se escribe varias veces sólo el dato original se guarda en el *Módulo de Restauración*, de forma que existe una posición en la memoria de restauración para cada dirección modificada, independientemente del número de veces que se haya escrito.

La inyección de fallos se realiza de forma ordenada en el tiempo. Para un elemento de memoria dado se evalúan fallos simples siguiendo un orden creciente con respecto al instante de inyección. En caso de una inyección exhaustiva, si un fallo se inyecta en el instante t_i en un elemento de memoria determinado, el siguiente fallo se insertará en el instante t_{i+1}. La Figura 54 representa parte de la secuencia de ejecución de una campaña de inyección indicando cómo se modifica el *Módulo de Restauración*.

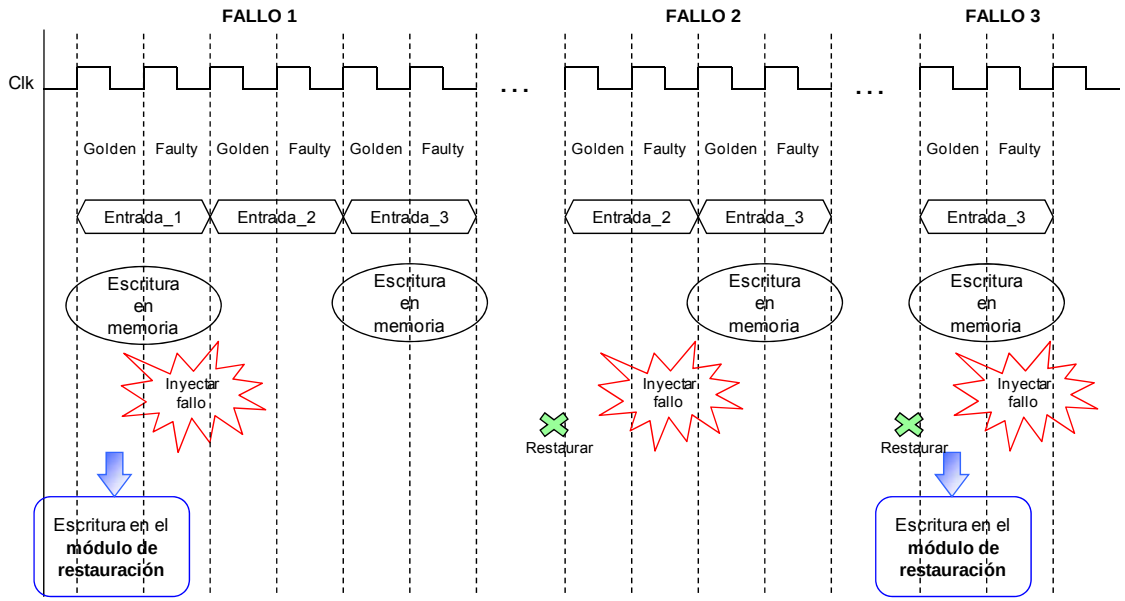


Figura 54. Ejemplo de cómo se realiza la escritura incremental del módulo de restauración.

En el ejemplo mostrado en la figura, el primer fallo se inyecta en el primer ciclo del banco de pruebas. En el primer ciclo de ejecución, se realiza una escritura en la memoria por lo que el valor modificado se guarda en el *Módulo de Restauración*. Cuando se finaliza la evaluación del fallo, se restaura la memoria con el dato que se había escrito en el *Módulo de Restauración* y así puede comenzarse la ejecución con la

siguiente entrada del banco de pruebas. En el ejemplo que se muestra en la Figura 54, antes de la inyección del segundo fallo no se produce ninguna escritura en memoria por lo que no es necesario modificar el *Módulo de Restauración*. La figura ilustra el carácter incremental de dicho módulo, debido al aumento de modificaciones realizadas en la memoria en los distintos ciclos.

Después de la emulación de un fallo, el *Módulo de Restauración*, contiene los datos originales de las direcciones que se modificaron durante la ejecución, previamente al instante de inyección del siguiente fallo. Para restaurar el estado previo a la inyección se escriben estos datos en la memoria *golden.* Son necesarias tantas escrituras como direcciones distintas se modificaron en la ejecución sin fallo.

Utilizando el modelo básico, la restauración debe también aplicarse a la *Memoria Faulty*, lo que se realiza de forma concurrente a la tarea de restauración de la *M*emoria *Golden* sin suponer un retraso en la evaluación. Utilizando el modelo ECAM, la *Memoria Faulty* simplemente debe borrarse, para eliminar todos los fallos almacenados durante la emulación del anterior fallo. La eliminación del contenido de la memoria con fallo se realiza en un único ciclo de reloj, añadiendo una etiqueta de un bit para indicar si la palabra correspondiente está ocupada o por el contrario con contiene datos válidos. Estas etiquetas son biestables y por lo tanto pueden escribirse en un solo ciclo de reloj.

4.5.1.5 Inyección en memorias empotradas

La inyección en una memoria empotrada de acuerdo con el modelo de fallo *bit-flip* requiere la modificación de uno de sus bits. El fallo queda fijado en función de la dirección de memoria afectada, la posición del bit y el instante de inyección.

El número de posibles fallos simples en una memoria es enorme. Sin embargo, pueden conocerse los efectos de muchos de esos fallos sin necesidad de realizar la emulación. Para la preclasificación de los efectos de un fallo en memoria se aplican las siguientes reglas:

- Un fallo es latente cuando se inyecta en una palabra de memoria que no se lee a lo largo del banco de pruebas.
- Un fallo es silencioso cuando se inyecta en una palabra de memoria en la que después del instante de inyección el acceso que se realiza es una escritura.
- Los fallos insertados en cualquiera de los ciclos de reloj comprendidos entre un acceso, de escritura o lectura, y otro de lectura tiene una clasificación equivalente, por lo que es suficiente con inyectar uno de ellos. Esta regla está directamente relacionada con la colapsación dinámica de fallos, optimización de la emulación que se propone y describe en el apartado 4.3.3.

Sólo es interesante inyectar fallos en las palabras de memoria que se leen y en los instantes de inyección anteriores a la lectura. Para realizar la inyección únicamente en las direcciones leídas se propone modificar el bus de datos de salida de la memoria aplicando la misma lógica de inyección utilizada en la técnica *Time-Multiplexed* para los biestables, propuesta en [Cive01a]. La Figura 55 muestra el esquema de la lógica necesaria para realizar la inyección de fallos en memorias.

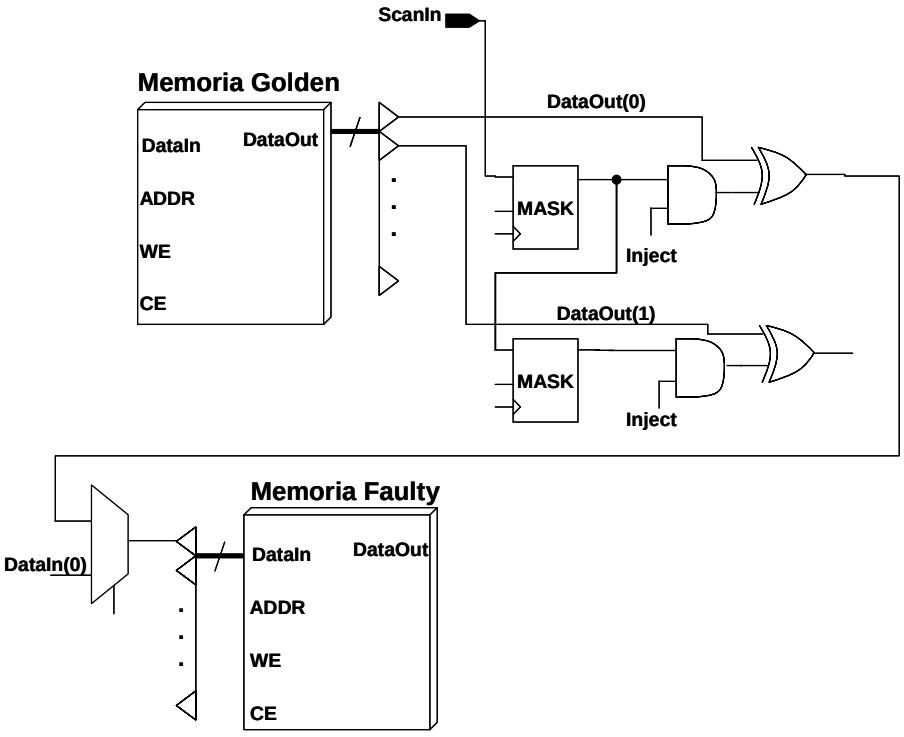


Figura 55. Esquema de la lógica de inyección de fallos en memorias.

La lógica de inyección se añade al bus de datos de salida de la memoria *golden* para modificar únicamente los datos que se leen, lo que es una aportación original de esta tesis. Para su implementación se añade un biestable de máscara por cada bit del bus de datos de salida. El bit indicado se modifica cuando se activan las señales de control correspondientes a la inyección, y el dato con fallo se escribe en la *Memoria Faulty*.

4.6 Resumen y conclusiones

La emulación de fallos con FPGA es una solución rápida y eficiente para la evaluación de la tolerancia a fallos. En las técnicas de emulación propuestas por otros autores la mayor limitación es la impuesta por la comunicación intensiva entre el PC y la FPGA, necesaria para controlar la campaña de inyección. En este capítulo se ha presentado un sistema de inyección, denominado Emulación Autónoma, cuya principal característica es que minimiza notablemente la necesidad de comunicación entre PC y FPGA. De hecho, aplicando este sistema de inyección la comunicación sólo se establece al principio del proceso para configurar el sistema y al final del mismo para recoger

todos los resultados obtenidos, esto es principalmente, el diccionario de fallos. Es decir, la campaña de inyección se realiza de forma autónoma sin necesidad del control por parte del PC.

Una vez reducida la interacción PC-FPGA es posible incluir ciertas funcionalidades en el propio sistema de inyección que facilitan la observación del comportamiento del circuito y aceleran el proceso de inyección. Las optimizaciones propuestas son tres. Dos de ellas tienen como objetivo la reducción de tiempo necesario para la clasificación del fallo insertado:

- Detección rápida de fallos silenciosos.
- Colapsación dinámica de la lista de fallos.

La tercera mejora, la restauración del estado previo al instante de inyección, reduce el tiempo empleado en la ejecución del circuito antes de la inyección del fallo, puesto que ese funcionamiento es conocido para un banco de pruebas dado.

Se han descrito tres posibles implementaciones para un sistema de Emulación Autónoma, considerando distintos niveles de optimización. La técnica que soporta la inclusión de todas las mejoras propuestas, *Time-Multiplexed*, se basa en la ejecución alternada del circuito sin y con fallo. Las otras dos técnicas, *State-Scan* y *Mask-Scan*, son simplificaciones de la primera. Incluyen menos optimizaciones, por lo que se aumenta el tiempo necesario para realizar una campaña de inyección, pero se reducen los recursos *hardware* necesarios para su implementación. Los resultados experimentales obtenidos con las distintas implementaciones, que se presentan en el capítulo 6, permiten estudiar la mejora lograda con cada una de las optimizaciones propuestas.

Se ha realizado una comparación teórica, bajo ciertas asunciones, sobre el tiempo empleado en cada técnica para la ejecución de la campaña de inyección de fallos, con objeto de prever y estimar cuál es más adecuada. El resultado indica que la velocidad de las distintas técnicas depende de las características del circuito y su banco de pruebas. Además, en la elección de una técnica frente a otra influyen otros factores, aparte de la velocidad, como los recursos *hardware* necesarios. Según estas estimaciones, las técnicas *Mask-Scan* y *Time-Multiplexed* ofrecen prestaciones similares cuando no se considera la aceleración obtenida mediante una detección rápida de fallos silenciosos. En caso contrario, la técnica *Time-Multiplexed* consigue mayores tasas de inyección de fallos cuanto mayor es el porcentaje de los fallos inyectados que se clasifican como silenciosos. Se espera que los circuitos evaluados, al disponer de técnicas de tolerancia a fallos, provoquen un gran número de fallos silenciosos, por lo que una clasificación temprana de éstos acelerara notablemente del proceso de inyección, como demuestran los resultados experimentales que se presentan en el

capítulo 6. Por otra parte, la técnica *State-Scan* es más rápida que la *Mask-Scan* a medida que aumenta el número de vectores del banco de pruebas. El modelo de Emulación Autónoma y sus implementaciones son aportaciones originales de esta tesis.

Por otro lado, este capítulo presenta un modelo para memorias empotradas que soporta la Emulación Autónoma y en especial la implementación *Time-Multiplexed*. El modelo propuesto permite la emulación del circuito con (*faulty*) o sin fallo (*golden*), de forma alternativa, inyectando fallos tanto en los biestables como en los bloques de memoria. Ambas ejecuciones (*faulty* y *golden*) se comparan en cada instante de la ejecución para detectar cuándo un fallo desaparece y se puede clasificar como silencioso. Para detectar la aparición de un error en el contenido de la memoria, se observan los accesos a memoria en la ejecución con y sin fallo, de forma que se evita tener que recorrer toda la memoria, lo que es un proceso lento. Por tanto, el modelo de memoria propuesto permite aplicar en los circuitos con memorias empotradas las optimizaciones propuestas para clasificar los fallos y restaurar el estado previo a la inyección.

Las memorias requieren una consideración especial para, teniendo en cuenta su limitada accesibilidad, mantener las optimizaciones en velocidad propuestas para un entorno de inyección basado en Emulación Autónoma. La solución descrita consiste en basar la observación del efecto del fallo, no en el contenido de la memoria, sino en las señales de control y acceso a la misma a lo largo de la ejecución. De esta forma, es posible detectar la aparición o cancelación de un error en un ciclo de reloj, sin necesidad de leer todas las palabras de memoria. Esta función se desempeña a expensas de incluir *hardware* adicional, como sucede también para los biestables. Se proponen dos posibles implementaciones del bloque de memoria utilizado para la ejecución con fallo. El modelo básico consiste en duplicar el bloque de memoria, mientras que el modelo ECAM almacena únicamente los valores con fallo, así como las direcciones donde están almacenados dichos datos. Ambos modelos implican distintos requisitos de *hardware* para su implementación. El modelo básico requiere menos lógica que el modelo ECAM puesto que este último implica cierta lógica de control adicional, por otra parte el modelo básico necesita más memoria disponible que el modelo ECAM puesto que consiste en duplicar la memoria a emular. El modelo de memoria propuesto y sus implementaciones son aportaciones originales de este trabajo de tesis doctoral.

Un estudio experimental de las características de las distintas propuestas realizadas permitirá evaluar las ventajas e inconvenientes de esta solución. Dicho estudio se presenta en el capítulo 6.

CAPÍTULO 5

INYECCIÓN DE FALLOS SEU EN MICROPROCESADORES

5. Inyección de Fallos SEU en Microprocesadores

En este capítulo se propone un entorno de inyección de fallos transitorios orientado a evaluar la tolerancia a SEUs de circuitos microprocesadores sobre un componente comercial. El objetivo es proponer un sistema capaz de inyectar fallos en diferentes microprocesadores de forma rápida, siendo lo más general posible y de bajo coste. La solución que se presenta en este capítulo, y que es una aportación original de este trabajo de tesis, se basa en las infraestructuras de depuración integradas en los microprocesadores actuales para acceder a los recursos internos del mismo y realizar la inyección de fallos y la observación de sus efectos. El sistema de inyección propuesto, al igual que en el sistema de inyección de Emulación Autónoma descrito en el capítulo 4, se ejecuta por completo en *hardware* para minimizar el intercambio de información necesaria con el PC y acelerar la ejecución de las distintas tareas de inyección.

5.1 Introducción

Los microprocesadores son componentes principales de los sistemas digitales actuales. Desde la aparición de los primeros procesadores, éstos se han aplicado al control de tareas críticas y en la actualidad forman parte de cada vez más diversas aplicaciones. La evaluación de la confiabilidad de circuitos microprocesadores es, por tanto, una tarea fundamental para asegurar el comportamiento adecuado del sistema en presencia de fallos. En origen, el desarrollo de técnicas de inyección y el estudio de la tolerancia a fallos de los sistemas digitales se centraba en estos circuitos por lo que los sistemas de inyección orientados a procesadores han sido ampliamente estudiados [Prad96]. Sin embargo, los avances en los diseños cada vez más complejos, con la inclusión de cada vez más capacidades en los circuitos proporcionan nuevos mecanismos para la inyección de fallos.

Como se vio en el capítulo 3, las técnicas de inyección pueden agruparse en dos categorías diferentes dependiendo de si la inyección se realiza sobre una descripción del circuito o, por el contrario, los fallos se insertan en un componente comercial. El capítulo 4 describe y propone un entorno de inyección basado en la emulación *hardware* de fallos de una descripción del circuito, el sistema de Emulación Autónoma. En caso de no disponer de dicha descripción o bien que sea necesario evaluar la implementación final del circuito, no es posible aplicar la Emulación Autónoma, y sólo pueden aplicarse técnicas de inyección sobre *hardware*, como generalmente sucede con los microprocesadores y otros circuitos IP. Dentro de las técnicas de inyección propuestas hasta el momento la más realista es la radiación con iones pesados, como se vio en el capítulo 3. Sin embargo, este método requiere un alto coste por lo que sólo se aplica en las últimas etapas del desarrollo de un circuito, para obtener medidas finales de confiabilidad y certificar el circuito. Para realizar evaluaciones preliminares de la

tolerancia a fallos en un componente comercial son necesarias técnicas de inyección de bajo coste como las técnicas *software* o las basadas en las capacidades de depuración disponibles en los microprocesadores actuales. Las técnicas *software* son rápidas puesto que la inyección consiste en ejecutar unas pocas instrucciones [Carr98][Vela00] (capítulo 3). Sin embargo, requieren modificar el código original por lo que son intrusivas y no resuelven de forma general el problema de la observación de los efectos. Las técnicas basadas en las capacidades de depuración [Folk98][Peng06][Fida06] no requieren modificar el código del procesador y permiten un acceso fácil y rápido a los recursos internos del circuito, mediante recursos disponibles en el propio microprocesador a estudiar. Además, hoy en día, las capacidades de depuración integradas en los CIs (microprocesadores, DSPs, etc.) son cada vez más importantes para facilitar el diseño de sistemas digitales complejos como SoCs, SoPCs (*System on Programmable Chip*) o sistemas empotrados dónde, de otra forma, acceder a los recursos internos del circuito sería una tarea complicada.

El sistema de inyección que se propone y describe en este capítulo aplica el concepto de sistema de inyección autónomo propuesto en el capítulo anterior, implementando el entorno de inyección en una plataforma *hardware* con el objetivo de paralelizar y acelerar las tareas de inyección. La solución propuesta utiliza las infraestructuras de depuración integradas en los microprocesadores actuales para acceder a los recursos internos del procesador y así realizar la inyección y observar los efectos originados. Un caso especial es el de los microprocesadores integrados en SoPCs. En estos casos, el sistema dispone de capacidades especiales propias de estos dispositivos como es la lógica programable presente en el sistema y que puede utilizarse para implementar ciertas optimizaciones del entorno de inyección.

A continuación, se presentan las características de los depuradores actuales que justifican su utilidad en la inyección de fallos. En el apartado 5.3 se describe el estándar IEEE 1149.1 que define el interfaz JTAG para test y depuración utilizando la cadena de rastreo periférico o *boundary scan*, como se denominará en adelante en este documento. En el apartado 5.4, se explica en detalle el sistema de inyección para microprocesadores que se propone en esta tesis como aportación original. Una optimización de ese entorno de inyección aplicable a sistemas integrados en un circuito con lógica programable, SoPC, se presenta en el apartado 5.5 también como aportación original y finalmente se realiza un resumen y se establecen las conclusiones relativas a este capítulo.

5.2 Infraestructuras de depuración en microprocesadores

El diseño de un circuito digital requiere de una fase de depuración para la detección y eliminación de posibles errores. En una primera fase el mecanismo de depuración es la simulación de la aplicación. Las herramientas de simulación más

actuales son capaces de simular el sistema completo. En el caso de microcontroladores, por ejemplo, simulan los recursos internos pero también las entradas y salidas tanto digitales como analógicas, los puertos de comunicaciones, interrupciones, temporizadores, etc. Una vez prototipado el microprocesador, la depuración requiere mecanismos *hardware* para la localización de errores, tanto en el circuito como en el *software* que se ejecuta. La depuración *hardware* permite comprobar el correcto comportamiento de la implementación final del circuito y su interacción con el exterior. Principalmente, se pueden distinguir tres mecanismos *hardware* distintos: monitores, emuladores en circuito (ICE, *In Circuit Emulator*) y depuradores o emuladores integrados en el chip (OCD).

Los monitores son programas especiales de supervisión que se añaden a la memoria de instrucciones del procesador y que se encargan de realizar tareas elementales de depuración. El monitor se comunica con la herramienta *software* de depuración (que se ejecuta en un computador externo) a través de una interfaz de entrada/salida del microprocesador. Este mecanismo no requiere de ningún *hardware* adicional para la depuración y permite la ejecución de la aplicación objetivo a la velocidad normal de funcionamiento. Los monitores son las herramientas menos potentes y consumen recursos del procesador para la ejecución del código de depuración.

Los emuladores en circuito o ICE consisten en un *hardware* específico que sustituye al microprocesador a depurar, para lo cual necesitan tener un conector con el mismo encapsulado. Los ICE proporcionan trazas de ejecución en tiempo real y de la actividad de los pines. No consumen recursos internos del procesador y son las herramientas más potentes. Sin embargo, son dispositivos muy caros y específicos para cada versión de microprocesador. El incremento de la velocidad, nivel de integración y complejidad de los circuitos modernos dificulta el desarrollo de ICEs y puede originar problemas en la transmisión de señales a través de los pines. Además el consumo creciente de SoCs y sistemas empotrados dónde no es posible sustituir un componente por otro ha llevado al estudio de nuevos mecanismos de depuración.

Para superar los inconvenientes de los monitores e ICEs se han desarrollado las infraestructuras de depuración integradas en el circuito microprocesador objetivo, OCD, que se controlan desde un computador externo, generalmente un PC, mediante una herramienta de depuración *software*, Figura 56. Esta lógica específica no consume recursos del microprocesador y se comunica con el PC mediante una interfaz serie. Por lo tanto, se incorporan alrededor de 4 ó 5 pines adicionales dedicados especialmente al manejo del depurador y la comunicación entre el PC y el OCD. De esta forma, no es necesario recurrir a los puertos o periféricos disponibles en el microprocesador bajo estudio lo que evita que se modifique el comportamiento normal del sistema y hace que

este mecanismo no sea invasivo desde el punto de vista *hardware*. La interfaz de depuración puede ser estándar, como es el caso de la interfaz JTAG, o *propietario*, es decir, desarrollada por el fabricante del microprocesador, como por ejemplo el BDM de Motorola, "Extended JTAG" (E-JTAG) de MIPS, o el MPLAB de Microchip. El OCD se considera la mejor opción para depurar, en especial, en sistemas empotrados y SoCs por lo que muchos de los fabricantes de microprocesadores y microcontroladores lo integran en sus CIs, como por ejemplo sucede en los procesadores ARM (varios fabricantes como Philips o Intel®), PowerPC (IBM), ColdFire o MPC800 (Freescale™ Semiconductor, antes Motorola), MIPS, AMD, etc.

Las ventajas de utilizar infraestructuras de depuración integradas son las siguientes:

- Bajo coste
- No requiere conexiones o adaptadores de elevado coste como era el caso de los ICEs
- Al estar integradas en el circuito no se necesita un dispositivo adicional para acceder a los recursos internos del microprocesador y se evitan los problemas de interconexionado que generaban los ICEs
- Proporcionan mayores capacidades de depuración que los monitores porque permiten el acceso a más partes del circuito.

Por otro lado hay que considerar las siguientes limitaciones:

- La comunicación entre el computador y el procesador es la principal limitación de estos mecanismos.
- Puesto que la lógica de depuración está integrada en el circuito las posibles acciones suelen ser más limitadas que las del ICE para reducir el coste.

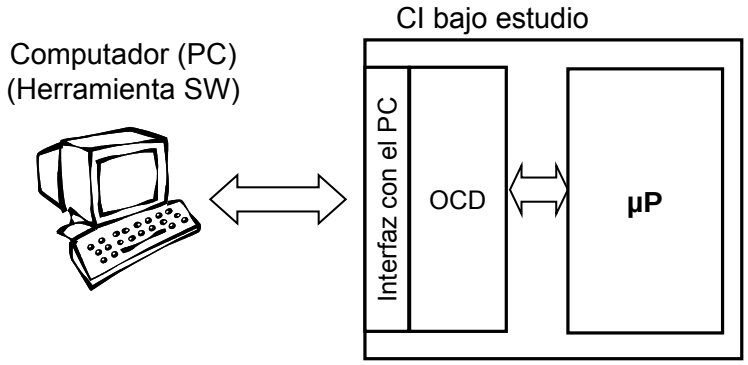


Figura 56. Sistema de depuración de un microprocesador utilizando un OCD.

Las capacidades que proporciona un OCD son:

- **Control de la ejecución.** Un OCD permite modificar la ejecución de una aplicación mediante el uso de distintos mecanismos y funciones como las siguientes:
    - **Puntos de ruptura (*breakpoints*)**. Permiten parar la ejecución normal de la aplicación, entrando en modo de depuración, cuando cierta condición se cumple. Estas condiciones suelen consistir en alcanzar una dirección determinada de la memoria de instrucciones o acceder a una dirección de la memoria de datos. Los puntos de ruptura pueden ser *hardware* o *software* en función del modo en el que se genera dicha condición. Para implementar un punto de ruptura *hardware* se requiere cierta lógica encargada de verificar la condición, por ejemplo en el caso de una instrucción es necesario un bloque encargado de comparar el bus de direcciones con el contenido del contador de programa. Las rupturas *software* consisten en modificar una sección de código de forma que cuando la ejecución de la aplicación alcance esa instrucción se genera una interrupción.
    - **Ejecución paso a paso.** La aplicación *software* se detiene tras la ejecución de cada instrucción.
    - Activación del modo de depuración mediante una **señal externa.** Este mecanismo permite llevar al microprocesador a un modo de depuración de manera forzada desde el exterior.
    - **Reanudar** la ejecución. Una vez el microprocesador entra en modo de depuración deteniéndose su ejecución normal, es posible reanudar dicha ejecución de nuevo devolviendo al microprocesador a un modo normal de funcionamiento.
- **Leer** el valor de registros (de uso general o especiales como el contador de programa o el registro de estado) y palabras de la memoria, sin detener la ejecución. Aunque, generalmente, la lectura de un registro suele ir precedida de la congelación de la aplicación para evitar que el valor se sobrescriba.
- **Escribir** nuevos valores en los registros (de uso general o registros especiales como el contador de programa o el registro de estado) y en la memoria sin detener la ejecución.

Estas funciones permiten el acceso a los recursos internos del procesador. Dicho acceso se considera la mayor dificultad a vencer en las técnicas de inyección aplicadas sobre componentes comerciales de circuitos complejos, como los microprocesadores.

Por lo tanto, los OCDs proporcionan un mecanismo eficaz y de bajo coste para inyectar fallos transitorios y evaluar la tolerancia a fallos de los microprocesadores.

La implementación de las infraestructuras de depuración varía de unos fabricantes a otros y algunas funcionalidades adicionales pueden estar disponibles en los circuitos, o no, dependiendo del fabricante. Por ejemplo, IBM incluye un conjunto de pines que permiten monitorizar trazas de ejecución en tiempo real. En cualquier caso, las utilidades básicas son suficientes para realizar la inyección en cualquier circuito con OCD. Evidentemente, si se dispone de capacidades adicionales puede ser posible introducir mejoras en el proceso de inyección. El *Nexus 5001™ Forum*[18] ha desarrollado un estándar [Nexus] de depuración para microprocesadores y microcontroladores. El estándar publicado en 2003 describe una interfaz de propósito general para el desarrollo de herramientas *software* de depuración y de las infraestructuras de depuración en procesadores empotrados.

Nexus describe un interfaz de depuración basado en el envío de mensajes entre el dispositivo a depurar y el computador externo (PC). Permite varias implementaciones que se clasifican en cuatro clases distintas en función de las características que incluye. Los dispositivos compatibles con la clase 1 son los más sencillos y los que menos capacidades de depuración proporcionan, mientras que la clase 4 es la más potente. Los dispositivos con un OCD de clase 1 sólo usan el interfaz JTAG, pero para el resto de las clases se debe incluir un puerto auxiliar que puede ser utilizado solo o en paralelo con el JTAG. El disponer de dos puertos permite enviar mensajes bidireccionales, de entrada y salida, por lo que se aumenta la velocidad de transmisión (*full-duplex*) con respecto a la clase 1 (*half-duplex*) en la que sólo se pueden enviar mensajes en una dirección cada instante. El puerto auxiliar se introdujo para obtener un mayor ancho de banda en el envío de mensajes entre el microprocesador y el computador externo y, así, dar soporte a la depuración en tiempo real. Los dispositivos con OCDs de acuerdo a la clase 4 son capaces de leer y escribir posiciones de memoria en tiempo real o monitorizar el flujo de programa en tiempo real.

Como se vio en el capítulo 3 (apartado 3.4.2.2) existen algunas propuestas de entornos de inyección basados en el uso de un depurador compatible con el estándar Nexus de la clase 4 [Peng06][Fida06], por lo que es el caso más restrictivo, utilizando el puerto auxiliar para la comunicación. En [Peng06] la herramienta de inyección era *software* y en [Fida06] se propone una implementación *hardware* del sistema de inyección, pero sólo se presentan resultados de simulación. Además dicha implementación *hardware* consiste en la modificación del OCD introduciendo lógica

[18] Formalmente conocido como *Global Embedded Processor Debug Interface Standard Consortium*

encargada de ciertas tareas de inyección, por lo que es necesario disponer de la descripción del OCD, lo que no es habitual.

Actualmente, debido a que la propuesta del estándar Nexus es relativamente reciente, no se ha implantado de forma mayoritaria en los dispositivos comerciales. Algunos fabricantes han aprovechado la cadena de *scan* del JTAG, disponible ya para realizar el test del CI, como interfaz con el OCD, e incluso, para descargar el código a ejecutar en el microprocesador. Además, los OCDs con interfaz JTAG se utilizan no sólo para depurar sino también en el test de fabricación puesto que permite comprobar la memoria y los registros de forma sencilla, y utilizando el mismo conector. Por estas razones el interfaz JTAG es el más utilizado para acceder a las infraestructuras integradas de depuración.

A continuación, y puesto que el sistema de inyección propuesto se basa en el uso de OCDs con interfaz JTAG, se presenta una revisión del estándar [JTAG] como parte del estado de la técnica, indicando los conceptos básicos del mismo.

5.3 El estándar JTAG

El sistema de inyección que se propone en este capítulo es aplicable a microprocesadores que incluyen un OCD con un interfaz JTAG. En este apartado se describen las características principales de dicho estándar y a las que se hará referencia durante los siguientes apartados.

El estándar JTAG [JTAG] define una circuitería que se puede integrar en un CI para realizar tareas de test. Dicha circuitería permite el test de un CI y sus interconexiones y el acceso a la actividad del circuito durante su operación normal. La especificación describe una arquitectura de test basada en una cadena de rastreo del contorno o de la periferia, denominada *boundary scan*, a la cual se puede acceder a través de un puerto de acceso serie, Figura 57, denominado TAP (*Test Access Port*). El registro de la periferia o registro de *boundary scan* es un registro serie, adyacente a cada pin del componente, de forma que las señales de los pines se pueden controlar y observar usando la lógica de test.

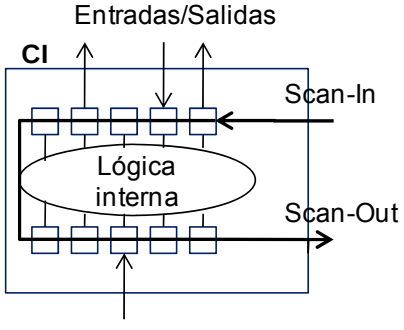


Figura 57. Boundary Scan

La arquitectura propuesta está formada por cuatro componentes: varios registros de datos, un registro para instrucciones, el TAP y un módulo de control denominado controlador de TAP, véase la Figura 58. Cada registro de datos está asociado a alguna parte del circuito a la que pueden acceder para leer información o para introducir un dato. Los comandos de test almacenados en el registro de instrucción indican la acción a ejecutar y seleccionan el registro de datos al que acceder. Los datos y las instrucciones son registros serie, que están conectados en paralelo a la misma entrada de datos serie (TDI) y la misma salida (TDO), las cuales son señales del TAP. La selección entre los registros de datos o el de instrucciones la realiza el controlador del TAP.

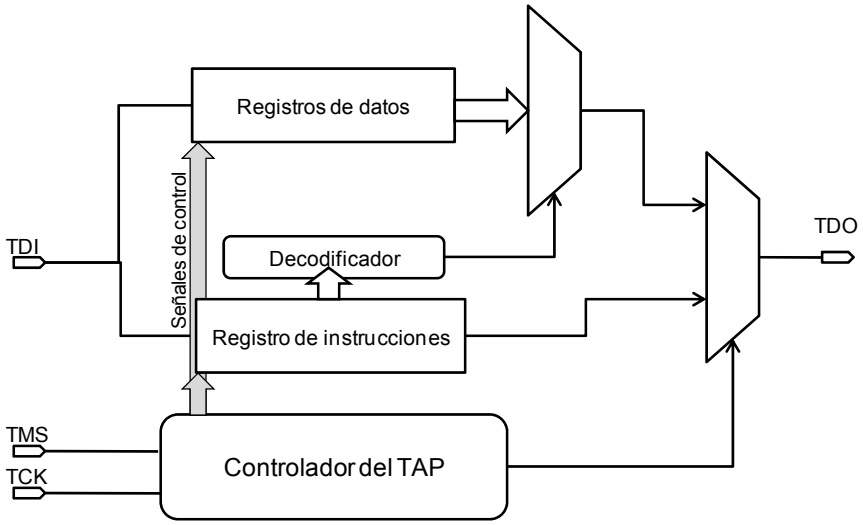


Figura 58. Arquitectura del estándar JTAG

5.3.1 El TAP

El puerto de acceso consta de cinco pines dedicados para realizar la transmisión de datos vía serie que son los siguientes:

- TDI (*Test Data Input*) es el dato de entrada.
- TMS (*Test Mode Select*) se utiliza para seleccionar el modo de test. Se decodifica por el controlador TAP para decidir las operaciones de test a realizar.
- TDO (*Test Data Output*) es el dato de salida.
- TCK (*Test ClocK input*) es la señal de reloj. Es un reloj independiente de la señal de reloj del CI que se utiliza para sincronizar el funcionamiento de la lógica de test. En los flancos de subida de TCK se muestrean las señales TMS y TDI. En los flancos de bajada de TCK se actualiza el estado del controlador TAP y se muestrea el dato de salida TDO.
- TRST (*Test Reset Input*) es la señal de inicialización asíncrona del controlador del TAP definido en el estándar. Su implementación es opcional.

5.3.2 Controlador de TAP

El controlador de TAP es una máquina de estados síncrona con TCK que recibe la señal TMS y genera las señales de control necesarias para controlar los registros de datos y el de instrucciones y la secuencia de operaciones de test. En la Figura 59 se muestra el diagrama de estados que describe el estándar para el controlador. Como puede observarse, los estados necesarios para acceder al registro de instrucciones, que incluyen la terminación *IR*, son los mismos que para el caso de un acceso a un registro de datos, que añaden la terminación *DR*. En el estado *Test-Logic-Reset* la lógica de test permanece desactivada, permitiendo la operación normal del CI. En *Run-Test/Idle* se ejecutan determinadas acciones de test o por el contrario se permanece en reposo en función de la instrucción a ejecutar. Los estados *Select-X-Scan*, siendo *X IR* o *DR*, son estados temporales en los que los registros no se modifican y se utilizan para seleccionar entre el registro de instrucciones o el de datos. En los estados *Capture-X*, se realiza una carga en paralelo del registro seleccionado, bien con datos de monitorización en el caso de datos, o bien con un valor por defecto en el caso de instrucciones. *Shift-X* es el estado en el que se realiza la transmisión serie de datos introduciendo el valor de TDI en el registro cada ciclo y leyendo el valor que se capturó anteriormente por TDO. En *Update-X* se actualizan los registros del JTAG con los valores que se introdujeron por serie en el estado *Shift-X*. El resto de estados son transitorios y en ellos no se realiza ninguna acción de test.

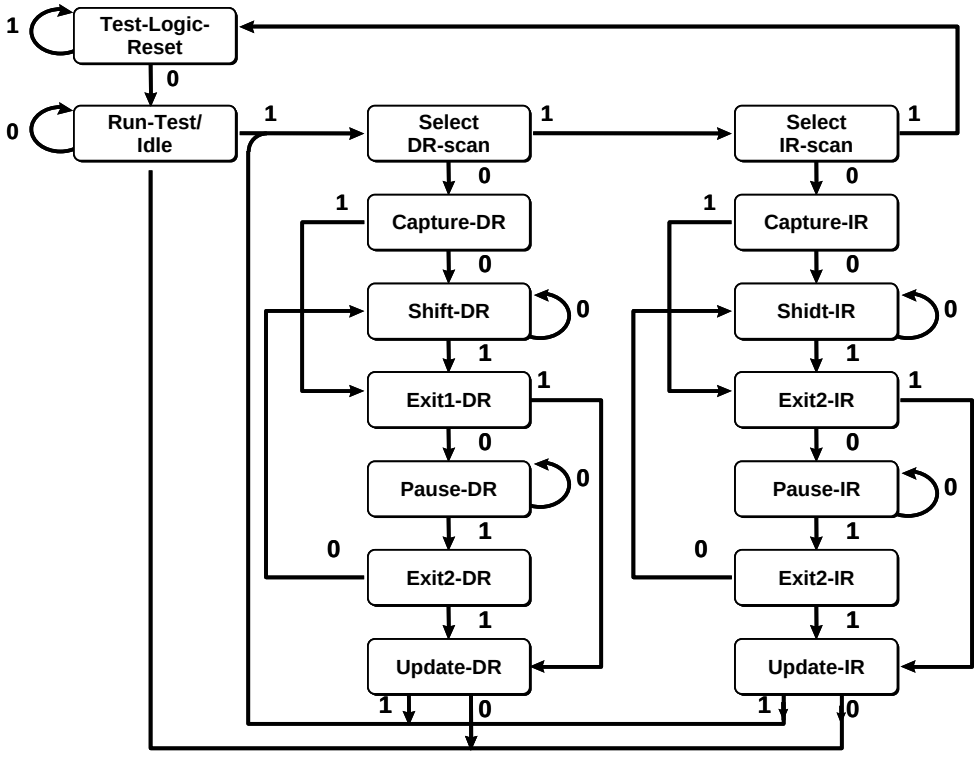


Figura 59. Diagrama de estados del sistema de control del TAP [JTAG]

5.3.3 Registros de datos

El estándar JTAG define un grupo de registros de datos que deben estar incluidos en la lógica de test y permite la existencia de otros opcionales, descritos por el fabricante. Los registros obligatorios son los siguientes:

- Registro de paso o de *bypass*. Tiene una longitud de 1 bit y simplemente proporciona una conexión a través del circuito entre el pin TDI y la salida TDO.
- Registro de la periferia o *boundary scan.* Permite el acceso a las entradas y salidas del sistema y se utiliza para comprobar la lógica interna y las interconexiones de la placa detectando defectos de fabricación.

Opcionalmente puede incluirse un registro para la identificación del dispositivo y tantos registros específicos del diseño como se requieran.

5.3.4 El registro de instrucción

Es un registro de desplazamiento serie que almacena un código de instrucción utilizado para seleccionar una acción de test a realizar o el registro de datos al que se desea acceder. Existen instrucciones obligatorias y otras opcionales que pueden añadir funcionalidades específicas como sucede en el caso de las infraestructuras de depuración integradas. Las instrucciones pueden ser públicas si el usuario del CI puede acceder a ellas y, en ese caso, deben estar documentadas, o privadas si sólo son conocidas por el fabricante. Algunas de ellas, a las que se hará referencia en capítulos posteriores se definen a continuación:

- BYPASS. Se selecciona el registro de *Bypass* entre el pin TDI y el TDO. No se realiza ninguna función de test. Se utiliza principalmente en SoCs con varios CIs cuya cadena de JTAG está conectada en serie para poder realizar tareas de test en cualquiera de los circuitos. De esta forma, cuando se desea realizar operaciones de test únicamente en uno de los circuitos, el resto se configura con la instrucción de *bypass*.
- IDCODE. Selecciona el registro de identificación del circuito, permitiendo leer el código del fabricante.
- INTEST. Se utiliza para escribir un dato en las entradas de la lógica del circuito y examinar las salidas.

5.3.5 Depuración a través del JTAG

Para realizar tareas de depuración a través del interfaz JTAG son necesarias instrucciones y registros especiales dedicados a dichas tareas y que permitan de alguna manera el acceso a los recursos internos como los registros y la memoria. En este caso,

una tarea de depuración consiste en una secuencia de comandos JTAG. Generalmente, los OCDs incluyen registros de datos JTAG para acceder al bus de datos, lo que permite leer la instrucción que se está ejecutando o introducir una instrucción encargada de leer el valor de un registro, escribirlo o acceder a la memoria.

5.4 Sistema de inyección de fallos en microprocesadores

El sistema de inyección que se propone en este capítulo está dirigido a circuitos microprocesadores que incluyen un OCD con interfaz JTAG. Dicho sistema consiste en un entorno de inyección implementado en *hardware* (en una FPGA) encargado de controlar el interfaz JTAG para realizar, a través de las capacidades de depuración, inyección de fallos *bit-flip* en los distintos elementos del procesador.

Este sistema de inyección proporciona ventajas, con respecto al estado de la técnica, en el compromiso necesario entre velocidad, generalidad y coste de la solución para evaluar la tolerancia a fallos en un microprocesador. La técnica propuesta no requiere la modificación del código *software* por lo que no es instrusiva con respecto a la aplicación, a diferencia de las técnicas basadas en *software* [Carr98][Vela00]. Por otro lado, las técnicas basadas en el uso de los mecanismos de depuración presentes en el estado de la técnica tienen limitaciones en la generalidad de la solución. Por ejemplo, como se describió en el capítulo 3 (apartado 3.4.2.2), [Reba99] propone una solución aplicable únicamente a microprocesadores de Motorola, mientras que en [Peng06][Fida06] la aplicación de las soluciones propuestas está condicionada a que el OCD del microprocesador a evaluar sea compatible con la clase 4 del estándar Nexus, lo que es una condición restrictiva, y en concreto, en [Fida06] la descripción del OCD debe estar disponible para su modificación incluyendo funciones relacionadas con la inyección.

El sistema propuesto para la inyección de fallos en microprocesadores en esta tesis doctoral, se basa en el uso de las capacidades básicas de depuración, disponibles en cualquier depurador. Además, se utiliza una interfaz JTAG para controlar el OCD, siendo la interfaz JTAG el mecanismo más común actualmente para acceder a las infraestructuras de depuración integradas.

Por otra parte, la implementación en *hardware* del sistema de inyección completo, evitando realizar tareas de inyección desde el *software,* acelera el proceso de la evaluación de la tolerancia a fallos. Al igual que en el sistema de Emulación Autónoma descrito en el capítulo 4, se disminuye la comunicación necesaria entre el PC y el *hardware*, comunicación que es un cuello de botella en la velocidad del proceso de evaluación. Además, ejecutar las tareas de inyección en *hardware* permite paralelizar ciertas funciones de modo que se realizan a mayor velocidad que en el caso de

implementarlas en *software*. El esquema del sistema propuesto se muestra en la Figura 60.

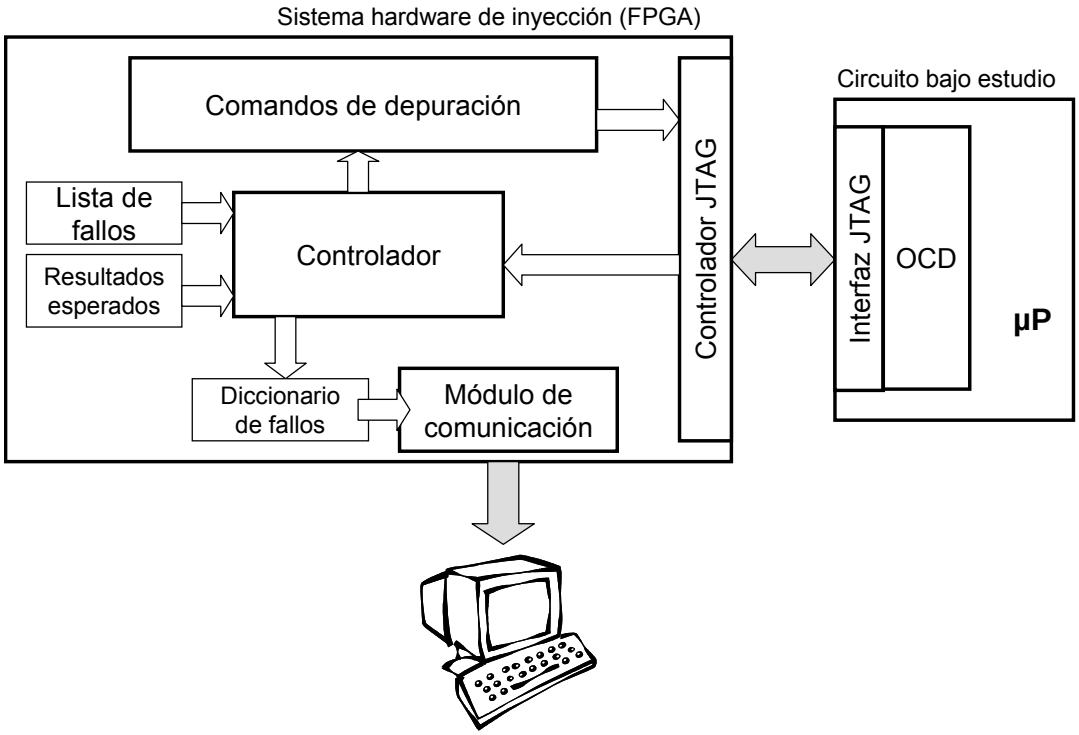


Figura 60. Arquitectura del entorno de inyección propuesto

El sistema de inyección se implementa en *hardware*, en una FPGA, mientras que el PC se encarga de la generación de dicho sistema, la programación de la FPGA y el análisis de resultados a partir de la clasificación de fallos obtenida. El sistema *hardware* de inyección se comunica, a través de una interfaz JTAG con el microprocesador, accediendo a los recursos internos mediante el control del OCD. El entorno propuesto está formado por los siguientes componentes:

- Lista de fallos. Es una memoria que almacena la lista con las posiciones (bit de la de memoria o de los registros del microprocesador) y los instantes de tiempo que caracterizan cada fallo.
- Resultados esperados. Es una memoria que almacena los resultados del circuito en ausencia de fallos o *golden run*.
- Diccionario de fallos. Es una memoria que almacena la clasificación de fallos al tiempo que los datos se van enviando al PC en paralelo con la ejecución de la campaña de inyección.
- Controlador de JTAG. Es una máquina de estados que controla la interfaz JTAG para acceder a la lógica de depuración.

- Controlador de inyección. Es una máquina de estados que se encarga de dirigir todo el proceso de inyección.
- Comandos de depuración. Este módulo contiene la secuencia de comandos JTAG necesarios para realizar cada una de las tareas de depuración utilizadas en la inyección de fallos.
- Módulo de comunicación. Este bloque lee la memoria de la clasificación de fallos obtenida y la envía al PC paralelamente a la ejecución del resto del entorno de inyección.

A continuación se detallan cada uno de estos bloques, describiendo las funciones que realizan, así como algunas características de implementación.

5.4.1 Lista de fallos

Como se explicó en el capítulo 2, los fallos transitorios SEU, pueden afectar a cualquier elemento de memoria en cualquier instante, por lo que la lista de fallos es un conjunto de dos dimensiones (posición en instante del fallo). En un microprocesador, los elementos susceptibles de sufrir un SEU son la memoria SRAM, los registros de propósito general, los registros especiales como el contador de programa o el registro de estados, los registros de segmentación (*pipeline*) y otros registros de control. Según el modelo *bit-flip*, los posibles instantes de inyección a considerar son los correspondientes a cada ciclo de reloj.

La manera más sencilla de fijar el instante de inyección es utilizar un punto de ruptura (*breakpoint*). Este se fija generalmente en cualquiera de las instrucciones de la aplicación objetivo. Cuando la ejecución alcanza el punto de ruptura, el microprocesador finaliza la instrucción en curso y entra en modo depuración, en el cual se detiene la ejecución y se aísla al microprocesador del exterior. Por lo tanto, este mecanismo limita los posibles instantes de inyección, puesto que no es posible acceder a las distintas etapas de segmentación (*pipeline*), durante la ejecución de una instrucción.

Cuando una aplicación contiene bucles de ejecución una misma instrucción puede ejecutarse un número N de veces. Generalmente, la inyección de un fallo en la ejecución i-ésima de la instrucción requiere alcanzar el punto de ruptura, i-veces lo que es un proceso lento. Para evitarlo se activa la inyección utilizando un mecanismo de temporización, que tras cierto tiempo (por ejemplo, fijado de forma aleatoria) lleva al microprocesador al modo de depuración o provoca la inyección del fallo. En algunos OCDs el *hardware* dedicado a la implementación de los puntos de ruptura puede contener la lógica necesaria para considerar también como condición el número de veces que se ha ejecutado una instrucción, lo cual facilita la inyección.

La lista de fallos puede almacenarse tanto en una memoria interna de la FPGA como en alguna de las memorias disponibles en la plataforma. En el primer caso, se obtienen mayores velocidades, pero el número de fallos a inyectar en una campaña está limitado al espacio disponible en la FPGA, que suele ser menor que la capacidad de la memoria de la placa.

5.4.2 Resultados esperados

En función de si la aplicación *software* es principalmente de control o de cálculo los resultados a almacenar difieren. En aplicaciones orientadas a cálculo, los resultados de la aplicación consisten en los datos obtenidos tras las operaciones realizadas y que se almacenan en la memoria de datos del microprocesador. En ese caso, los resultados esperados son dichos valores y la comparación de resultados con los esperados para clasificar el fallo suele realizarse al final de la aplicación.

Si la aplicación es principalmente de control, se requiere almacenar el valor de las señales de salida en los instantes en los que su valor deba cambiar. El análisis de resultados no puede realizarse únicamente al final de la aplicación. Es necesario establecer unos criterios, en función de la aplicación objetivo, para establecer cuándo deben compararse los resultados obtenidos con los esperados. En las aplicaciones de control, el peor caso es aquel en el que es necesario almacenar el valor de las salidas en cada instante de reloj o en cada instrucción ejecutada. En ambos tipos de aplicaciones, se pueden utilizar métodos de compresión de datos para reducir la cantidad de palabras a almacenar.

El análisis de resultados se realiza en *hardware* para acelerar el proceso de evaluación de la tolerancia a fallos. Para ello es necesario disponer de los resultados del sistema libre de fallos. Estos resultados se comparan con aquellos obtenidos en presencia de fallos para poder clasificar los efectos que el fallo ocasiona. Al igual que en el caso de la lista de fallos, los resultados se almacenan en una memoria que bien puede estar en la plataforma *hardware*, o ser implementada con la lógica disponible en la FPGA.

5.4.3 Diccionario de fallos

Los fallos se clasifican en las siguientes categorías:

- *Averías* si provocan un error en las salidas o resultados del microprocesador.
- *Sin avería* cuando el resultado de la aplicación obtenido es correcto. En este caso puede profundizarse más clasificando el efecto en función de si algún elemento de memoria almacena un dato erróneo o no. Para ello habría que leer los datos de dichos elementos.

- *Pérdida de secuencia* si la secuencia de ejecución ha sido alterada llevando a la aplicación a ejecutar instrucciones inválidas o a permanecer en bucles sin fin. Estos efectos se detectan utilizando un bloque de "*perro guardián*" o *watchdog* que comprueba si en un tiempo predefinido se alcanza la instrucción correspondiente al fin de la aplicación.

La clasificación de los fallos puede analizarse en función de la parte del microprocesador en la que se inyecta el fallo, estudiando si los fallos permanecen latentes, los tiempos de latencia, etc. Sin embargo, cuanta más información se desee obtener, mayores son las necesidades de *hardware*, por lo que es necesario alcanzar un compromiso entre la información a obtener y los recursos disponibles.

5.4.4 Controlador de JTAG

El sistema de inyección propuesto utiliza un controlador de JTAG para manejar la interfaz del OCD del microprocesador con el objetivo de acceder a los recursos internos. El controlador de JTAG del sistema se implementa en *hardware* para evitar la limitación impuesta por los protocolos de comunicación entre un PC y el microprocesador y acelerar así la campaña de inyección.

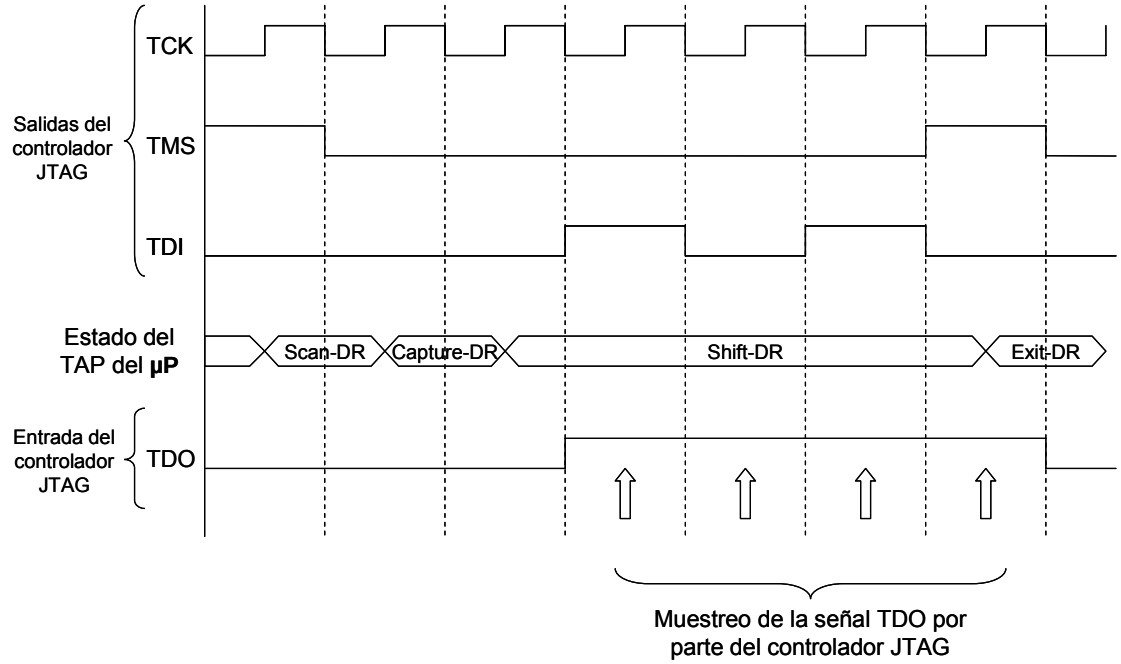


Figura 61. Cronograma de operación del controlador de JTAG

El módulo controlador de JTAG genera las señales TDI, TCK y TMS para controlar el TAP del microprocesador bajo evaluación, y muestrea la señal TDO generada por éste. Este sistema debe tener en cuenta los tiempos definidos por el estándar JTAG para el muestreo de las distintas señales. TDI y TMS se muestrean en los flancos de subida de TCK por lo que es aconsejable generarlos en el flanco de

bajada, mientras que TDO se muestrea en los de bajada y deberá ser muestreado por el controlador JTAG en el flanco de subida. En la Figura 61 se muestra un ejemplo de operación del controlador JTAG, en el cual el valor de entrada introducido en serie por TDI sería "1010" y el dato de salida leído por TDO "1111".

5.4.5 Controlador de inyección

El control del proceso de inyección se realiza en *hardware* de igual forma a como se propuso en el capítulo anterior para el sistema de Emulación Autónoma. El controlador de inyección es una máquina de estados finitos encargada de supervisar el proceso completo de inyección y controlar el resto de componentes del sistema de inyección. Lee las características del fallo a inyectar de la lista de fallos, determina los comandos de depuración a ejecutar controlando el bloque de *comandos de depuración* y se encarga de gestionar la comparación de los resultados almacenando el diccionario de fallos en la memoria correspondiente.

Una señal externa activa la operación de este módulo. Entonces el controlador realiza los siguientes pasos:

1. Se realiza una ejecución sin fallos y se almacenan los resultados obtenidos en la memoria de *resultados esperados*.
2. Se leen la posición y el instante de inyección del fallo a inyectar de la lista de fallos.
3. Se establece un punto de ruptura para fijar el instante de inyección.
4. Se reinicia la aplicación y se activa la ejecución normal del microprocesador.
5. Se comprueba el modo de operación hasta que se verifica que se ha entrado en modo depuración (ha saltado el punto de ruptura).
6. Se realiza la inyección. Para ello primero hay que leer el valor almacenado en la posición del fallo, modificarlo y escribir el nuevo valor erróneo.
7. Se establece un punto de ruptura en el momento de la ejecución en la que se quiere comparar el resultado de la aplicación con el esperado.
8. Se reanuda la ejecución hasta alcanzar el punto de ruptura.
9. Se comparan los resultados. Si la aplicación es de cálculo este punto coincide con el final de la aplicación. En caso contrario, si el fallo aún no ha dado lugar a una avería es necesario repetir los pasos anteriores.

Entonces, se vuelve al punto 7 para establecer otro instante de observación hasta que se llegue al final.

10. La clasificación del fallo se almacena en el diccionario de fallos.

Este bloque puede incluir funcionalidades opcionales que permitan obtener información adicional, como por ejemplo una ejecución libre de fallos paso a paso con el objetivo de conocer la traza de la ejecución de la aplicación.

5.4.6 Comandos de depuración

Como ya se ha dicho anteriormente, este bloque contiene la lógica encargada de las acciones de depuración necesarias para realizar la inyección de fallos. Una acción de depuración, como por ejemplo leer o escribir un registro, se ejecuta introduciendo varias instrucciones y datos JTAG. Dependiendo de la acción a realizar, el conjunto de comandos JTAG es distinto. El bloque de *comandos de depuración* contiene el conjunto de comandos JTAG necesarios para implementar cada acción de depuración necesaria. Cada uno de los conjuntos de comandos recibe el nombre de "*utilidad*". En la Figura 62 se representa la estructura del bloque de *comandos de depuración*.

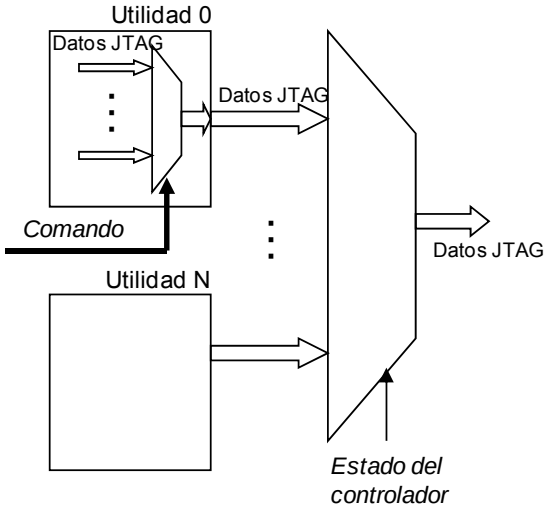


Figura 62. Estructura del módulo de comandos de depuración

Como muestra la figura, en función del estado del *controlador* se establece la utilidad que hay que aplicar. Dentro de cada utilidad se selecciona el valor de las señales necesarias para controlar el interfaz JTAG (las entradas del bloque *controlador de JTAG*) en base al orden de comandos requerido.

Las utilidades básicas necesarias son:

- Establecer/eliminar un punto de ruptura en una instrucción determinada. Mecanismo utilizado para establecer el instante de inyección del fallo o el final de la aplicación.

- Leer/escribir una palabra de memoria dada la dirección. Esto permite modificar el valor de un bit, introduciendo un fallo *bit-flip* en una posición de memoria.
- Leer/escribir un registro dado su identificador. Esto permite modificar el valor de un bit, introduciendo un fallo *bit-flip* en un registro del microprocesador.
- Reanudar la ejecución. Después de que el microprocesador alcance un punto de ruptura, deteniendo su ejecución y entrando en modo de depuración para realizar alguna de las tareas de inyección (inserción del fallo, lectura del contenido de registros para el análisis de resultados, etc.) es necesario reanudar la ejecución.
- Reiniciar la ejecución. Tras la evaluación de un fallo, se reinicia la aplicación para comenzar con una nueva evaluación de otro fallo de la lista de fallo.
- Comprobar si el sistema está en modo depuración. Con las funciones básicas de depuración el OCD no comunica activamente que entra en modo depuración, por lo que el sistema de inyección debe comprobarlo periódicamente, para, así, proceder a realizar las tareas de inyección correspondientes.

5.4.7 Módulo de comunicación

El módulo de comunicación se encarga de enviar la información obtenida desde el sistema de inyección al PC. Este módulo consta de una interfaz de un puerto de comunicación, como por ejemplo el puerto serie, y de un bloque central que lo controla y decide qué información enviar al PC. La operación de este bloque sólo depende de la disponibilidad de datos a enviar, pero su ejecución es independiente del resto de módulos y se realiza de forma concurrente. Se puede utilizar para enviar la clasificación de fallos obtenidos, para leer los resultados de la aplicación cuando la ejecución se realiza sin fallos, para leer la traza de la ejecución o enviar mensajes al PC a lo largo de la campaña de inyección como método de comprobación.

Los resultados de la clasificación se envían al PC en el instante en el que están disponibles, por lo que no es necesario esperar al final de la campaña de inyección para poder acceder a ellos. Esto es posible porque la implementación *hardware* de las tareas de inyección permite paralelizar funciones lo que acelera el proceso de inyección.

5.4.8 Protocolo de inyección

Una vez descrito el sistema *hardware* de inyección y los distintos componentes que lo constituyen, se explica el protocolo requerido para realizar una campaña de inyección de fallos con dicho sistema. Los pasos necesarios son los siguientes:

1. Configurar el sistema de inyección con la lista de fallos, el número de fallos a inyectar, y las características propias de la cadena JTAG del microprocesador bajo estudio y su OCD, como los códigos de las instrucciones o la longitud de los registros de datos destinados a tareas de depuración.
2. Se programa la FPGA con el entorno de inyección.
3. Se activa la campaña de inyección, a través de un puerto de entrada y salida de propósito general o del propio módulo de comunicación.
4. Los datos de la clasificación de fallos son enviados al *host* PC durante la realización de la campaña.
5. La campaña de inyección finaliza. Se indica a través de un LED de estado o del módulo de comunicación.

La comunicación con el PC se realiza al principio para configurar la FPGA y para recoger los resultados. El envío de resultados al PC se realiza en paralelo con la propia campaña de inyección de fallos para acelerar el proceso. Por lo tanto se minimiza la interacción con el PC realizándose la inyección de forma autónoma.

5.5 Evaluación de la fiabilidad en SoPC

La tecnología actual permite integrar sistemas completos en un único chip. Dentro de estos sistemas, los sistemas integrados con lógica programable proporcionan unas características especiales que los convierten en dispositivos de gran interés. Un SoPC es un sistema que integra uno o varios microprocesadores, bloques de memoria y lógica programable, todo ello interconectado, Figura 63.

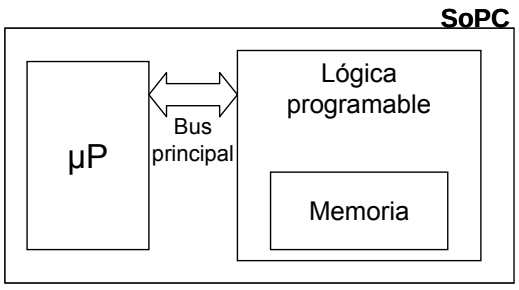


Figura 63. Arquitectura general interna de un SoPC

La combinación de todos los componentes hace que los dispositivos SoPCs sean muy potentes y ofrezcan grandes prestaciones. Los principales fabricantes de estos sistemas son los fabricantes de FPGAs como Xilinx® o Altera®. Estos fabricantes ofrecen dispositivos con hasta cuatro microprocesadores integrados en una FPGA.

La propiedad de reconfigurabililidad de la lógica programable proporciona gran flexibilidad y versatilidad para modificar la funcionalidad implementada y adaptarla a las necesidades de distintas aplicaciones. Esta capacidad de adaptación los hace una solución muy adecuada y de bajo coste para aplicaciones que deban ser actualizadas o para sistemas con un número relativamente bajo de productos fabricados.

En el proceso de evaluación de la tolerancia a fallos de un SoPC es importante poder evaluar la confiabilidad de cada uno de sus componentes en funcionamiento junto con todos los demás, lo que añade complejidad al proceso de evaluación. El diseño implementado en la lógica programable puede evaluarse de igual forma a como se haría con un diseño implementado en una FPGA, inyectando fallos en la lógica de usuario y en la memoria de configuración. La lógica de usuario puede evaluarse con el método de Emulación Autónoma propuesto en el capítulo 4 como aportación original de este trabajo de tesis. Algunas técnicas propuestas para resolver el problema de la inyección en la memoria de configuración fueron descritas en el capítulo 3 (apartado 3.4.2.1). Por otro lado, para la evaluación del microprocesador se puede aplicar la técnica de inyección de fallos que se describe en el apartado anterior, basada en el uso de las capacidades de depuración integradas en el microprocesador a través del interfaz JTAG. Sin embargo, un SoPC ofrece unas características especiales debido a la lógica programable que incluye, que permite proponer algunas optimizaciones en el sistema de inyección. A continuación, se describen las mejoras que se obtienen al implementar tareas de inyección en la lógica programable disponible en el SoPC bajo estudio.

5.5.1 Optimizaciones

La observabilidad y controlabilidad del microprocesador integrado en un SoPC por parte del sistema de inyección aumentan considerablemente cuando ciertas tareas se implementan en el propio sistema a estudiar. A continuación se describen las mejoras obtenidas con el sistema propuesto.

5.5.1.1 Requisitos

En el caso de un SoPC no es necesario recurrir a *hardware* adicional, sino que el entorno de inyección puede implementarse en la lógica programable disponible en el propio sistema, Figura 64. Como se demuestra en los resultados presentados en el capítulo 6, el área necesaria para la implementación del entorno de inyección supone un porcentaje pequeño de la capacidad disponible en las FPGAs comerciales hoy en día.

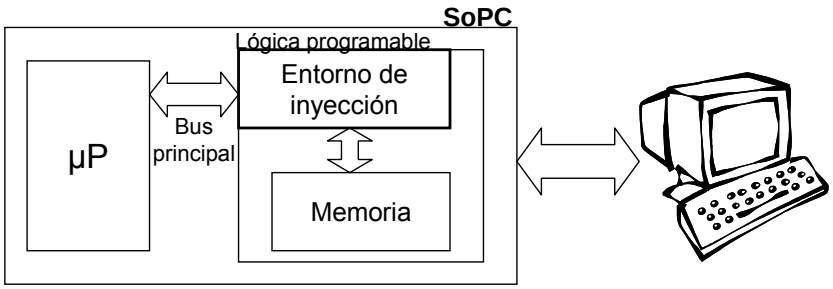


Figura 64. Entorno de inyección para la evaluación de la tolerancia a fallos de SoPCs

5.5.1.2 Sincronismo

El reloj del bloque de inyección es el mismo que el del microprocesador al estar integrados en el mismo chip, lo que supone una mejora en la velocidad del sistema de inyección. Las razones son las siguientes:

- La frecuencia máxima de funcionamiento del reloj TCK de la interfaz JTAG está limitada por los retardos de las señales JTAG. Estos retardos se reducen lo máximo posible cuando el controlador y generador de las señales JTAG están implementados en el mismo CI.
- No es necesario sincronizar el entorno de inyección con el circuito bajo test. Algunos circuitos incluyen lógica de sincronización para muestrear las señales JTAG con el reloj del propio sistema, lo que introduce un retardo adicional. Según el sistema propuesto para realizar la inyección en un SoPC no es necesaria la sincronización de ambos circuitos por lo que esta limitación desaparece.

5.5.1.3 Accesibilidad

La lógica programable tiene acceso a todas las señales que conforman la interfaz del microprocesador y que denominamos bus principal (Figura 64) así como a la memoria del sistema (SoPC), sin necesidad de utilizar un protocolo de comunicación ni de detener la ejecución. Esto permite realizar mejoras en dos tareas fundamentales de la inyección de fallos, el análisis de resultados y la activación del fallo.

El **análisis de resultados** se realiza en el propio sistema, en la lógica programable, sin necesidad de tener que leer a través del interfaz JTAG los resultados de la aplicación. El acceso a la memoria es directo puesto que el sistema de inyección se conecta a sus señales de control. El número de ciclos necesarios para realizar esta tarea se reduce considerablemente con respecto al tiempo empleado usando el interfaz JTAG.

La compresión de datos, por ejemplo usando firmas (MISR, *Multiple Input Shift Register*), se propuso en el apartado 5.4 para acelerar el análisis de resultados. En un SoPC se pueden calcular firmas dinámicas de la ejecución del sistema, observando la evolución del bus del microprocesador. Se propone almacenar la firma de la evolución

de la ejecución en varios puntos de comprobación. Este mecanismo optimiza la detección de las averías y de las pérdidas de secuencia. Observando el bus a lo largo de la ejecución es posible detectar las escrituras en memoria y detectar así fallos latentes y averías cuando los resultados son parte del contenido de la memoria.

Con respecto a la **activación del fallo**, en el caso de disponer del entorno de inyección en el mismo CI que el circuito a evaluar, no es necesario establecer puntos de ruptura porque el bus de direcciones es accesible. Para activar la inyección del fallo se utiliza un temporizador capaz de activar la inyección en cada ciclo de reloj, lo cual es más rápido que el uso de *breakpoints*. Esta solución evita los retardos e inconvenientes que se producen al inyectar fallos en bucles, puesto que, además, se dispone del mismo reloj y la activación del fallo está sincronizada con la ejecución de la aplicación bajo estudio. Es habitual que los microprocesadores integrados dispongan de una entrada que permite detener la ejecución, llevándola a un modo de depuración sin necesidad de introducir un conjunto de comandos JTAG.

Gracias a la mayor accesibilidad a las distintas entradas y salidas del microprocesador se reducen las *utilidades* a implementar en el sistema de inyección. Por ejemplo, el acceso a memoria para la inyección de fallos, en un SoPC no se realiza mediante la lógica de depuración, puesto que al tener acceso a los buses de la memoria y a las señales de control, en el caso de un SoPC se modifican las señales de control directamente.

5.6 Resumen y conclusiones

La Emulación Autónoma propuesta y descrita en el capítulo 4 es una solución efectiva, rápida y de bajo coste para la inyección de fallos en CIs digitales siempre que se disponga de su descripción. En caso contrario, es necesario recurrir a una técnica de inyección sobre un prototipo del circuito o un componente comercial. En este capítulo se ha presentado un sistema de inyección orientado a la evaluación de la tolerancia a SEUs en microprocesadores, dónde la inyección se realiza sobre un componente comercial en vez de sobre una descripción del circuito. El sistema propuesto se basa en el uso de las infraestructuras de depuración integradas en el microprocesador a estudiar para realizar la inyección. Además, se presenta una variación del sistema propuesto para el caso particular de realizar la inyección en un microprocesador integrado en un SoPC.

Los microprocesadores son circuitos muy comunes en los sistemas digitales actuales por lo que la evaluación de su tolerancia a fallos es fundamental. Normalmente, se carece de la descripción de estos circuitos puesto que se utilizan microprocesadores comerciales (IPs). Hoy en día, la mayoría de los microprocesadores disponen de infraestructuras de depuración integradas en el propio circuito; de otra forma, el acceso a los recursos internos se complicaría o sería inviable en los complejos sistemas

actuales, como SoCs o sistemas empotrados. Dichas infraestructuras son un mecanismo sencillo de acceso a la lógica interna del circuito y pueden utilizarse para la inyección de fallos.

Como se explicó en el capítulo 3, existen propuestas de otros autores para inyectar fallos a través de las capacidades de depuración. [Fida06][Peng06] son las propuestas más generales al estar orientadas a microprocesadores con infraestructuras de depuración compatibles con el estándar Nexus. En ambos casos, el OCD debe ser de clase 4 que ofrece las mayores funciones de depuración y por lo tanto, es una condición restrictiva. En [Peng06] el entorno de inyección se implementa en *software* en el *host* PC lo que requiere una comunicación intensiva entre el PC y el circuito a evaluar y supone una limitación en el proceso de evaluación. Por otro lado, [Fida06] propone una infraestructura de depuración modificada que incluye tareas específicas para la inyección. Esta solución es muy efectiva pero requiere disponer de la descripción del OCD. Además, no se presentan resultados de su implementación sino tan sólo de simulaciones.

En el presente capítulo se ha descrito un entorno de inyección para microprocesadores basado en el uso del OCD a través del interfaz JTAG. Con este sistema se obtiene una solución para la inyección de fallos transitorios en microprocesadores que proporciona un compromiso entre la velocidad, la intrusividad, la generalidad y el coste de la solución.

La comunicación con el PC limita la velocidad del proceso de inyección por lo que el entorno de evaluación propuesto se implementa por completo en *hardware*, en una FPGA, aplicando así el concepto de Emulación Autónoma presentado en el capítulo anterior que acelera la ejecución de las campañas de inyección.

El uso del OCD proporciona un mecanismo de bajo coste, sencillo y no intrusivo puesto que no requiere la modificación del sistema de origen ni de su aplicación. Además, debido a la relativamente reciente publicación del estándar de depuración Nexus y a que la interfaz JTAG es utilizada ya en los CI para tareas de test, la mayoría de los OCD disponen de esta interfaz para su comunicación con el exterior. Las funciones de depuración necesarias para implementar la solución propuesta son básicas y están disponibles en cualquier OCD. Por estas razones, se puede concluir que el sistema propuesto es ampliamente general y aplicable a una gran variedad de microprocesadores.

La aplicación del sistema de inyección propuesto en un SoPC es susceptible de implementar optimizaciones que mejoran las prestaciones de la técnica de inyección. Estas mejoras se obtienen al poder implementar el entorno de inyección en la lógica

programable disponible en el propio SoPC a estudiar. Las mejoras que se consiguen pueden resumirse en los siguientes puntos:

- Mejora de la velocidad. Debido a la reducción de los retardos de las señales y el aumento en la controlabilidad y observabilidad de las señales del microprocesador.
- Facilidad para realizar un análisis más detallado de los efectos de los fallos.
- El entorno de inyección se implementa en el propio sistema, evitándose el uso de un *hardware* dedicado.

Como muestran los resultados experimentales que se presentan en el capítulo 6 los recursos necesarios para implementar el sistema de inyección son un pequeño porcentaje de los recursos disponibles en los SoPCs comerciales.

En definitiva, se ha presentado un sistema de inyección que da solución a la evaluación de la tolerancia a fallos SEU en microprocesadores cuando la inyección se realiza sobre un componente comercial o un prototipo. El sistema propuesto se basa en los recursos disponibles en la mayoría de los microprocesadores actuales, como son los OCDs con interfaz JTAG, utilizando las capacidades básicas de depuración. Además, la implementación en *hardware* del sistema de inyección permite acelerar la evaluación con respecto a una implementación *software*. Este sistema junto con la Emulación Autónoma descrita en el capítulo 4 permite estudiar sistemas complejos como SoPCs.

El sistema de inyección propuesto para evaluar microprocesadores, así como las particularización del sistema para el caso de un SoPC son aportaciones originales de este trabajo de tesis doctoral.

CAPÍTULO 6

RESULTADOS EXPERIMENTALES

6. Resultados Experimentales

Las técnicas de inyección propuestas en este trabajo de tesis se han probado experimentalmente y los resultados obtenidos se presentan y analizan en este capítulo. En el presente trabajo de tesis se ha propuesto un entorno de inyección que proporciona solución a las necesidades de los circuitos actuales:

- En el capítulo 4 se presentó un sistema de inyección basado en emulación con FPGAs, aplicable a circuitos digitales de los cuales está disponible una descripción en un lenguaje *hardware* de alto nivel, en concreto VHDL.
- En el capítulo 5 se presentó un sistema de inyección para circuitos microprocesadores cuando la inyección se realiza sobre un componente comercial o prototipo final, en lugar de sobre una descripción del circuito.

Los resultados experimentales que se presentan siguen esta estructura, de forma que se distinguen dos conjuntos de experimentos. En el apartado 6.1, se describen los experimentos realizados para estudiar y probar el sistema de Emulación Autónoma y en el apartado 6.2, se evalúa la técnica de inyección propuesta para microprocesadores sobre un componente comercial. En ambos casos, los experimentos desarrollados consisten en aplicar la técnica de inyección propuesta para evaluar la tolerancia a fallos en circuitos de prueba y circuitos reales. El sistema de inyección se analiza en función del área necesaria para su implementación, el tiempo de evaluación empleado y la cobertura de fallos obtenida. Dentro de cada uno de los dos apartados se describen los circuitos utilizados, el método experimental aplicado y los resultados obtenidos.

6.1 Emulación Autónoma

Las técnicas propuestas para la implementación de un sistema de Emulación Autónoma para la inyección de fallos transitorios se han evaluado utilizando un conjunto de circuitos de prueba, con el objetivo de analizar la eficiencia y prestaciones de cada técnica. Estos resultados se comparan con las estimaciones realizadas de forma teórica para confirmar y ampliar las conclusiones extraídas. Asimismo, se pretende mostrar el coste en recursos de las técnicas propuestas y encontrar la solución más óptima. Además, se han evaluado dos circuitos reales complejos, un circuito de la industria aeroespacial encargado de tareas de control y el microprocesador LEON2.

6.1.1 Descripción de los circuitos de prueba

Las técnicas de inyección propuestas se prueban y comparan aplicándolas a circuitos de referencia o *benchmarks*. Su aplicabilidad a circuitos reales, de mayor complejidad, se prueba con dos circuitos industriales, un circuito de aplicación aeroespacial que denominaremos Circuito_A y el microprocesador LEON2.

6.1.1.1 Circuitos de referencia

Los circuitos de referencia utilizados para realizar las pruebas de las técnicas de inyección propuestas son algunos de los circuitos de prueba presentados en [Corn00]. También se evalúa un circuito que implementa el algoritmo CORDIC para datos de 16 bits. La descripción de este circuito, que denominamos CORDIC16 está disponible en [OpenCore] de forma gratuita. Los circuitos de prueba utilizados son representativos de circuitos típicos o partes de circuitos y están descritos con VHDL, en el nivel de transferencia de registros. En concreto, se han evaluado los circuitos denominados b12, b14 y b15 además del CORDIC16.

La Tabla 2 recoge algunas características de los circuitos de referencia utilizados como son el número de bits de entrada y de salida, y el número de biestables. Estas características dan información sobre el tamaño de los circuitos y la cantidad de fallos *bit-flip* que se pueden originar (depende del número de biestables). El circuito b12 consiste en un juego de un jugador (adivinar una secuencia), el circuito b14 es un subconjunto de procesador Viper y el circuito b15 es un subconjunto del procesador 80386. El circuito que implementa el algoritmo CORDIC consiste básicamente en una serie de registros encadenados sin realimentación. Además, se ha endurecido parcialmente el circuito b14, triplicando los módulos (TMR) de salida en las principales salidas del circuito. Comparar los resultados, en área, tiempo y cobertura de fallos obtenida, permite comprobar la eficiencia de la técnica de inyección en el diseño y evaluación de circuitos tolerantes a fallos.

	#Entradas	#Salidas	#Biestables
b12	5	6	119
b14	32	54	215
b14_TMR	32	54	323
b15	36	70	418
CORDIC16	51	48	893

Tabla 2. Características de los circuitos de referencia utilizados para evaluar las técnicas de inyección propuestas.

Ninguno de estos circuitos contiene memorias empotradas porque el primer objetivo es estudiar las características de la Emulación Autónoma propuesta y las prestaciones que ofrece. La inyección y emulación de circuitos con memorias empotradas se analiza con circuitos reales.

6.1.1.2 Circuito_A

El diseño circuito_A es una aplicación real de la industria aeroespacial, donde la tolerancia a fallos es un factor esencial. Este circuito es fundamentalmente una interfaz inteligente de los dispositivos telemétricos y los componentes motores de una aplicación

espacial, con la unidad central de procesamiento de la misión. Integra numerosos mecanismos de control de los modos de función, así como, de prevención y diagnóstico de errores. Es un circuito robusto por diseño pero tiene algunos elementos críticos que conviene endurecer con las técnicas típicas pasivas de redundancia *hardware*. Se han inyectado fallos en dos versiones del circuito. Circuito_A es una versión sin endurecer, mientras que Circuito_A_TMR incluye la triplicación de biestables en algunas salidas críticas, así como, en ciertas señales internas. En la Tabla 3 se recogen el número de bits de entrada, de salida y el número de biestables de ambas versiones del circuito.

	#Entradas	#Salidas	#Biestables
Circuito_A	32	68	484
Circuito_A_TMR	32	68	582

Tabla 3. Características del circuito_A y su versión endurecida parcialmente.

6.1.1.3 LEON2

La descripción VHDL LEON2 es sintetizable e implementa un procesador de 32 bits de acuerdo con una arquitectura SPARC (*Scalable Processor ARChitecture*), versión 8. La descripción utilizada es la disponible en [Gaisler][19]. Este procesador se desarrolló inicialmente para utilizarlo en aplicaciones espaciales para las que eran necesarias altas prestaciones y cierto nivel de tolerancia a fallos. De hecho, una versión endurecida se utiliza actualmente en la Agencia Espacial Europea (ESA) [Gaisler].

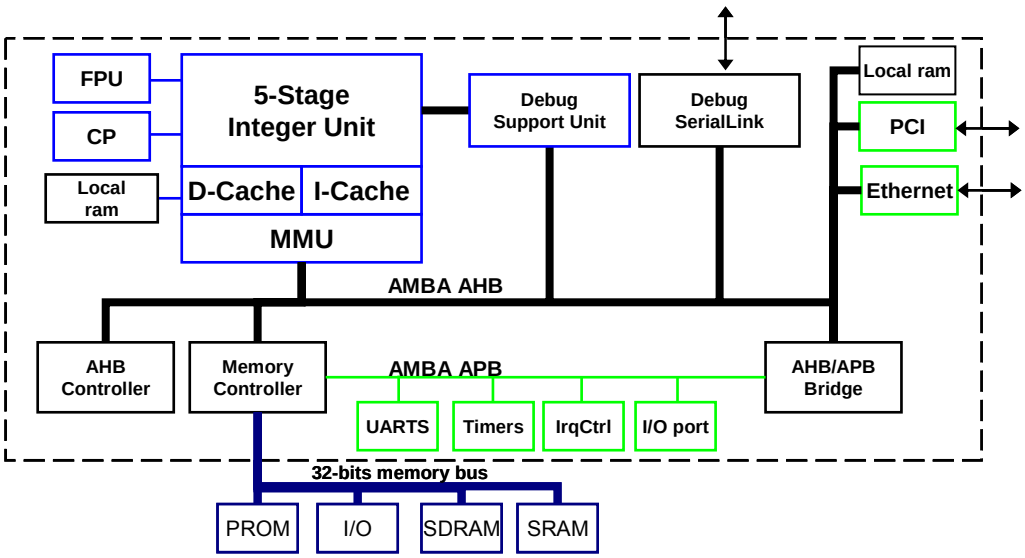


Figura 65. Diagrama de bloques del microprocesador LEON2

La Figura 65 muestra un diagrama de bloques del LEON2. El procesador contiene una unidad de enteros (IU, *Integer Unit*), memoria caché, un bloque de control de la memoria, módulos de depuración, y opcionalmente puede incluir una unidad de

[19] Versión disponible bajo una licencia pública general menor GNU (LGPL, *Lesser General Public License*)

punto flotante (FPU, *Floating Point Unit*) y un coprocesador. Estos dos últimos módulos opcionales no se han incluido en la versión implementada para realizar los experimentos. La descripción utilizada para los experimentos de inyección desarrollados contiene 2,625 biestables, 8 KB de memoria caché de instrucciones, 8KB de memoria caché de datos, y memoria principal. Se considera que la memoria principal no se implementa en SRAM y por lo tanto no es susceptible de sufrir SEUs. Como entradas del circuito se utiliza el bus de datos de la memoria principal, y se consideran como salidas los datos que se escriben en la memoria principal y el puerto de entrada/salida.

6.1.2 Método experimental utilizado

En este apartado se describe el protocolo aplicado en los experimentos de inyección realizados mediante Emulación Autónoma. El protocolo definido contiene los pasos seguidos en la implementación de un sistema de inyección basado en una estructura de Emulación Autónoma, así como, las particularidades a tener en cuenta en función de la técnica utilizada, *Time-Multiplexed*, *Scan-State* o *Mask-Scan*. También se presentan las herramientas *hardware* y *software* utilizadas en la realización de los experimentos. Algunas de dichas herramientas son comerciales, mientras que otras han sido desarrolladas por el equipo investigador en el que se ha desarrollado este trabajo de tesis, con el fin de automatizar la generación del sistema de inyección.

6.1.2.1 Protocolo definido

Las tareas realizadas para la implementación del sistema de inyección basado en Emulación Autónoma con FPGA están representadas en la Figura 66. En la izquierda de la figura se indican las tareas y en la derecha se muestran los resultados principales generados.

Se parte de la descripción en VHDL del circuito bajo estudio y de un banco de pruebas representativo de su operación normal. En primer lugar, se modifica la descripción del circuito de acuerdo con alguna de las técnicas de inyección que se han propuesto en el capítulo 4, *Time-Multiplexed*, *State-Scan* o *Mask-Scan*, sustituyendo cada biestable o bloque de memoria por la estructura correspondiente. El circuito modificado o instrumentado es uno de los bloques que conforman el sistema de inyección y debe ser añadido al sistema emulador, al igual que el bloque de control de la emulación, los vectores de entrada del banco de pruebas o el interfaz de comunicación con el PC (véase la Figura 67 en la página 157 de este documento). Los módulos encargados del control, inyección y observación de resultados dependen únicamente de la técnica de inyección a aplicar, por lo que una vez generados son válidos para todo circuito. Es decir, para evaluar un nuevo circuito, sólo se requiere modificar los módulos correspondientes al circuito instrumentado y a su banco de pruebas.

Una vez generado el circuito de emulación, éste se implementa en la plataforma FPGA, y se realizan las campañas de inyección necesarias para la evaluación de la tolerancia a fallos tipo SEU del circuito.

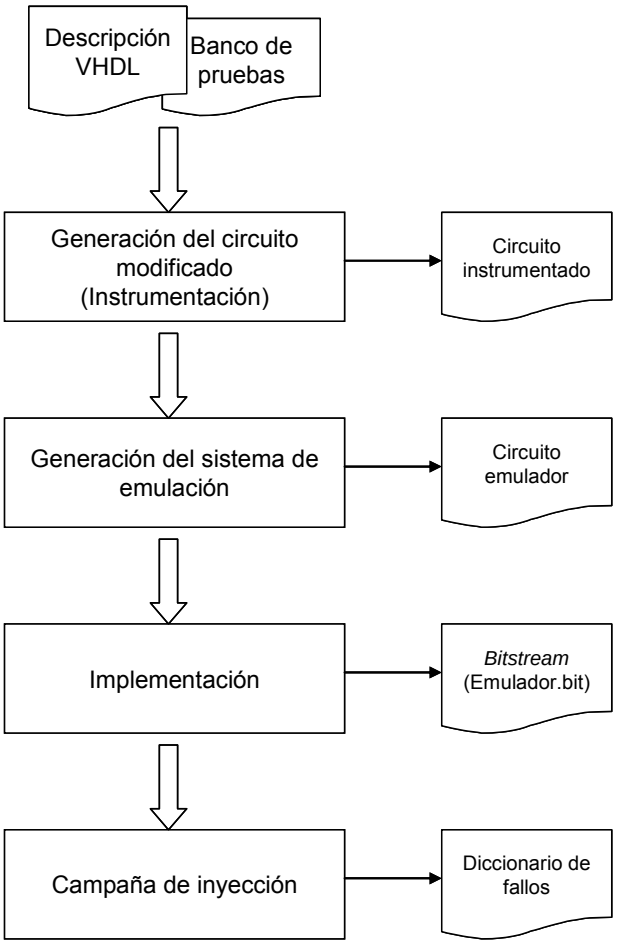


Figura 66. Protocolo seguido en la implementación de un sistema de Emulación Autónoma

6.1.2.1.1 Generación del circuito instrumentado

La instrumentación de un circuito consiste en sustituir cada elemento en el que se desea inyectar fallos, en concreto un biestable o un bloque de memoria síncrona, por otro bloque modificado con *hardware* adicional para el desarrollo de tareas de inyección. El instrumento o componente modificado incluye ciertas capacidades adicionales, necesarias durante el proceso de inyección, como la auto-inyección de fallos o capacidades de observación y análisis del comportamiento del circuito en presencia de fallos. La interfaz de los componentes modificados se ve alterada por la adición de nuevos puertos de entrada y salida que permiten el control y acceso a las nuevas funciones añadidas. Por lo tanto, la interconexión entre las nuevas señales, así como, la modificación de la interfaz del propio circuito también es parte de los cambios a realizar durante el proceso de instrumentación.

La modificación del circuito a evaluar es automática para evitar posibles errores introducidos por el diseñador durante el proceso de instrumentación. Por tanto, es necesaria una herramienta capaz de localizar y sustituir cada componente a modificar. Esta tarea se facilita enormemente si se dispone de una *netlist* del circuito, descrita estructuralmente en función de los elementos básicos del circuito como biestables y bloques de memoria, puesto que entonces su localización y sustitución se realizan de forma directa. Es posible, describir un circuito como una *netlist* estructural usando VHDL, sin embargo, la descripción se hace de modo comportamental para facilitar el diseño. En este caso, el primer paso es sintetizar el diseño para obtener una *netlist* en función de sus componentes y descrita en VHDL que posteriormente se modifica.

6.1.2.1.2 Generación del sistema de emulación

La generación del sistema de emulación consiste en construir un diseño jerárquico que contenga todos los bloques necesarios, descritos en el apartado 4.2, para implementarlo en la FPGA (Figura 67). Como se ha dicho en el apartado anterior, los módulos que constituyen el emulador son válidos para todo circuito, con excepción del propio circuito a estudiar y su banco de pruebas, dependiendo únicamente de la técnica de inyección utilizada.

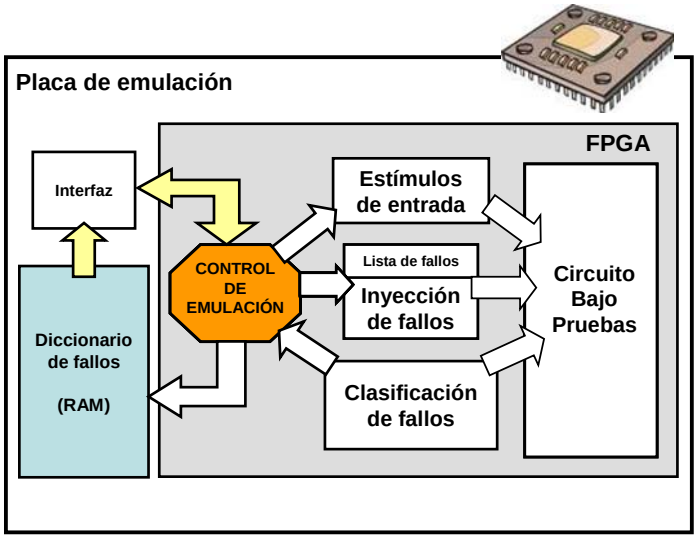


Figura 67. Estructura del sistema de Emulación Autónoma

Los bloques o funcionalidades que se implementan en la FPGA como parte del emulador son los siguientes:

- El circuito instrumentado, que se genera como se ha explicado en el apartado anterior.
- El banco de pruebas. Es el conjunto de vectores de entrada del circuito para cada ciclo de reloj de la ejecución. Como ya se ha dicho anteriormente, los estímulos de entrada del circuito influyen directamente sobre su comportamiento en

presencia de fallos, ya que fijan la actividad de las distintas partes del circuito, por lo que es importante realizar la inyección de fallos cuando se está aplicando una carga de trabajo representativa. Es importante que este conjunto de entradas se aplique a la velocidad de funcionamiento del circuito. Por lo tanto, conviene que estén disponibles en el mismo soporte *hardware* que el circuito. Este bloque puede implementarse como una memoria o como un circuito generador de los vectores de entrada. El banco de pruebas dentro del entorno de inyección debe modificarse cuando se evalúan distintos circuitos, o también cuando, para un mismo circuito, se realizan campañas de inyección bajo distintas cargas de trabajo. En la mayoría de los experimentos que se presentan en este capítulo el banco de pruebas se ha implementado en una memoria, aunque para algunos circuitos se optó por utilizar un circuito encargado de generar los vectores de entrada para cada ciclo.

- El control de la emulación. Este bloque depende de la técnica de inyección elegida, *Time-Multiplexed*, *State-Scan* o *Mask-Scan*. El bloque de control de la campaña de inyección para cada una de las técnicas propuestas se ha desarrollado de forma que éste bloque es independiente del circuito que se evalúa. Por lo tanto, su inserción como parte del sistema de emulación es directa para cada campaña a realizar.
- Bloque de comunicación entre el PC y la FPGA. Este bloque depende fundamentalmente de la plataforma de emulación utilizada. Una vez seleccionada una plataforma *hardware*, el bloque de comunicación no se modifica para los distintos experimentos. A lo largo del desarrollo de este trabajo de tesis se han utilizado dos plataformas de emulación distintas que se describen en detalle en el apartado *6.1.2.2 Herramientas*, por lo que en función de la interfaz y recursos disponibles se han implementado distintos bloques de comunicación.

En definitiva, la generación del circuito emulador es una tarea sencilla y fácilmente automatizable. Los componentes que conforman el sistema de emulación están definidos y desarrollados en función de la técnica de inyección que se aplica (*Time-Multiplexed*, *State-Scan*, *Mask-Scan*), a excepción del propio circuito que se quiere evaluar y su banco de pruebas, por lo que no es necesario modificarlos para cada nuevo experimento de inyección.

6.1.2.1.3 Implementación del sistema de emulación

El sistema de inyección se ha descrito en VHDL, e implementado en una FPGA mediante las herramientas de síntesis que proporciona el fabricante. Puesto que el sistema de inyección está descrito en VHDL puede implementarse en cualquier

dispositivo FPGA, con independencia de la tecnología y el fabricante. El único requisito es que el dispositivo debe disponer de los recursos necesarios, lo que, como se verá en los resultados obtenidos, se cumple con las FPGAs actuales gracias al aumento de la densidad de integración con las nuevas tecnologías. Las FPGAs son cada vez más potentes y ofrecen mayores prestaciones por lo que se facilita la implementación.

Una vez programada la FPGA se realizan las campañas de inyección de fallos. Éstas se activan desde el PC y es el *hardware* el encargado de realizar todas las tareas de inyección. Cuando finaliza la emulación, el PC descarga el diccionario de fallos. Se ha desarrollado una herramienta *software* para acceder a la memoria disponible en la plataforma *hardware* de emulación, dónde se almacenan los resultados. Esta memoria se usa en todas las técnicas para almacenar los fallos que luego hay que descargar. Por otro lado, en la técnica *State-Scan* hace falta guardar los estados de inyección. En ese caso, se utiliza también la memoria de la plataforma FPGA para almacenar dichos estados antes de comenzar la campaña de inyección.

La interfaz entre el PC y la plataforma *hardware* debe permitir el envío y recepción de mensajes entre ambos componentes, para activar la campaña de inyección y notificar su finalización. En función de la plataforma de emulación utilizada, se pueden usar distintos sistemas de comunicación, y por lo tanto distintas herramientas *software* que permitan establecer el intercambio de información. Dichas herramientas se presentan en el apartado 6.1.2.2.

6.1.2.1.4 Resultados de la campaña de inyección

Los resultados de la campaña de inyección que se presentan en este capítulo consisten en la clasificación de fallos, el área necesaria para la implementación del sistema de inyección y el tiempo empleado en realizar el experimento.

El sistema de inyección almacena la clasificación de cada fallo en función del efecto producido. Para obtener una medida estadística de la cobertura de fallos a partir del diccionario de fallos se utilizan herramientas *software* desarrolladas para tal fin en el grupo de investigación en el que se ha desarrollado esta tesis.

Por otro lado, para estudiar la eficiencia de la solución propuesta, la Emulación Autónoma, y de sus posibles implementaciones, se analiza la mejora conseguida en la velocidad del proceso, que es uno de los principales objetivos perseguidos por la nueva técnica de inyección propuesta. El indicador utilizado es el tiempo medio empleado por fallo durante el experimento de inyección para una determinada frecuencia de reloj del circuito emulador. Este tiempo medio se calcula como la relación entre el tiempo transcurrido desde la activación de la campaña (desde el PC) hasta la notificación de su finalización, y el número total de fallos insertados.

Es importante también considerar los requisitos en cuestión de recursos *hardware* como indicador del coste de la solución propuesta. En los resultados se presentan los recursos necesarios en términos de los componentes básicos disponibles en la plataforma FPGA, esto es, número de biestables, número de *Look-up Tables*[20] (LUTs) o memoria utilizada. Estos resultados permiten estudiar el compromiso alcanzado entre prestaciones y coste en cada una de las técnicas de inyección propuestas y comprobar que el método de Emulación Autónoma es una solución de bajo coste, tanto en recursos como por facilitar el desarrollo del circuito al aplicarse de forma rápida en la etapa de diseño, lo que era otro de los objetivos que se plantearon al inicio de este trabajo de tesis.

6.1.2.2 Herramientas y recursos utilizados

En este apartado se describen las herramientas *software* usadas para el desarrollo e implementación de una campaña de inyección mediante un sistema de Emulación Autónoma, así como las plataformas *hardware* utilizadas. Entre las herramientas *software* necesarias, algunas son comerciales y otras han sido desarrolladas para automatizar ciertas tareas.

Las herramientas de síntesis e implementación en la FPGA utilizadas son:

- *Precision RTL* de Mentor Graphics®, para la obtención de la *netlist* estructural en VHDL necesaria en el proceso de instrumentación.
- Xilinx ISE para la implementación y programación del sistema de emulación en la FPGA.

Para las pruebas funcionales y depuración del sistema de emulación durante su desarrollo se ha utilizado la herramienta de simulación ModelSim de Mentor Graphics®.

Por otra parte, para la modificación de los circuitos se ha desarrollado una aplicación con C++, que analiza el código VHDL para localizar los biestables o bloques de memoria que se desean instrumentar y los sustituye por las estructuras modificadas correspondientes añadiendo las interconexiones necesarias. La aplicación para instrumentar automáticamente se desarrolló a partir de una herramienta de análisis de diseños VHDL realizada como parte del trabajo de la tesis doctoral [Garc04a].

La ejecución automática de una o varias emulaciones de fallos así como el análisis de resultados, esto es, la obtención de la cobertura de fallos, se realizan mediante ficheros de comandos o *scripts* en TCL (*Tool Command Language*).

[20] Una LUT de *n*-bits se utiliza para codificar cualquier función lógica booleana de *n* entradas como una tabla de verdad.

Con respecto al *hardware*, durante el desarrollo de este trabajo de tesis la plataforma *hardware* utilizada para implementar el sistema de emulación se ha variado debido a los avances de la tecnología y por lo tanto al aumento de prestaciones en los nuevos dispositivos FPGAs disponibles en el mercado. Se han utilizado dos plataformas FPGA:

- Una placa RC1000 de Celoxica [Celoxica], con una FPGA de Xilinx® Virtex™-2000E (con tecnología CMOS de 180nm), con 43.200 celdas lógicas[21] [Xilinx], y una memoria RAM de 8MB. Esta placa consta de una interfaz PCI que permite una comunicación rápida con el PC.
- Una placa de evaluación de la FPGA de Xilinx® Virtex™-4 LX. Incluye una FPGA XC4VLX60 (tecnología CMOS de 90nm) con cerca de 60.000 celdas lógicas [Xilinx], 32 MB de memoria SDRAM y distintos puertos de comunicación (RS-232, Ethernet y USB).

La interfaz de comunicación utilizada en cada caso es distinta puesto que los recursos disponibles difieren. La placa RC1000 dispone de un registro de control y otro de estado para el envío y recepción de mensajes entre el PC y la FPGA, véase la Figura 68. El registro de control se puede escribir únicamente desde el PC, por lo que se utiliza para enviar datos de 1 *byte* desde el PC a la FPGA. Por su parte, el registro de estado se escribe únicamente desde la FPGA y se usa para enviar información, 1 *byte*, desde la FPGA hacia el PC. En la ejecución de una campaña de inyección, el PC activa el proceso de evaluación mediante el registro de control, y comprueba si se finaliza leyendo el registro de estado.

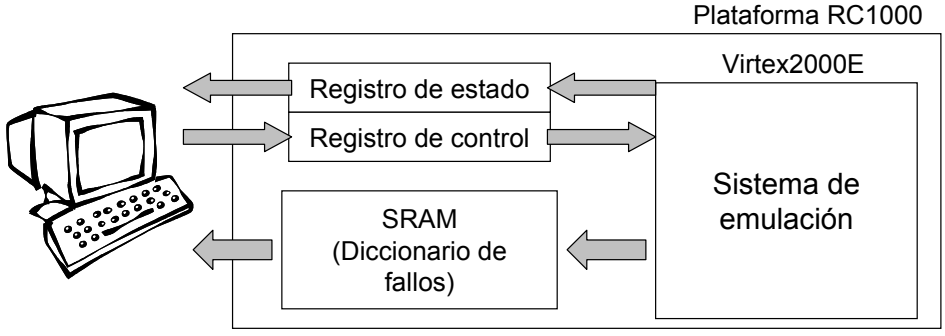


Figura 68. Esquema de la interfaz de comunicación utilizada para la emulación en la plataforma RC1000

El fabricante de la plataforma RC1000 (Celoxica en el momento de su adquisición) proporciona herramientas *software* para manejar la plataforma y facilitar el uso de los recursos de los que dispone. Dentro de estas herramientas hay utilidades que

[21] Xilinx mide la capacidad de sus dispositivos en función del número de celdas lógicas. Una celda lógica se define como la combinación de una LUT de 4-bits y un biestable dentro del mismo bloque programable. Sin embargo Xilinx añade un 12,5% al número total así calculado, para incluir las capacidades que ofrecen otros componentes como los multiplexores disponibles en cada bloque lógico.

permiten escribir el registro de control y leer el registro de estado desde el PC. Para el acceso a la memoria de la RC1000, se ha desarrollado otra herramienta en C++ que utiliza la biblioteca de funciones (API, *Application Programming Interface*) que suministra el fabricante.

La placa de evaluación de Virtex™-4 LX utiliza una tecnología más avanzada que la plataforma RC1000 (90 nm y 0.18 μm respectivamente), con mayor capacidad de integración, por lo que dispone de un mayor número de recursos lógicos. En los experimentos de inyección de fallos implementados sobre esta placa, se ha utilizado la interfaz serie de comunicación RS-232, por simplicidad frente a la interfaz USB. En este caso, ha sido necesario desarrollar un componente *hardware* encargado de comunicar la FPGA y el PC, equivalente a los registros de control y estado disponibles en la plataforma RC1000.

6.1.3 Resultados experimentales

Las técnicas de inyección propuestas en este trabajo de tesis doctoral se analizan en función de su eficacia, coste y velocidad. Los resultados a obtener son los siguientes:

- La clasificación de fallos. El diccionario de fallos proporciona información sobre la tolerancia a fallos del circuito en sus distintas partes, lo que es el objetivo fundamental de toda campaña de inyección.
- Velocidad del proceso de inyección. Uno de los principales objetivos establecidos en este trabajo de tesis doctoral, es acelerar el proceso de inyección notablemente, por lo que el tiempo medio empleado en la inyección de cada fallo es un factor fundamental en la caracterización del sistema de Emulación Autónoma y en el análisis comparativo de las distintas implementaciones propuestas.
- El incremento de área. El número de recursos necesarios para la implementación del sistema de emulación, con respecto al número de recursos disponibles en la FPGA, ofrece un indicador del coste de implementación de la técnica utilizada.

Cada experimento consiste en realizar una inyección exhaustiva, considerando todos los posibles fallos simples que se pueden producir en el circuito a lo largo de la aplicación de un banco de pruebas determinado.

Como se ha dicho anteriormente, se han evaluado tanto circuitos de prueba como circuitos reales. Por un lado, los experimentos con los circuitos de prueba tienen como objetivo medir la eficiencia de la inyección de fallos mediante Emulación Autónoma, y realizar un análisis comparativo de las distintas implementaciones propuestas. Algunos de estos circuitos se han evaluado en otros trabajos aplicando otras técnicas propuestas en la literatura. La comparación entre ambos resultados permite evaluar la mejora

obtenida con el sistema de Emulación Autónoma. Con respecto a la comparación entre las posibles técnicas propuestas, éstas se comparan en términos de la precisión que proporcionan con respecto a la clasificación de fallos, a los recursos empleados y a la aceleración conseguida. En el capítulo 4, se presentó una estimación teórica de la mejora obtenida con cada técnica. Según esos cálculos, la técnica de inyección que logra una mayor aceleración depende de la relación existente entre la longitud del banco de pruebas y el número de elementos de memoria del circuito, así como, del porcentaje de los fallos inyectados que son silenciosos. Con el objetivo de analizar la influencia de la longitud del banco de prueba en el proceso de inyección (clasificación de fallos, incremento de área y velocidad de emulación) se aplican distintos bancos de pruebas, en concreto se utilizan dos conjuntos de estímulos en cada circuito, con distintas longitudes. Los resultados experimentales permitirán medir la influencia de todas las aproximaciones propuestas, comprobando la estimación realizada y evaluando el efecto de la clasificación rápida de los fallos silenciosos.

Por otro lado, se evalúan circuitos reales. Estos experimentos tienen como objetivo demostrar la capacidad del sistema de inyección propuesto en este trabajo de tesis doctoral para evaluar de forma rápida, eficiente y con un bajo coste, un considerable número de fallos en circuitos complejos. A continuación se presentan los resultados obtenidos.

6.1.3.1 Clasificación de fallos

Los fallos se han clasificado en cuatro categorías distintas. Cuando el fallo provoca un error en las salidas del circuito se clasifica como *avería*, en caso contrario se distingue entre los fallos cuyos efectos desaparecen por completo, clasificados como fallos *silenciosos*, y los que permanecen almacenados en el circuito, que se clasifican como *latentes* si el error está en los biestables o como *latentes en RAM* si el error afecta únicamente a la memoria empotrada disponible en el circuito. De todos los circuitos evaluados sólo el microprocesador LEON2 dispone de memoria empotrada por lo que sólo en ese caso tiene sentido considerar los fallos de tipo *latentes en RAM*.

La Tabla 4 recoge la clasificación de fallos obtenida al aplicar el sistema de Emulación Autónoma en los distintos circuitos. Para cada circuito se muestra el número de ciclos de ejecución del banco de pruebas, el número total de fallos inyectados y el porcentaje de fallos clasificado en cada una de las categorías consideradas. Para los circuitos de prueba b12, b14, b14_TMR y b15 se muestran los resultados obtenidos aplicando dos bancos de pruebas distintos, uno con 160 ciclos de ejecución y el otro con 600 ciclos. El diccionario de fallos obtenido tras la inserción de SEUs en biestables es independiente de la técnica de inyección aplicada, aunque la técnica *Mask-Scan*, según se describió en el capítulo 4, no permite distinguir entre fallos silenciosos y latentes, por

lo que, proporciona un menor detalle en la clasificación. En el caso de la inyección en memorias, se han realizado campañas de inyección sobre el microprocesador LEON2 con los dos modelos de memoria propuestos, el modelo básico y el modelo ECAM descritos en el apartado 4.5, para estudiar la precisión obtenida con cada uno. Nótese que en un circuito real y complejo como es el microprocesador LEON2 se han inyectado doce millones de fallos, caracterizando el circuito de forma completa para una aplicación determinada.

Circuito		#Ciclos de ejecución	#Fallos inyectados	Clasificación de fallos			
				%Silencioso	%Avería	%Latente	%Latente en RAM
b12		160	19.040	14,6	29,8	55,6	-
		600	71.400	13,3	33,3	53,4	-
b14		160	34.400	46,4	49,2	4,4	-
		600	129.000	38,5	59,6	1,9	-
b14_TMR		160	51.680	80,7	16,0	3,2	-
		600	193.800	75,7	23,0	1,3	-
b15		160	66.880	27,4	19,9	52,7	-
		600	250.800	26,8	18,6	54,6	-
CORDIC16		150.000	129.750.000	16.07	83.92	0.01	-
CIRCUIT_A		1.500	726.000	31,0	21,4	47,7	-
CIRCUIT_A_TMR		1.500	873.000	47,2	14,1	38,7	-
LEON2	Básico	3.800	12.129.600	32,6	7,3	59,6	0,4
	ECAM			35,1	8,8	43,6	12,5

Tabla 4. Clasificación de fallos obtenida en cada experimento de inyección de fallos.

Los circuitos de prueba b12 y b15 presentan una mayoría de fallos que permanecen latentes en el circuito para el conjunto de estímulos aplicado. En el circuito b14 los fallos producen averías o desaparecen, siendo menos de un 5% el número de fallos latentes. El circuito CORDIC16 tiene una estructura segmentada, de forma que los fallos se transmiten a través del circuito hasta las salidas, produciéndose un 83,92% de averías en la campaña de inyección. Con respecto a los circuitos industriales CIRCUITO_A y LEON2, se observa un considerable número de fallos latentes. Esto se debe a que son circuitos complejos y la campaña de inyección realizada considera un conjunto relativamente pequeño de estímulos, que no activan la funcionalidad completa de dichos circuitos.

En las versiones endurecidas con TMR de los circuitos b14 y CIRCUITO_A el número de posibles fallos simples aumenta puesto que se ha incrementado el número de los elementos de memoria susceptibles de sufrir un SEU. La inyección de un fallo en cada uno de los componentes que conforman un módulo triplicado genera efectos

equivalentes, por lo que sólo es necesario inyectar en uno de ellos para obtener la clasificación total. Sin embargo, las clasificaciones obtenidas para b14_TMR y CIRCUITO_A_TMR se presentan en porcentaje con respecto al número total de posibles fallos con el objetivo de obtener la probabilidad total de que el fallo produzca un efecto determinado. Se puede observar que en las versiones endurecidas de los circuitos se aumenta significativamente el número de fallos silenciosos, disminuyendo tanto las averías como los fallos latentes, por lo que se comprueba la mejora conseguida en la tolerancia a fallos del circuito endurecido con respecto a la versión original.

Los resultados ilustran la dependencia existente entre la clasificación de fallos y el conjunto de estímulos aplicado. Generalmente, el número de fallos latentes disminuye para bancos de prueba con mayores longitudes, puesto que esto conlleva normalmente una mayor actividad de las distintas partes del circuito.

El microprocesador LEON2 incluye memorias empotradas. En el capítulo 4, se presentaron dos posibles modelos de memoria compatibles con el sistema de Emulación Autónoma, el modelo básico y el modelo ECAM. Como se puede observar en la Tabla 4, la clasificación de fallos obtenida depende del modelo de memoria utilizado El modelo de memoria básico consiste en replicar la memoria para realizar la ejecución con y sin fallo alternativamente, de acuerdo con la técnica *Time-Multiplexed*, mientras que en el modelo ECAM la memoria de fallo almacena únicamente las direcciones de la memoria que contienen fallo así como el dato correspondiente. El modelo básico es muy sencillo y requiere menos lógica en su implementación que el modelo ECAM como se verá en los datos de área que se presentan en el apartado 6.1.3.3. Sin embargo, el modelo básico es menos preciso que el modelo ECAM y la clasificación de fallos obtenida con sendos modelos difiere. Las diferencias encontradas en la clasificación de fallos y sus causas se explican a continuación:

- El porcentaje de fallos silenciosos que se obtiene es menor utilizando el modelo básico que en el caso de aplicar la memoria ECAM. El modelo básico no permite detectar la cancelación de fallos en memoria porque no se almacena la dirección afectada por un fallo. Sin embargo el modelo ECAM almacena la dirección de la palabra con fallo y el dato, de forma que la cancelación del fallo se detecta cuando se accede a dicha posición y se reescribe el dato con un valor correcto.
- El porcentaje de fallos *latentes en RAM* y el porcentaje de averías disminuyen, mientras que el número de fallos clasificados como latentes aumenta notablemente si se utiliza el modelo de memoria básico. La causa de estas tres discrepancias es la misma y como se explica a continuación se debe a la falta de precisión del modelo básico. En los experimentos de inyección con memorias

empotradas no se ha aplicado la restauración del estado, y la ejecución comienza desde el principio del banco de pruebas para cada fallo. Suponiendo que ninguna posición de memoria se lee antes de ser escrita la única diferencia esperada en la clasificación de fallos para ambos modelos sería para los fallos silenciosos. Sin embargo, esa hipótesis no se cumple en el LEON2 y hay posiciones de memoria que se leen antes de escribirse. Al aplicar el modelo ECAM, la memoria de fallos está vacía al comenzar la emulación de un fallo y los datos que se leen de la memoria son correctos, se leen de la memoria *golden*, hasta que se produce la inserción del fallo. Por otra parte, en el modelo básico, la memoria con fallo almacena el dato erróneo de una emulación a otra. El dato con fallo se lee antes de ser escrito de nuevo, introduciendo en el circuito errores adicionales o, por el contrario, cancelando la inserción del nuevo fallo. Por lo tanto, las diferencias de averías y latentes se deben a que no se cumple una de las hipótesis de partida planteadas en el apartado 4.5.

En consecuencia, el modelo ECAM es más preciso que el básico, pero si se cumplen las hipótesis de partida, las diferencias se encuentran únicamente en el número de fallos silenciosos y *latentes en RAM*, por lo que el modelo básico es adecuado si el principal interés se centra en la tasa de avería.

Localización del fallo	#Elementos de memoria	#Fallos inyectados	Clasificación de fallos			
			%Silencioso	%Avería	%Latente	%Latente en RAM
Unidad entera	1.284	4.879.200	50,89	11,71	36,57	0,82
Controlador de la memoria	209	794.200	60,80	8,14	30,80	0,26
Reset	6	22.800	0,04	98,93	1,03	0,00
Controlador de interrupciones	64	243.200	25,03	2,50	72,47	0,00
Unidad de depuración	473	1.497.400	7,63	0,02	92,34	0,00
Periféricos, temporizadores	456	1.732.800	28,98	0,00	71,02	0,00
Otros biestables	88	334.400	29,60	3,34	37,06	0,00
Memoria	-	2.325.600	16,87	16,80	2,99	63,35

Tabla 5. Clasificación de fallos obtenida en el circuito LEON2 en función de la localización del fallo, cuando se aplica el modelo de memoria ECAM.

Un análisis de la clasificación de fallos en función de su localización para el LEON2 se presenta en la Tabla 5. Este análisis se ha realizado utilizando el modelo de memoria ECAM. La tabla recoge cada componente evaluado, el número de elementos de memoria correspondiente y la clasificación obtenida. En el caso de la memoria empotrada, la inyección se realiza en las posiciones a las que se accede en cada ciclo de

ejecución, puesto que en el resto los fallos permanecen latentes. De esta forma, no es necesario inyectar en todas las localizaciones posibles para obtener la clasificación de todos los fallos.

La campaña de inyección realizada para evaluar cada componente del circuito es exhaustiva. Los porcentajes mostrados son datos relativos al número total de fallos inyectados en una localización particular, así por ejemplo, el 50,89% de los fallos insertados en la unidad entera del microprocesador son fallos silenciosos. La lógica de reset presenta un 98,93% de averías por lo que es un componente crítico, sin embargo, la probabilidad de que un SEU afecte a este módulo es mucho menor que en el resto de componentes debido al reducido número de elementos sensibles. Por otro lado, los periféricos no son sensibles puesto que no se utilizan en la aplicación ejecutada en los experimentos. Según los resultados obtenidos las zonas más críticas son la memoria y la unidad entera.

6.1.3.2 Velocidad del proceso de emulación de fallos

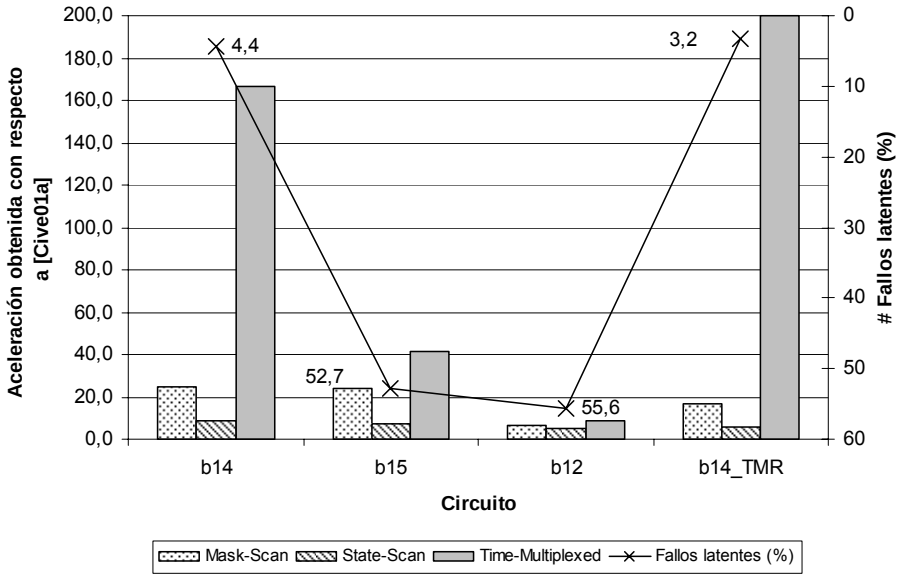
Uno de los objetivos principales de este trabajo de tesis es acelerar la inyección de fallos. La Tabla 6 muestra los resultados de tiempo obtenidos correspondientes a cada circuito evaluado. Los experimentos se han realizado a una frecuencia de reloj de 25MHz y en la tabla se muestra el tiempo medio necesario para evaluar un fallo.

Circuito		#Ciclos de ejecución	Tiempo medio de inyección (µs/fallo)		
			Mask-Scan	*State-Scan*	*Time-Multiplexed*
b12		160	5,7	7,5	4,2
		600	20,3	10,8	13,6
b14		160	4,1	11,2	0,6
		600	17	16	0,6
b14_TMR		160	6,0	16,3	0,5
		600	21,6	23,1	0,5
b15		160	5,8	19,9	3,4
		600	21,8	27,1	13,1
CORDIC16		150.000	-	-	5,4
CIRCUITO_A		1.500	53,0	43,9	32,8
CIRCUITO_A_TMR		1.500	-	-	26,7
LEON2	Básico	3.800	-	-	240,3
	ECAM		-	-	138,3

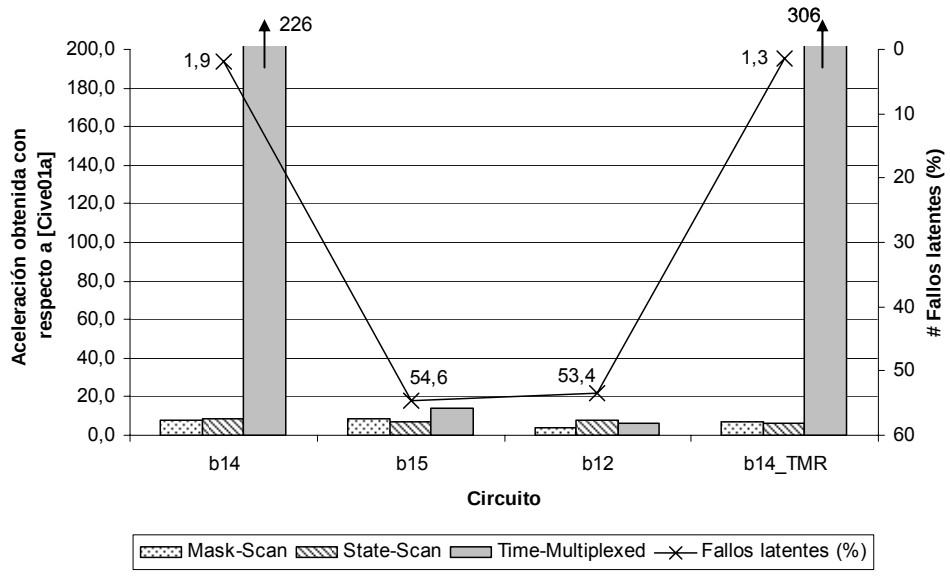
Tabla 6. Tiempo medio empleado en la campaña de inyección realizada para cada circuito.

Los circuitos de prueba se han evaluado con las tres técnicas propuestas para comparar sus características, aplicando dos bancos de pruebas distintos. En [Cive01a] se presentan resultados de la inyección de fallos en el circuito b14 con 100 ciclos de

ejecución, obteniendo una tasa de inyección de 100µs/fallo. En la Tabla 6 se muestra que aplicando Emulación Autónoma se consigue una tasa de inyección de hasta 0,6µs/fallo, por lo que el sistema de inyección propuesto supone una aceleración de prácticamente dos órdenes de magnitud con respecto a técnicas de emulación propuestas en la literatura que a su vez aceleraban la inyección hasta cuatro órdenes de magnitud con respecto a la simulación. En la Figura 69 se representa gráficamente la mejora obtenida en la velocidad con las técnicas de inyección basadas en Emulación Autónoma con respecto a las tasas de inyección logradas mediante [Cive01a].



a)Aceleración obtenida para un banco de pruebas de 160 ciclos



b) Aceleración obtenida para un banco de pruebas de 600 ciclos

Figura 69. Aceleración obtenida con las técnicas de inyección basadas en Emulación Autónoma con respecto a [Cive01a]

La Figura 69 ilustra como influye el número de fallos latentes en la velocidad alcanzada por las distintas técnicas. Las técnicas de inyección propuestas permiten la clasificación temprana de averías y/o fallos silenciosos, pero en el caso de fallos latentes no se incluyen optimizaciones. Por lo tanto, el efecto de las optimizaciones implementadas es mayor en circuitos con un bajo porcentaje de fallos latentes. Comparando las tres técnicas de inyección propuestas, se observa que la técnica *Time-Multiplexed* proporciona una mayor aceleración que las otras dos soluciones, excepto para el circuito b12 y un banco de pruebas de 600 ciclos de reloj. Esta mejora se debe a que la técnica *Time-Multiplexed* permite detectar rápidamente tanto averías como fallos silenciosos. Los circuitos con estructuras tolerantes a fallos como el b14_TMR y CIRCUITO_A_TMR, presentan un número elevado de fallos silenciosos y el tiempo medio empleado por fallo disminuye con respecto a la versión del circuito sin endurecer al utilizar *Time-Multiplexed*. En el caso en que la mayor parte de los fallos es latente, la emulación del fallo debe ejecutarse hasta el final del banco de pruebas, por lo que no se obtiene beneficio de las optimizaciones implementadas y el tiempo medio de inyección aumenta en comparación con los circuitos en los que se obtienen porcentajes considerables de averías y silenciosos. En el circuito b12, más de la mitad de los fallos son latentes y el número de fallos silenciosos supone menos del 15% por lo que en este caso el efecto de las optimizaciones es considerablemente menor que en el resto de circuitos evaluados.

En relación con la influencia de la longitud del banco de pruebas, los resultados obtenidos muestran que el tiempo medio de inyección aplicando la técnica *Time-Multiplexed* aumenta cuando aumenta el número de ciclos de ejecución. La razón es que dicha técnica de inyección alterna las ejecuciones con y sin fallo por lo que se utilizan dos ciclos de reloj por cada ciclo del banco de pruebas. De esta forma, para fallos latentes, donde se ejecuta el banco de pruebas hasta el final, se introduce un retardo con respecto a la aplicación de las otras dos técnicas. Este es el caso de los circuitos b12 y b15. Por otro lado, se observa que la técnica *State-Scan* es más rápida que la *Mask-Scan* sólo cuando se cumple que $C > 2 \cdot F$, donde C es el número de ciclos del banco de pruebas y F es el número de biestables del circuito, tal como se estimó en el apartado 4.4.4.

Por lo tanto, el tiempo medio para la inyección de fallos dependerá de la clasificación de fallos. El beneficio obtenido de las distintas optimizaciones implementadas depende de la clasificación de fallos y, por esta razón, la elección de la técnica más rápida debe realizarse de acuerdo a la clasificación esperada. En circuitos tolerantes a fallos se espera un alto número de fallos silenciosos por lo que en principio siempre es más adecuada la técnica *Time-Multiplexed*. Si se espera que la mayor parte de los fallos sea latente, por ejemplo cuando no se ejercita la funcionalidad completa del

circuito a evaluar, es más adecuado utilizar *State-Scan*. En el CIRCUITO_A se ha realizado el proceso de evaluación aplicando las tres técnicas de inyección propuestas, resultando la más rápida la técnica *Time-Multiplexed*. Por esta razón, para el CIRCUITO_A_TMR únicamente se ha realizado el experimento utilizando dicha técnica, puesto que la inserción de triplicación modular enmascara fallos y aumenta el número de fallos silenciosos.

Con respecto al circuito LEON2, con memorias empotradas, el uso del modelo de memoria ECAM proporciona mayor velocidad en el proceso de inyección gracias a la detección de los fallos silenciosos en la memoria.

6.1.3.3 Resultados en área

En este apartado se presentan los recursos necesarios para implementar el sistema de Emulación Autónoma utilizando cada una de las tres técnicas propuestas. El estudio de los recursos necesarios junto con las características proporcionadas por cada técnica, en cuestión de precisión y velocidad, permite evaluar el compromiso entre coste y prestaciones alcanzado en cada caso. Las conclusiones obtenidas a partir del análisis de los resultados permitirán seleccionar la técnica de inyección más adecuada que se ajuste al dispositivo de emulación disponible ofreciendo las mayores prestaciones. En caso de que el circuito sea complejo, con una gran cantidad de elementos lógicos, pueden utilizarse las técnicas con una menor necesidad de recursos lógicos.

Como se explicó al comienzo de este capítulo, se han utilizado dos plataformas FPGAs distintas durante la realización de los experimentos. Los dos dispositivos FPGA utilizan diferentes tecnologías y tienen arquitecturas internas distintas. Por lo tanto, los recursos necesarios para la implementación de un circuito en sendas FPGAs no son comparables. Todos los circuitos, a excepción del microprocesador LEON2 se han evaluado en la placa RC1000. La placa de evaluación de Virtex-4 se ha utilizado para emular fallos en el LEON2, tanto para el modelo de memoria básico como para el modelo ECAM.

La Tabla 7 recoge los resultados de área en términos de biestables necesarios para la implementación del sistema de inyección. Para cada una de las tres técnicas de inyección propuestas se muestra el número de biestables que conforman el sistema de Emulación Autónoma, así como el porcentaje que supone, relativo al número de biestables del circuito original, es decir, del circuito sin instrumentar.

El número de biestables adicionales se corresponden con los biestables del instrumento y los necesarios para implementar el control del emulador. La técnica más sencilla, y por lo tanto, la que menos lógica requiere es la técnica *Mask-Scan*, mientras que el uso de la técnica *Time-Multiplexed* implica un incremento en los biestables utilizados de alrededor del 300% en circuitos sin memorias empotradas. Los

experimentos realizados con el LEON2 muestran que el modelo de memoria ECAM requiere de alrededor de 850% de incremento en términos de biestables, con respecto al circuito original, para su implementación. Mientras que si se utiliza el modelo básico el incremento de área es del 483%. Esta diferencia en la cantidad de recursos necesarios se debe principalmente al control de la memoria ECAM.

Circuito		#ciclos	#FFs (%)			
			Original	*Mask-Scan*	*State-Scan*	*Time-Multiplexed*
b12		160	119	327 (174,8%)	365 (206,7%)	564 (373,9%)
		600		331 (178,2%)	372 (212,6%)	568 (377,3%)
b14		160	215	520 (141,9%)	539 (150,7%)	1.077 (400,9%)
		600		524 (143,7%)	545 (153,5%)	1.035 (381,4%)
b14_TMR		160	323	785 (143,0%)	804 (148,9%)	1.483 (359,1%)
		600		789 (144,3%)	810 (150,8%)	1.473 (356,0%)
b15		160	418	962 (130,1%)	991 (137,1%)	1.919 (359,1%)
		600		966 (131,1%)	998 (138,8%)	1.926 (360,8%)
CORDIC16		150.000	865	-	-	4.182 (483,4%)
CIRCUITO_A		1.500	484	1.102 (127,7%)	1.099 (127,1%)	2.038 (321,1%)
CIRCUITO_A_TMR		1.500	582	1.378 (136,8%)	1.297 (122,9%)	2.456 (322,0%)
LEON2	Básico	3.800	2.580	-	-	12.470 (483,3%)
	ECAM			-	-	21.969 (851,5%)

Tabla 7. Resultados de área requerida en términos de biestables para la implementación del sistema de Emulación Autónoma.

La Tabla 8 contiene los datos relativos al número de LUTs empleadas para implementar el sistema de inyección en función de la técnica utilizada. De igual forma que para los biestables, la técnica *Time-Multiplexed* implica el mayor incremento de área. Aunque los incrementos son menores que para el caso de los biestables, en términos absolutos se utilizan más LUTs que biestables, por lo que el factor limitante para implementar el sistema de inyección de un circuito dado vendrá impuesto por el número de LUTs. Es por esto, que se comparte la lógica combinacional en la técnica *Time-Multiplexed* en vez de replicar por completo el circuito bajo evaluación. También

en términos de LUTs el modelo de memoria ECAM requiere de un mayor número de recursos que el modelo básico.

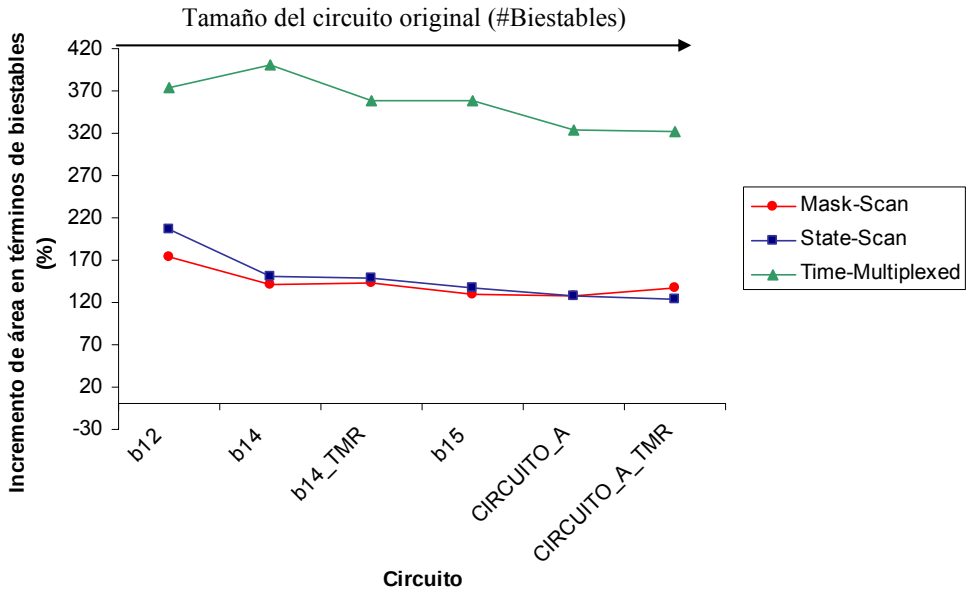
En el caso del circuito b14_TMR se observa como el circuito emulador completo ocupa menor número de LUTs que el circuito original para las técnicas *Mask-Scan* y *State-Scan*. Esto ocurre para este caso particular porque la herramienta de síntesis agrupa algunas funciones lógicas del sistema emulador, optimizando el uso de área con respecto al circuito original.

Circuito		#ciclos	#LUTs (%)			
			Original	*Mask-Scan*	*State-Scan*	*Time-Multiplexed*
b12		160	362	710 (96,1%)	758 (109,4%)	1.481 (309,1%)
		600		716 (97,8%)	773 (113,5%)	1.496 (313,3%)
b14		160	1.172	1.586 (35,3%)	1.684 (43,7%)	4.255 (263,1%)
		600		1.592 (35,8%)	1.694 (44,5%)	4.291 (266,1%)
b14_TMR		160	2.126	2.073 (-2,5%)	2.099 (-1,3%)	4.861 (128,6%)
		600		2.079 (-2.2%)	2.110 (-0,8%)	4.702 (121,2%)
b15		160	2.322	3.523 (51,7%)	3.315 (42,8%)	7.845 (237,9%)
		600		3.529 (52,0%)	3.328 (43,3%)	7.793 (235,6%)
CORDIC16		150.000	1.681	-	-	7.064 (420,2%)
CIRCUITO_A		1.500	932	1.821 (95,4%)	1.831 (96,5%)	4.798 (414,8%)
CIRCUITO_A_TMR		1.500	955	2.070 (116,8%)	2.090 (118,8%)	5.184 (442,8%)
LEON2	Básico	3.800	8.630	-	-	23.241 (269,3%)
	ECAM			-	-	45.918 (532,1%)

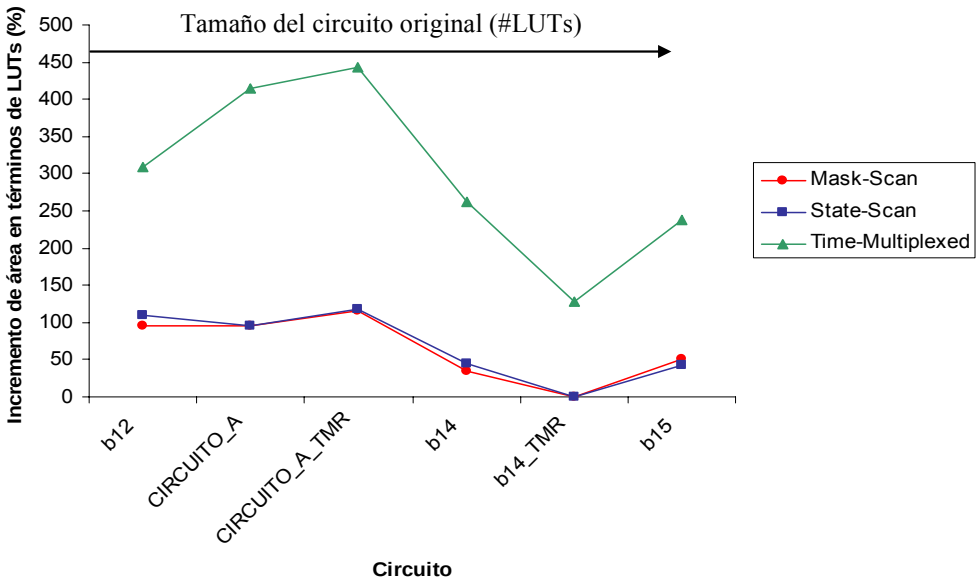
Tabla 8. Resultados de área en términos de LUTs requeridas para la implementación del sistema de Emulación Autónoma.

En la Figura 70 se representa gráficamente el incremento de área necesario para implementar cada técnica de inyección propuesta, en términos de biestables y LUTs. La figura muestra el porcentaje de recursos adicionales correspondiente a cada técnica y a cada circuito. La técnica *Time-Multiplexed* es la técnica que más recursos consume. Con respecto al porcentaje de biestables, se observa que el incremento relativo al número de

biestables del circuito original disminuye ligeramente cuando el tamaño del circuito es mayor. Sin embargo, esta tendencia no se observa en los datos relativos al número de LUTs, puesto que la implementación de las funciones lógicas en la FPGA depende de la herramienta de síntesis y de la propia lógica a implementar.



a) Incremento de área en términos de biestables



b) Incremento de área en términos de LUTs

Figura 70. Incremento de área necesario para la implementación de las distintas técnicas de inyección propuestas en esta tesis

Por otra parte, en las técnicas *Mask-Scan* y *State-Scan* es necesaria cierta memoria para almacenar los resultados esperados, utilizados en la clasificación de fallos como averías. Además, la técnica *State-Scan* se basa en la inyección del fallo mediante la introducción, a través de una cadena de *scan*, del estado con fallo directamente en los biestables. Estos estados se almacenan también en memoria para permitir una Emulación Autónoma y evitar la comunicación entre el *hardware* de emulación y el PC. Los requisitos de memoria se presentan en la Tabla 9. Evidentemente, la técnica *State-Scan* implica las mayores necesidades en términos de memoria. Las plataformas FPGA actuales disponen de memorias con varios MB de capacidad, por lo que estos requisitos no suponen un inconveniente.

Los datos mostrados en Tabla 9 no incluyen la memoria que sería necesaria para almacenar el banco de pruebas o el diccionario de fallos porque no son requisitos imprescindibles para la aplicación de las distintas técnicas de inyección. Los vectores de entrada del circuito pueden almacenarse de forma comprimida o generarse mediante una lógica dedicada a dicha función. Por otra parte, el diccionario de fallos requeriría $N \cdot 2$ bits de memoria, donde N es el número de fallos inyectados y siendo necesarios dos bits para su clasificación en cuatro categorías distintas. Sin embargo, si es interesante más información, como por ejemplo el tiempo que tarda el fallo en propagarse, o menos, considerándose únicamente el número total de fallos pertenecientes a cada categoría, las necesidades de memoria serían distintas.

Circuito	#ciclos	Memoria (Kbits)		
		Mask-Scan	*State-Scan*	*Time-Multiplexed*
b12	160	0,9	2.213,6	0
	600	3,5	8.301,0	0
b14	160	8,4	7.231,1	0
	600	31,6	27.116,6	0
b14_TMR	160	8,4	16.309,8	0
	600	31,6	61.161,9	0
b15	160	10,9	27.311,6	0
	600	41,0	102.418,4	0
CORDIC16	150.000	-	-	0
CIRCUITO_A	1.500	99,6	343.248,0	0
CIRCUITO_A_TMR	1.500	99,6	496.277,3	0

Tabla 9. Resultados en área en términos de Kbits de memoria requerida para implementar el sistema de Emulación Autónoma en circuitos sin memorias empotradas.

En el caso de circuitos con memorias empotradas, el modelo básico requiere el doble de bloques de memoria, puesto que consiste en duplicar los bloques originales. Por otro lado, el modelo ECAM implica un incremento en bloques de memoria nulo,

porque la memoria encargada de almacenar las posiciones con fallo se implementa mediante biestables y LUTs.

6.2 Inyección en microprocesadores comerciales

En este apartado se presentan los experimentos de inyección de fallos realizados sobre circuitos microprocesadores, utilizando el método propuesto en el capítulo 5. Se han desarrollado dos conjuntos de experimentos. Por una parte, se ha realizado una campaña de inyección sobre un microprocesador ARM7-S para evaluar el sistema de inyección propuesto basado en el uso de la estructura de depuración integrada. Por otro lado, se han realizado experimentos de inyección de fallos sobre un microprocesador PowerPC integrado en una FPGA de Xilinx® Virtex-II Pro con el objetivo de evaluar la viabilidad del sistema de inyección para SoPCs, propuesto y descrito en el capítulo 5.

6.2.1 Descripción de los circuitos de prueba

Los circuitos microprocesadores utilizados en los experimentos de inyección de fallos son microprocesadores complejos, de grandes prestaciones y comúnmente utilizados en la industria. A continuación se describen las características fundamentales de ambos circuitos.

6.2.1.1 ARM7

La técnica de inyección propuesta en el capítulo 5 se ha utilizado para realizar una campaña de inyección en un microcontrolador basado en un procesador ARM. El microcontrolador utilizado es el LPC2129 de NXP Semiconductors [LPC2129] que contiene un procesador ARM7TDMI-S [ARM7] con 16 KB de memoria SRAM y 256 KB de memoria *flash* en el chip del microcontrolador. El ARM7TDMI-S en un microprocesador de propósito general de 32-bits, segmentado en tres etapas y de tipo RISC (*Reduced Instruction Set Computer*). Este procesador tiene una arquitectura de Von Neumann por lo que tiene una única memoria para datos e instrucciones.

El ARM7TDMI-S contiene una estructura de depuración integrada que incluye las capacidades básicas de depuración e incluso soporta depuración en tiempo real y se comunica con el exterior mediante una interfaz JTAG. Sin embargo, el sistema de inyección no utiliza las capacidades de depuración avanzadas porque el propósito que se persigue consiste en obtener un sistema de inyección de fallos transitorios lo más general posible y con el menor número de requisitos necesarios. Utilizando únicamente las capacidades básicas de depuración el sistema de inyección propuesto puede aplicarse a cualquier procesador con OCD e interfaz JTAG.

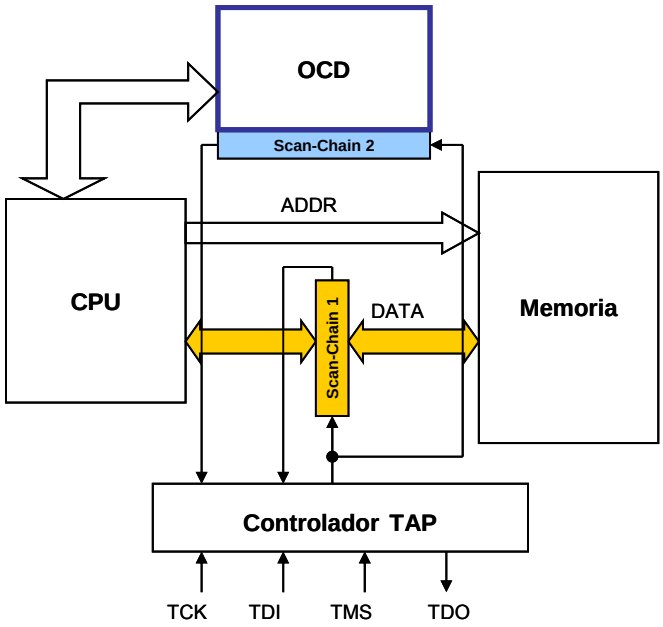


Figura 71. Arquitectura de la lógica integrada en el microprocesador ARM7-S.

La estructura de la lógica integrada en el microprocesador se representa en la Figura 71. Como puede observarse en la figura, el microprocesador dispone de dos cadenas de *scan* conectadas con la interfaz JTAG. Estas cadenas se corresponden con registros de datos del estándar JTAG, de forma que mediante la instrucción adecuada se puede seleccionar cualquiera de las cadenas para escribir o leer datos. La cadena número 1 está conectada al bus de datos del procesador. Leer la cadena 1, permite conocer los datos que se escriben en memoria o aquellos que lee el procesador, como por ejemplo las instrucciones a ejecutar. Una escritura en la cadena número 1 permite ejecutar instrucciones en el procesador, como la lectura o escritura de un registro. Por otra parte, la cadena de *scan* número 2 está conectada a la estructura de depuración lo que permite controlar todas las capacidades de depuración disponibles mediante la ejecución de la secuencia de instrucciones JTAG adecuada. La Tabla 10 muestra el conjunto de instrucciones JTAG públicas disponibles en el microprocesador ARM7TDMI-S.

Instrucción JTAG	Código binario
SCAN_N	0010
INTEST	1100
IDCODE	1110
BYPASS	1111
RESTART	0100

Tabla 10. Conjunto de instrucciones JTAG públicas para el microprocesador ARM7TDMI-S [ARM7]

La instrucción SCAN_N permite seleccionar el registro de datos. Cuando la instrucción INTEST se introduce en el registro de instrucción la cadena seleccionada

entra en modo de pruebas, de forma que pueden escribirse y leerse datos de dicha cadena. La instrucción IDCODE conecta la interfaz JTAG con el registro de datos que contiene el identificador del dispositivo. La instrucción BYPASS conecta un registro de un único bit entre el conector de entrada y el de salida. Cuando esa instrucción es ejecutada el sistema no se ve afectado y continúa con su operación normal. Por último, la instrucción RESTART reinicia la ejecución del procesador, sacándolo del modo de depuración.

La aplicación *software* utilizada en las pruebas con este microprocesador consiste en una multiplicación de matrices de dimensión 9x9. Se considera que el resultado de la aplicación es la matriz producto, almacenada en la memoria SRAM disponible en el microcontrolador. Todos los registros y palabras de memoria SRAM son sensibles, con igual probabilidad, de sufrir un SEU. Con respecto a la memoria, solo se inyectan fallos en aquellas posiciones que se usan durante la aplicación, en este caso en aquellas posiciones utilizadas para almacenar las distintas matrices y los resultados intermedios requeridos. Un fallo en cualquiera de las palabras de memoria restantes no se propaga a través del circuito y por lo tanto no tendrían un efecto visible a las salidas. El código de la aplicación se almacena en memoria *flash.* Se asume que la memoria *flash* no es sensible a fallos SEU y por lo tanto no se inyectan *bit-flips* en la memoria de instrucciones.

6.2.1.2 PowerPC (VirtexII-Pro)

Como ya se dijo en el capítulo 5, un SoPC es un sistema que integra distintos componentes, entre los que se encuentran uno o varios microprocesadores, y lógica programable. La inyección de fallos en el microprocesador integrado en un SoPC es un caso particular de la inyección de fallos en microprocesadores. En el caso de un SoPC, es posible utilizar la propia lógica programable del sistema para implementar funciones de inyección de forma que se facilita el acceso al microprocesador. En el capítulo 5, se propone un sistema de inyección que, además de utilizar las capacidades de depuración del microprocesador para acceder a los recursos internos, acelera el proceso de evaluación y lo optimiza mediante el uso de la lógica programable disponible en el SoPC.

Con el método de inyección de fallos propuesto en este trabajo de tesis doctoral para SoPCs, se han realizado experimentos de inyección sobre una FPGA Virtex-II Pro de Xilinx® [V2 Pro] que contiene un microprocesador PowerPC 405 [PowerPC] integrado. El PowerPC 405 es un microprocesador de 32-bits, contiene 16KB de memoria caché de instrucciones y 16KB de memoria caché para datos, 32 registros de propósito general y 49 de propósito especial, así como capacidades de depuración integradas, con una interfaz JTAG.

El SoPC utilizado contiene los siguientes componentes:

- Una memoria de 32KB.
- Un puerto serie RS-232 como periférico.
- Un controlador 10/100 MAC Ethernet.
- Un bloque para el control de una memoria externa de 1MB.

En la documentación que proporciona Xilinx® existen ocho instrucciones JTAG para depuración que, aunque se indican los códigos binarios, no están documentadas. En este caso, el control de las capacidades de depuración para el control de la campaña de inyección de fallos y la modificación de los elementos de memoria con la inserción de *bit-flip*s, se realiza mediante una herramienta *software*. De esta forma, el sistema de inyección aplicado al microprocesador ARM se comunica con el OCD mediante *hardware*, mientras que el sistema de inyección que evalúa el PowerPC utiliza herramientas *software*. Así, comparando los resultados obtenidos con ambos sistemas de inyección, se evalúa el efecto de implementar el entorno de inyección en *hardware*.

El programa *software* ejecutado por el PowerPC 405 durante la campaña de inyección es un diseño de referencia, documentado en [WebServer], que consiste en un servidor *web* con TCP/IP. Para ejecutar dicha aplicación es necesario conectar al SoPC una placa *hardware* adicional de comunicación Ethernet.

6.2.2 Método experimental utilizado

En este apartado se describe el protocolo definido para realizar una campaña de inyección de fallos en un microprocesador implementado en *hardware*, de acuerdo con el método propuesto en este trabajo de tesis doctoral. Se distinguen dos casos:

- La inyección en un microprocesador sobre su implementación final.
- La inyección en un microprocesador integrado en un SoPC.

El entorno de inyección desarrollado para cada caso es distinto, por lo que el protocolo a seguir y las herramientas necesarias se presentan por separado para cada caso.

6.2.2.1 Inyección de fallos en un microprocesador sobre un componente comercial

6.2.2.1.1 Protocolo definido

El sistema de inyección de fallos que se propone para evaluar la tolerancia a fallos de circuitos microprocesadores, de los cuales no se dispone de una descripción en un lenguaje *hardware* de alto nivel, se puede aplicar a cualquier microprocesador que

disponga de una estructura de depuración integrada con interfaz JTAG. El entorno de inyección es general, a excepción del bloque *hardware* que contiene las instrucciones JTAG necesarias para implementar cada uno de los comandos de depuración necesarios durante el proceso de inyección, y la lista de fallos. El protocolo definido para realizar la campaña de inyección de fallos se muestra en la Figura 72.

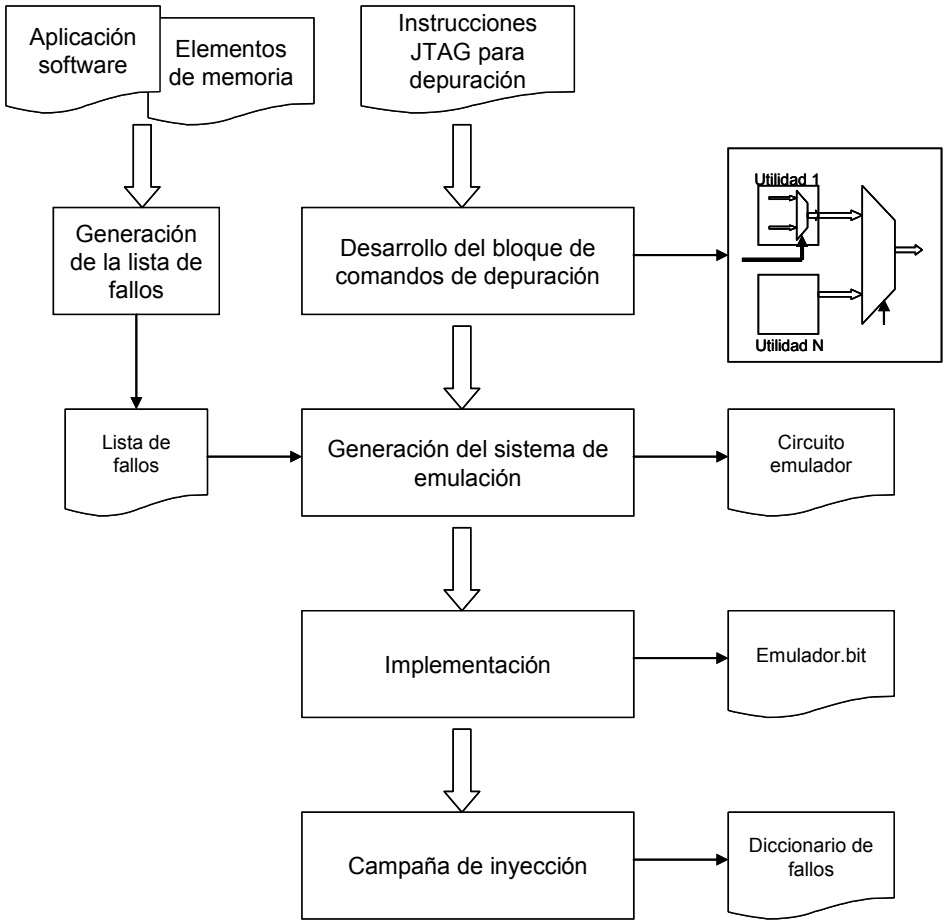


Figura 72. Protocolo definido para realizar la campaña de inyección de fallos basada en el uso del OCD con interfaz JTAG

En primer lugar, se desarrolla el bloque de comandos de depuración a partir de las instrucciones JTAG para depuración del microprocesador a evaluar. A diferencia del sistema de Emulación Autónoma, válido cuando se dispone de una descripción del circuito, en este caso la campaña de inyección no es exhaustiva y debe generarse la lista de fallos a partir de los posibles instantes de inyección, fijados por la aplicación *software* ejecutada, y los elementos de memoria susceptibles, determinados por el microprocesador a evaluar y el uso que hace la aplicación a ejecutar de sus recursos internos. A continuación, se integra la lista de fallos y el bloque de comandos en el sistema de inyección, que como se ha dicho anteriormente, es general para distintos procesadores. La descripción del sistema de inyección se implementa en una FPGA la cual se conecta mediante pines de entrada salida a la interfaz JTAG del microprocesador. La campaña de inyección se activa mediante una señal externa y el

diccionario de fallo se envía al PC mediante el puerto serie, a medida que los datos están disponibles.

6.2.2.1.2 Herramientas y recursos utilizados

El sistema de inyección basado en OCDs con interfaz JTAG se ha descrito en VHDL. La FPGA utilizada para su implementación es una placa de evaluación de la familia de FPGAs Spartan-3 de Xilinx®, que son dispositivos de bajo coste. La placa contiene una FPGA XC3S1000. La herramienta ISE de Xilinx® se ha usado para implementar el sistema de inyección y programarlo en la FPGA. Las pruebas y depuración del diseño se han realizado con la herramienta de simulación ModelSim de Mentor Graphics®.

6.2.2.2 Inyección de fallos en un microprocesador integrado en un SoPC

La evaluación de la tolerancia a fallos de un microprocesador integrado en un SoPC se puede considerar como un caso particular dentro de la evaluación de componentes microprocesadores comerciales. Los recursos internos del procesador son accesibles mediante el uso de las capacidades de depuración disponibles a través del interfaz JTAG. En el caso de un SoPC el sistema de inyección o algunas de sus funciones pueden implementarse en la lógica programable disponible en el propio sistema a evaluar. Se ha desarrollado un sistema de inyección para la evaluación de la tolerancia a fallos de un microprocesador integrado en el SoPC Virtex-II Pro. El sistema realizado consiste en un componente *hardware* implementado en el SoPC y otro componente *software* en el PC. A diferencia del sistema de inyección utilizado para evaluar el microprocesador ARM, en este caso, el control de las capacidades de depuración se realiza desde el PC por sencillez. El esquema del entorno de inyección desarrollado se presenta en la Figura 73.

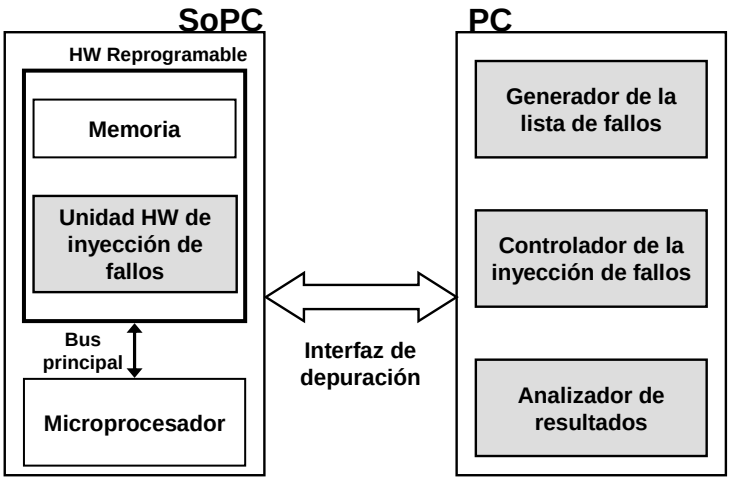


Figura 73. Estructura del entorno de inyección implementado para evaluar microprocesadores integrados en SoPC.

La generación de la lista de fallos, la clasificación de fallos y el control del proceso de inyección se realiza en el PC. El controlador de la inyección de fallos se encarga de las siguientes tareas:

1. Realizar una ejecución *golden* almacenando el resultado correcto de la aplicación objetivo.
2. Leer de la lista de fallos el tiempo de inyección y la localización dónde se debe insertar el fallo, enviando esta información al módulo *hardware* de inyección.
3. Se traspasa el control al módulo *hardware* de inyección hasta alcanzar el instante de inyección.
4. Inyectar el fallo a estudiar aplicando comandos de depuración mediante una herramienta *software*.
5. Se devuelve el control de la campaña de inyección al módulo *hardware* hasta que se alcanza el final de la aplicación.
6. Leer el resultado obtenido y comparar con el resultado que se obtuvo en la ejecución *golden*, clasificando el fallo en la categoría que corresponda.
7. Si quedan más fallos por insertar volver al paso 2.

La unidad *hardware* de inyección se implementa en la lógica programable disponible en SoPC. Este módulo tiene acceso a todas las señales del interfaz del microprocesador, denominado en la Figura 73 bus principal, lo que facilita la comunicación entre la unidad *hardware* de inyección y los distintos componentes del SoPC, como por ejemplo la memoria. De esta forma, el módulo de inyección es capaz de conocer la instrucción que ejecuta el microprocesador y los datos que se leen y se escriben en memoria directamente, sin necesidad de acceder a estos datos vía serie utilizando la interfaz JTAG. La unidad *hardware* de inyección realiza las siguientes tareas:

1. Espera la activación por parte del PC.
2. Una vez activado, mientras no se alcanza el instante de inyección, se cuentan los ciclos de reloj ejecutados.
3. Cuando se alcanza el instante de inyección la ejecución del procesador se detiene. Nótese, que esta acción se realiza desde el propio SoPC por lo que se evita el uso de la interfaz JTAG que es una interfaz serie.
4. El módulo *hardware* espera a que el PC le indique que la inserción del fallo ha finalizado.

5. Se desbloquea la ejecución del procesador que continúa con su operación normal hasta alcanzar el final de la aplicación. En caso de tratarse de una aplicación de control, dónde los resultados no se generan únicamente al final de la misma, se establecen puntos de comprobación intermedios.

La lectura de los resultados de la aplicación se acelera al aplicar un sistema de compresión de datos como el cálculo de una firma. En el experimento realizado se ha utilizado un registro de desplazamiento con múltiples entradas, MISR (*Multiple Input Shift Register*) para calcular la firma de los datos de salida de la aplicación *software*. De esta forma, sólo un dato debe ser transferido desde el SoPC al PC para determinar el efecto del fallo. El cálculo de la firma se realiza paralelamente a la ejecución de la campaña de inyección de fallos, según se generan los datos de salida.

6.2.2.2.1 Protocolo definido

La Figura 74 muestra el protocolo definido para realizar la campaña de inyección que permite la evaluación de la tolerancia a fallos de un microprocesador integrado en un SoPC.

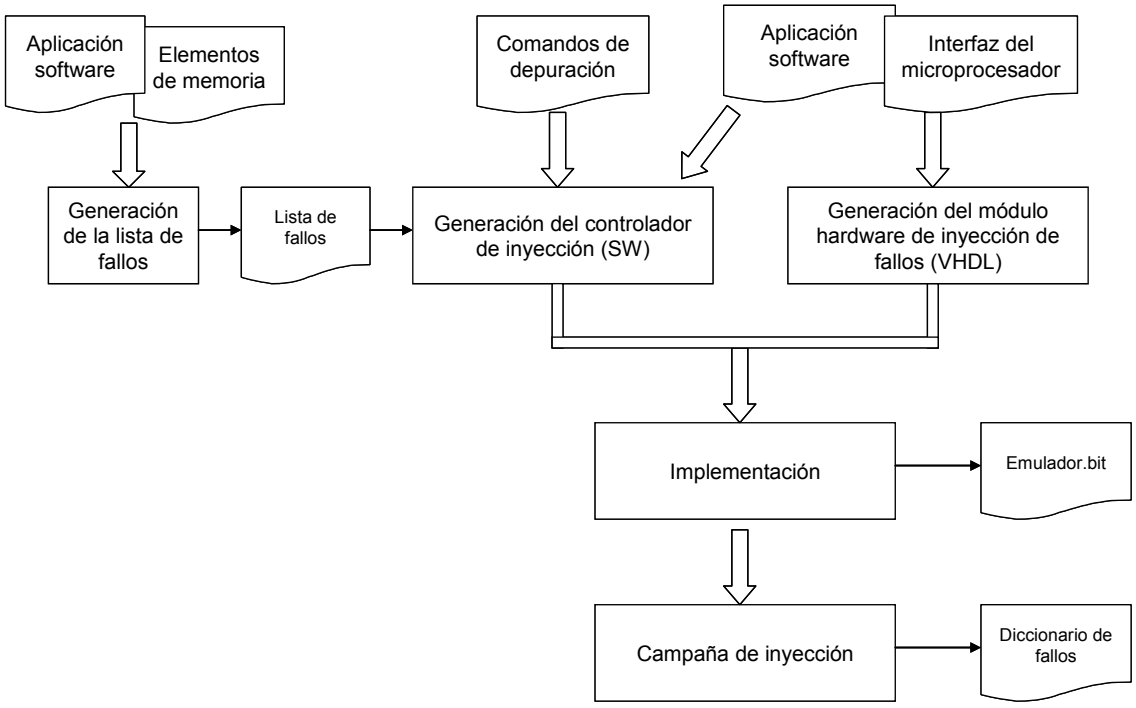


Figura 74. Protocolo para la evaluación de microprocesadores integrados en un SoPC.

El sistema de inyección consta de un componente *hardware* y otro *software*. El módulo *hardware* de inyección, descrito en VHDL, consiste en una máquina de control encargada del manejo de la ejecución del microprocesador en función de las órdenes recibidas desde el PC, y de la compresión de los datos que conforman el resultado de la aplicación ejecutada. La adaptación de este módulo para la evaluación de un microprocesador consiste básicamente en modificar el interfaz del módulo, de acuerdo

con el bus principal de microprocesador, y ajustar parámetros relativos a la aplicación *software* como la cantidad de datos que conforman los resultados.

La generación del controlador *software* de inyección se realiza a partir de los comandos de depuración necesarios para el acceso a los distintos recursos internos del microprocesador, y de la aplicación *software* ejecutada. En función de la aplicación objetivo se establece el tiempo máximo que determina cuándo se produce una pérdida de secuencia. El controlador de inyección se ha diseñado con el objetivo de que sea lo más general posible. La adaptación a un microprocesador distinto se reduce a modificar los parámetros relacionados con la aplicación y los comandos de depuración.

Una vez generado el sistema de inyección, la unidad *hardware* de inyección se integra en el SoPC y se inicia la campaña de inyección desde el PC.

6.2.2.2.2 Herramientas y recursos utilizados

El sistema de inyección propuesto consta de un componente *hardware* y otro *software*. Para el diseño y la implementación de la parte *hardware* del entorno de inyección se han utilizado las entornos ISE y EDK (*Embedded Development Kit*) de Xilinx®. La herramienta EDK consiste en un conjunto de herramientas *software* para el diseño completo de SoPCs, integrando tanto el desarrollo del componente *hardware* como la aplicación *software*. El código fuente de la aplicación se puede escribir en lenguajes de alto nivel como C, C++, o en lenguaje ensamblador. EDK proporciona las herramientas necesarias para la creación de estos ficheros, su compilación y generación de los ficheros ejecutables correspondientes. Los compiladores utilizados son compiladores GNU. XMD (*Xilinx Microprocessor Debug*) es la herramienta proporcionada por Xilinx® dentro de EDK, que conecta la plataforma *hardware* con el PC para su depuración. XMD proporciona un acceso completo a los registros de uso general y especial, a la memoria caché, de instrucciones y de datos, así como a la memoria principal del microprocesador. La interfaz de depuración con dicha herramienta puede realizarse mediante la herramienta GNU de depuración (GDB), o utilizando una interfaz TCL. La interfaz TCL permite ejecutar comandos de depuración mediante la aplicación de *scripts* y es el método que se ha utilizado para la inyección de fallos. El esquema de la relación entre las distintas herramientas para depuración utilizadas en el sistema de inyección desarrollado se presenta en la Figura 75. El componente *software* del sistema de inyección, esto es, el controlador de la inyección, consiste en un conjunto de ficheros TCL.

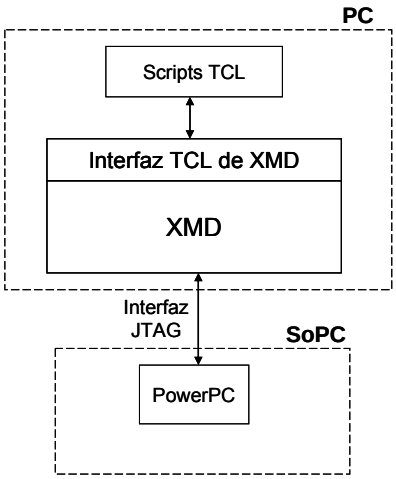


Figura 75. Esquema de las herramientas de depuración utilizadas.

El componente *hardware* del sistema de inyección se implementa en la lógica programable disponible en el SoPC a estudiar. Este componente requiere, como se verá en los resultados experimentales que se presentan más adelante, una pequeña cantidad de recursos lógicos con respecto a los recursos disponibles, debido a que parte del sistema de inyección se implementa en el PC. Por lo tanto, no se necesita una placa *hardware* adicional al propio sistema a evaluar.

6.2.3 Resultados experimentales

De igual modo que para el análisis del sistema de Emulación Autónoma, el sistema de inyección orientado a componentes microprocesadores comerciales se estudia en función de su precisión, coste y rapidez. Por lo tanto, los resultados experimentales considerados consisten en la clasificación de fallos obtenida, la velocidad del proceso de inyección y la cantidad de recursos necesarios para implementar el sistema de inyección.

Cada experimento consiste en la inyección de un número determinado de fallos, analizando los factores mencionados anteriormente. La lista de fallos se genera aplicando una distribución de probabilidad uniforme, tanto a las posibles localizaciones del fallo como a los instantes de inyección.

6.2.3.1 Clasificación de fallos

Los fallos inyectados en los microprocesadores ARM y PowerPC se clasifican en tres categorías distintas, **avería** si el fallo provoca un error en el resultado de la aplicación, **sin avería** si la aplicación finaliza y el resultado coincide con el resultado de la ejecución sin fallo, o **pérdida de secuencia** cuando la secuencia de ejecución sufre un error y la aplicación no finaliza. La Tabla 11 contiene la clasificación de fallos obtenida en las campañas de inyección realizadas.

Circuito	Aplicación	#Fallos inyectados	Clasificación de fallos		
			Averías	Sin avería	Pérdida de secuencia
ARM	Producto de matrices de 9x9	4.000	2.396 (59,9%)	1.555 (38,9%)	49 (1,2%)
PowerPC	Web server	6.462	183 (2,8%)	5.392 (83,4%)	887 (13,7%)

Tabla 11. Clasificación de fallos obtenida en los experimentos realizados sobre los distintos microprocesadores.

La campaña de inyección realizada sobre el microprocesador ARM consiste en la inserción de 4.000 fallos generados aplicando una distribución de probabilidad uniforme. Los elementos de memoria que se consideran susceptibles de sufrir un SEU son todos los registros de uso general y especial, incluidos el contador de programa, el registro de estado y el puntero de pila. El código se almacena en memoria flash por lo que estas posiciones no se consideran sensibles a SEUs y sólo se insertan fallos en la memoria de datos y en especial en la parte de memoria utilizada por la aplicación, esto es, la parte utilizada para almacenar las matrices. Los posibles instantes de inyección se determinan estableciendo puntos de ruptura, por lo que se limitan a la primera etapa de la ejecución de una instrucción en ensamblador.

La mayoría de los fallos inyectados en el ARM, casi un 60%, se clasifican como avería, mientras que el porcentaje de fallos que producen una pérdida de secuencia es prácticamente despreciable. Esto se debe a la localización de los fallos insertados. Todos los elementos de memoria se consideran igualmente probables de sufrir un fallo por lo que la mayoría de los fallos insertados se corresponden con localizaciones de memoria, en detrimento de fallos insertados en registros. Una pérdida de secuencia se produce al inyectar un fallo en algún registro utilizado para generar la dirección de la siguiente instrucción a ejecutar, puesto que no inyectamos en las instrucciones, como por ejemplo el contador de programa. Como los fallos insertados en los registros representan un porcentaje pequeño con respecto a la lista completa de fallos (240 de los 4000 fallos insertados), el porcentaje de pérdidas de secuencia es prácticamente despreciable. Por otro lado, la aplicación ejecutada en el microprocesador está orientada a cálculo y la mayor parte de los fallos insertados se inyectan precisamente en los datos utilizados (3.760 de los 4.000 fallos insertados), lo que explica que la mayoría de los fallos provoquen una avería.

La campaña de inyección realizada sobre el microprocesador PowerPC consiste en la inserción de 6.462 *bit-flip*s. En este caso, se considera que el código y los datos se almacenan en la memoria RAM externa y que el resultado de la aplicación consiste en los datos de la página *web* que el sistema envía al PC. El resultado de la aplicación se almacena en la memoria de datos. La lista de fallos se genera considerando que

cualquier bit de los registros de uso general o especial y de la memoria de datos e instrucciones puede ser sensible a un SEU. Con respecto al instante de inyección, en este caso, el sistema de inyección y el microprocesador tienen la misma señal de reloj por lo que el instante de inyección puede seleccionarse entre todos los posibles ciclos de ejecución.

El diccionario de fallos obtenido para el PowerPC muestra que la mayor parte de los fallos no tienen un efecto visible a la salida del sistema. Cuando el fallo produce un comportamiento erróneo del sistema el efecto más probable consiste en que el sistema no consigue enviar al PC la página *web* y por lo tanto el fallo se clasifica como una pérdida de secuencia. Tan sólo un 2,8% de los fallos tienen como efecto el envío de la página con datos erróneos, lo que se clasifica como avería.

6.2.3.2 Velocidad del proceso de inyección de fallos

Los sistemas de inyección utilizados para realizar experimentos sobre ambos microprocesadores, el orientado al caso general y el sistema orientado a microprocesadores integrados en SoPCs, presentan diferencias que influyen en el tiempo necesario para realizar el proceso de inyección. Estas diferencias son las siguientes:

- Cómo se determina el instante de inyección. El sistema de inyección orientado a componentes microprocesadores comerciales, utilizado para realizar una campaña de inyección sobre el microprocesador ARM, utiliza puntos de ruptura para establecer el instante de inyección, mientras que en los experimentos realizados con el PowerPC empotrado en un SoPC, el instante de inyección se establece en función de ciclos de reloj gracias a que la señal de reloj es la misma tanto para el microprocesador como para el sistema de inyección.
- La aplicación *software* consistente en un producto de matrices de dimensión 9x9 y contiene instrucciones que se ejecutan numerosas veces dentro de bucles de ejecución. En el experimento realizado con el ARM, para inyectar un fallo cuando la aplicación ejecuta una instrucción dada *j* veces, es necesario establecer un punto de ruptura en dicha instrucción y alcanzarlo *j* veces, lo que supone detener la ejecución del microprocesador. En el experimento realizado con el SoPC, la ejecución del microprocesador se detiene únicamente una vez, independientemente del número de veces que se ejecute dicha instrucción en un bucle, puesto que el instante de inyección se fija por el número de ciclos de reloj ejecutados.
- La inserción del fallo. En el sistema de inyección implementado para evaluar un microprocesador empotrado en un SoPC, se accede a las capacidades de depuración a través de una herramienta *software*, mientras que en el sistema de

inyección implementado para el caso general, el control del OCD se realiza en *hardware*.

- La lectura de los resultados de la aplicación para la clasificación del fallo. El sistema de inyección orientado a evaluar un microprocesador empotrado en un SoPC, tiene acceso directo a las señales de control de la memoria dónde se almacenan los resultados, mientras que en el sistema de inyección orientado de forma general a microprocesadores implementados en *hardware* cada palabra de memoria debe leerse utilizando órdenes del depurador y la interfaz serie JTAG.

La Tabla 12 recoge el tiempo medio empleado en tareas de inyección en las distintas pruebas realizadas. En la campaña realizada con el ARM7TDMI-S la frecuencia de ejecución era de 60 MHz mientras que la frecuencia del reloj JTAG, señal TCK, era de 4MHz. En el caso del microprocesador PowerPC, la frecuencia de ejecución era de 100MHz y la frecuencia del reloj JTAG era de 5MHz. De los resultados obtenidos, se observa que a pesar de los numerosos bucles de ejecución que conforman un producto de matrices, el tiempo medio de inyección medido en ambos sistemas es del mismo orden. Si el mecanismo para fijar el instante de inyección en el sistema orientado a SoPCs es el uso de puntos de ruptura el tiempo medio aumenta hasta alcanzar los 10s/fallo. Esto indica que el factor más importante en la aceleración del proceso de inyección es la implementación *hardware* del control del OCD.

Microprocesador bajo evaluación	Aplicación *software*	Tiempo medio de inyección (ms/fallo)
ARM	Producto de Matrices 9x9	218
PowerPC	Producto de Matrices 9x9	260
	Servidor Web	2.000

Tabla 12. Tiempo medio empleado en las tareas de inyección para cada experimento realizado.

En caso de que cada instrucción se ejecute una única vez (sin bucles) el tiempo medio empleado en tareas de inyección al aplicar el sistema de evaluación general es de 2ms/fallo con TCK a 4MHz. Este tiempo incluye el tiempo empleado en realizar las instrucciones JTAG para parar la ejecución una vez, inyectar el fallo y recoger y analizar los resultados. La lectura de resultados es la tarea más lenta y el tiempo necesario para ello depende de la cantidad de datos a leer.

La campaña de inyección realizada cuando la aplicación consiste en un servidor *web* es la más lenta. La ejecución sin fallo requiere 12s, y el tiempo medio por fallo medido es de 14s. En este caso, el número de fallos que provoca una pérdida de secuencia supone el 13,7%, que es un porcentaje considerable. Un fallo se clasifica como una pérdida de secuencia, cuando después de transcurrido un tiempo *t* mayor que

la duración de la ejecución sin fallos, la aplicación no ha finalizado. Por lo tanto, para el servidor *web*, en casi el 14% de los fallos se emplea un tiempo adicional en clasificar los fallos lo que tiene como consecuencia que el tiempo medio por fallo observado sea considerablemente mayor que en el caso del producto de matrices.

6.2.3.3 Resultados en área

En este apartado se presentan los recursos necesarios para implementar el sistema de inyección propuesto para evaluar componentes microprocesadores comerciales, y el caso particular de microprocesadores integrados en SoPCs. El estudio de los recursos necesarios junto con las características proporcionadas por cada técnica, en cuestión de precisión y velocidad, permite evaluar el compromiso entre coste y prestaciones alcanzado en cada caso.

Los dos sistemas de inyección propuestos para microprocesadores se han implementado en distintas plataformas *hardware*. Como se indicó anteriormente, el sistema de inyección basado en el uso del OCD a través del control de su interfaz JTAG se implementa en una FPGA Spartan-3, un FPGA de bajo coste de Xilinx, mientras que el sistema de inyección orientado a evaluar microprocesadores en SoPCs se implementa en la lógica programable disponible en el propio SoPC bajo estudio. La Tabla 13 muestra los resultados de área en términos de número de biestables (#FFs) y número de LUTs (#LUTs) utilizados para la implementación de ambos sistemas de evaluación, así como el porcentaje que supone con respecto a los recursos disponibles en la FPGA. El sistema de inyección utilizado para insertar fallos en el microprocesador ARM es más complejo que para un SoPC porque incluye el control de la interfaz JTAG y la secuencia de instrucciones JTAG necesaria para implementar las acciones de depuración requeridas durante la inyección. En cualquier caso, dicho sistema se implementa en una FPGA de bajo coste, ocupando un 24% de LUTs y un 3% de biestables. Por otro lado, el sistema de inyección aplicado sobre el microprocesador PowerPC, empotrado en una placa Virtex-II Pro, es muy sencillo y los recursos necesarios para su implementación son mínimos, menos de un 10%.

Microprocesador bajo evaluación	Aplicación *software*	Placa *hardware*	Recursos utilizados		Recursos disponibles	
			#FF (%)	#LUTs (%)	#FFs	#LUTs
ARM	Producto de Matrices 9x9	Spartan-3 (SC3S1000)	567 (3,7%)	3.813 (24,8%)	15.360	15.360
PowerPC	Servidor Web	Virtex-II Pro (XC2VP4)	205 (3,4%)	574 (9,5%)	6.016	6.016

Tabla 13. Resultados de área para la implementación de los sistemas de inyección propuestos para microprocesadores y microprocesadores integrados en SoPC.

El tamaño del sistema de inyección no depende significativamente de la aplicación, en ninguno de los dos casos, sino tan sólo del procesador a evaluar.

6.3 Resumen y conclusiones

En este capítulo se han presentado los resultados experimentales obtenidos en la caracterización de los sistemas de inyección de fallos transitorios en circuitos digitales propuestos en este trabajo de tesis doctoral. Cada una de las técnicas de inyección propuestas se estudia en función de su eficiencia, velocidad y coste. Para evaluar dichos aspectos se presenta la clasificación de fallos obtenida, el tiempo medio empleado por fallo y los recursos necesarios para su implementación.

En primer lugar, se presentan los resultados de las pruebas realizadas relativas al sistema de Emulación Autónoma, las distintas implementaciones propuestas y la inyección de fallos con circuitos que contienen memorias empotradas. En segundo lugar, se prueba el sistema de inyección orientado a microprocesadores implementados en *hardware* propuesto en el capítulo 5, así como la solución propuesta para el caso particular en el que el microprocesador a evaluar está integrado en un SoPC.

Con respecto al sistema de Emulación Autónoma, se han realizado experimentos con distintos circuitos, tanto circuitos de prueba como circuitos reales. Cada experimento consiste en realizar una inyección exhaustiva de todos los posibles fallos simples. Los experimentos con los circuitos de prueba se han utilizado para realizar un análisis comparativo de las distintas implementaciones propuestas para la Emulación Autónoma, *Time-Multiplexed*, *State-Scan* y *Mask-Scan*. La clasificación de fallos obtenida con cada técnica es la misma, aunque la técnica *Mask-Scan* no permite distinguir entre fallos silenciosos y fallos latentes. Los experimentos realizados con circuitos que contienen memorias empotradas muestran las diferencias en la clasificación obtenida con los dos modelos de memorias propuestos para soportar la Emulación Autónoma. El modelo de memoria básico es más sencillo pero más impreciso y, por lo tanto, menos eficaz.

Los resultados obtenidos muestran que la técnica *Time-Multiplexed* proporciona una mayor velocidad del proceso de inyección gracias a la detección rápida de fallos silenciosos. A pesar, de que en la técnica *Time-Multiplexed* se ejecutan alternativamente el circuito con fallo y sin fallo, duplicándose los ciclos de ejecución, se obtienen tasas de inyección del orden de µs/fallo, permitiendo la inyecciones de **millones de fallos en un segundo**. El circuito de prueba b14 es utilizado para evaluar otra técnica de inyección basada en emulación de fallos con FPGAs en [Cive01a] obteniéndose una tasa de 100µs/fallo. Por lo tanto la Emulación Autónoma acelera el proceso de inyección con respecto a otras técnicas de emulación hasta dos órdenes de magnitud.

Para comparar las distintas técnicas propuestas para implementar un sistema de Emulación Autónoma se realizan campañas de inyección aplicando las tres técnicas sobre los mismos circuitos y se comparan tanto el tiempo medio de inyección como los recursos lógicos necesarios para la implementación de cada técnica. En el capítulo 4, se presentó una estimación teórica para el tiempo medio empleado por cada técnica en el proceso de inyección. Dicha estimación consideraba el peor caso en la clasificación de fallos y no incluía la mejora obtenida con la clasificación rápida de fallos silenciosos. Según la estimación teórica, la técnica de inyección que logra una mayor aceleración depende de la relación existente entre la longitud del banco de pruebas y el número de elementos de memoria del circuito. Con el objetivo de analizar la influencia de la longitud del banco de prueba en el proceso de inyección se han aplicado dos conjuntos de estímulos en cada circuito de prueba, con distintas longitudes. Los resultados experimentales coinciden con la estimación realizada cuando se comparan las técnicas *State-Scan* y *Mask-Scan*, de forma que la técnica *State-Scan* es más rápida que *Mask-Scan* cuando el número de ciclos de ejecución es mayor que dos veces el número de biestables del circuito, lo que es el caso general. En caso que el número de fallos silenciosos sea despreciable en relación con los fallos latentes del circuito, la técnica *Time-Multiplexed* no proporciona un beneficio significativo. Sin embargo, los circuitos tolerantes a fallos presentan un número considerable de fallos silenciosos gracias a los mecanismos de redundancia implementados. Se han realizado campañas de inyección sobre versiones endurecidas de algunos de los circuitos evaluados. La clasificación de fallos obtenidos muestra el aumento de fallos silenciosos y la reducción de averías, es decir, la mejora del nivel de confiabilidad del circuito endurecido. Por lo tanto, para la evaluación de circuitos tolerantes a fallos, la técnica más rápida será la técnica *Time-Multiplexed*.

Los recursos necesarios para implementar un sistema de Emulación Autónoma de acuerdo con cada una de las tres técnicas propuestas son inversamente proporcionales a las prestaciones ofrecidas. La técnica *Time-Multiplexed* es la más rápida y permite observar el contenido de cada biestable del circuito, comparándolo con el estado esperado sin fallo en cada ciclo de ejecución. Esta implementación es la más costosa, requiriendo hasta un 400% de incremento en el área utilizada con respecto a la implementación del circuito a evaluar. Si el circuito contiene memorias empotradas este incremento de área es el doble si se aplica el modelo ECAM con respecto al incremento involucrado si se utiliza el modelo de memoria básico, debido a la complejidad del modelo ECAM.

Los experimentos realizados con circuitos reales demuestran la capacidad del sistema de Emulación Autónoma para evaluar de forma rápida, eficiente y con un bajo coste, un considerable número de fallos en circuitos complejos. Dentro de los circuitos

reales evaluados se encuentra el microprocesador LEON2, del que se presenta un análisis de la clasificación de fallos en función de la zona del microprocesador en la que se inserta el *bit-flip*.

Por otra parte, se presentan resultados sobre el sistema de inyección propuesto para evaluar microprocesadores de los cuales no se dispone una descripción sino sólo de una implementación final del mismo. Se presentan dos grupos de resultados. Por un lado, se presentan los resultados correspondientes al sistema de inyección para microprocesadores *hardware* en general, basado en la inyección de fallos a través del interfaz JTAG del OCD integrado en el microprocesador a evaluar. Por otro lado, se presentan los resultados correspondientes al sistema de inyección propuesto para el caso particular en el que el microprocesador está integrado en un SoPC. En ambos casos los experimentos se han realizado sobre microprocesadores comerciales y complejos, como son el ARM y el PowerPC. Los resultados experimentales permiten probar la viabilidad de las técnicas propuestas así como la aceleración del proceso de inyección debida a la implementación *hardware* del control de la interfaz JTAG y de las órdenes necesarias para el manejo de la infraestructura de depuración. Sin embargo, hay que notar que las velocidades de inyección obtenidas con sendos sistemas de evaluación son menores que en el caso de la Emulación Autónoma puesto que la accesibilidad y controlabilidad de los recursos internos de los circuitos implementados en *hardware* son menores que en el caso de disponer de una descripción en un lenguaje de alto nivel. El tiempo medio por fallo empleado en el proceso de inyección depende de la cantidad de datos a leer como resultados de la aplicación. Los recursos necesarios para la implementación de un sistema de inyección para microprocesadores implican un pequeño porcentaje de los recursos disponibles en las FPGAs actuales, de forma que se puede implementar en una FPGA de bajo coste o en el SoPC bajo estudio, según corresponda.

Los sistemas de inyección propuestos no dependen de la tecnología utilizada por lo que se pueden implementar en cualquier FPGA con la suficiente cantidad de recursos. La portabilidad de una plataforma a otra es sencilla y esto permite aprovechar los avances en la tecnología con el desarrollo y comercialización de FPGAs cada vez más potentes y con mayor capacidad de integración. De hecho, durante el desarrollo de este trabajo de tesis doctoral y dentro de los experimentos realizados, se han utilizado distintas plataformas *hardware*.

En resumen, los resultados experimentales obtenidos permiten concluir que la implementación *hardware* del sistema de inyección de fallos transitorios acelera notablemente el proceso de evaluación de la tolerancia a fallos, ya que permite paralelizar distintas tareas de inyección, y proporciona un método rápido, eficiente y de bajo coste para la evaluación de la cobertura de fallos en circuitos digitales.

CAPÍTULO 7

CONCLUSIONES Y LÍNEAS FUTURAS

7. Conclusiones y Líneas Futuras

Las técnicas de inyección de fallos son el método más utilizado por la comunidad científica para evaluar el nivel de tolerancia a fallos de un CI. La complejidad de los circuitos actuales requiere el uso de técnicas de inyección rápidas que permitan la evaluación de un considerable número de fallos en un tiempo aceptable. Este trabajo de tesis doctoral presenta nuevas soluciones para la inyección de fallos SEUs en circuitos digitales, con el principal objetivo de acelerar el proceso de evaluación, alcanzando un compromiso con otros requisitos como la eficiencia, generalidad y bajo coste.

Se ha propuesto un entorno de inyección basado en emulación con FPGAs, en el que el sistema de inyección se implementa en *hardware*. De acuerdo con el entorno de inyección propuesto, se han descrito y desarrollado distintas técnicas para dar soluciones a las necesidades de los sistemas digitales actuales, que tienden a integrar distintos componentes en un mismo chip, SoCs. Por una parte, se presenta una técnica de inyección aplicable cuando se dispone de la descripción del circuito en un lenguaje de descripción *hardware*, como VHDL, y por otro lado, se ha desarrollado una técnica de inyección orientada a la evaluación de la tolerancia a fallos SEU en microprocesadores cuando se dispone de un componente ya fabricado.

A partir del estudio del estado de la técnica se puede concluir que, cuando se dispone de una descripción del circuito, la técnica de inyección más rápida es la emulación de fallos basada en FPGAs. Sin embargo, las soluciones basadas en emulación propuestas por otros autores, que proporcionan tasas de inyección de hasta 100μs/fallo, presentan una limitación en velocidad debido a la intensa comunicación requerida entre el PC y el *hardware*, puesto que el control de la campaña de inyección se realiza en *software* desde el PC. La primera aportación original es un sistema de **Emulación Autónoma que minimiza drásticamente la comunicación** entre el PC y la plataforma *hardware* de emulación [Lope07a][Lope07b][Lope05b][Lope05c]. La Emulación Autónoma permite explotar al máximo las ventajas de la emulación en términos de velocidad puesto que todas las tareas relacionadas con la inyección, como el control de la campaña, la observación del comportamiento del circuito y la clasificación de los fallos, o la aplicación de los estímulos, se realizan en *hardware* y es posible paralelizar tareas. Para ello, se explotan los recursos empotrados en las plataformas FPGA, como es el caso de los bloques de memoria RAM. Además, el sistema de Emulación Autónoma permite implementar una serie de **optimizaciones del proceso de inyección** que reducen el tiempo empleado en la evaluación de cada fallo, y por lo tanto disminuyen el tiempo necesario para realizar la campaña completa de fallos. Dichas optimizaciones pueden implementarse en un sistema de Emulación Autónoma

por la mayor observabilidad y controlabilidad que ofrece este entorno en comparación con las técnicas de inyección presentes en la literatura. Los resultados experimentales realizados muestran que con la Emulación Autónoma se pueden obtener tasas de inyección del orden de **1µs/fallo,** lo que supone una mejora de dos órdenes de magnitud con respecto al estado de la técnica. Estas tasas de inyección permiten realizar campañas de inyección con millones de fallos e incluso realizar campañas exhaustivas, lo que hasta ahora era un proceso muy lento o inviable.

Se han presentado y analizado tres posibles implementaciones de un sistema de Emulación Autónoma denominadas *Time-Multiplexed* [Lope05a], *State-Scan* [Port04] y *Mask-Scan* [Lope04][Garc04b]. Estas técnicas son aportaciones originales de esta tesis doctoral. Las técnicas *Time-Multiplexed* y *Mask-Scan* utilizan el mismo mecanismo de inyección, pero se diferencian en el mecanismo de observación utilizado para la clasificación de los fallos. Mientras que en la técnica *Time-Multiplexed* la ejecución con fallo y sin fallo se realizan alternativamente, comparándose en cada ciclo el contenido de los elementos de memoria, en la técnica *Mask-Scan* sólo se comparan las salidas del circuito. Por otro lado, la técnica *State-Scan* se basa en la inserción del estado del circuito correspondiente al instante de inyección mediante una cadena de *scan*. Las tres técnicas implementan distintos niveles de optimización, por lo que ofrecen diferentes compromisos en términos de prestaciones y recursos necesarios. Se ha realizado un análisis comparativo de las tres implementaciones propuestas que se corroboró y completó con los resultados experimentales. Según los datos obtenidos, la técnica *Time-Multiplexed* ofrece las mejores prestaciones aunque requiere de un considerable número de recursos. Sin embargo, teniendo en cuenta la evolución de los dispositivos y plataformas FPGAs, con gran cantidad de recursos disponibles, esto no supone un inconveniente importante. De hecho como parte de los experimentos y pruebas desarrolladas se ha realizado una campaña de inyección mediante dicha técnica en un procesador LEON2, que es un circuito complejo de gran interés en la industria espacial (donde se utiliza su versión endurecida). La solución propuesta es independiente de la tecnología puesto que no se basa en ningún mecanismo de configuración particular o característica ofrecida por algún fabricante, de forma que puede implementarse en cualquier dispositivo FPGA que contenga la cantidad de recursos lógicos necesarios.

Se han desarrollado, como aportación original, **modelos de memoria que soportan la Emulación Autónoma** para la inyección de fallos en circuitos con memorias empotradas [Garc06][Port06]. Generalmente, las soluciones basadas en emulación de fallos mediante la instrumentación del circuito, propuestas por otros autores, no se contemplan la emulación de circuitos con memorias empotradas. Las propuestas que sí presentan trabajos relacionados con dichos circuitos, generalmente orientadas a emular circuitos microprocesadores, proporcionan soluciones lentas en

comparación con la inyección en biestables o con una limitada capacidad de análisis de los resultados, debido a la limitada accesibilidad de las memorias (una palabra en cada ciclo de reloj). La creciente importancia de las memorias empotradas en los diseños actuales hace necesario utilizar bloques de memoria para la implementación del circuito en la FPGA durante la emulación de fallos, que soporten la inyección y la observación de los efectos de forma rápida.

Los bloques de memoria presentan fuertes limitaciones de observabilidad y controlabilidad por lo que requieren de especial atención para poder utilizarlas en la Emulación Autónoma y mantener los beneficios obtenidos con las distintas optimizaciones implementadas. La solución presentada se basa en detectar los fallos que se introducen en la memoria a partir de los valores de las señales de control y de datos, evitando tener que acceder a cada palabra para comprobar si hay o no un fallo en los datos. Con respecto a la inyección en memoria, se consideran como posibles fallos aquellos que se producen en cada ciclo de ejecución y que afectan al contenido de memoria que está siendo accedido en ese instante. De esta forma, se evita inyectar de manera innecesaria en palabras de memoria que no se utilizan y que pueden clasificarse como latentes sin necesidad de evaluarlos. Se han propuesto dos modelos de memoria, *Básico* y *ECAM*, con distinta complejidad y precisión, de acuerdo con la técnica de inyección *Time-Multiplexed* propuesta en esta tesis. Las pruebas realizadas con dichos modelos se han desarrollado sobre un circuito complejo, como es el microprocesador LEON2 que dispone de memorias empotradas para implementar, por ejemplo, la memoria caché o la pila. Los resultados obtenidos muestran cómo los modelos propuestos permiten mantener los beneficios en velocidad proporcionados por la Emulación Autónoma, permitiendo estudiar los efectos que los fallos inyectados en biestables provocan en la memoria así como los efectos que los fallos insertados en memoria producen en el comportamiento del circuito.

Además, se han desarrollado herramientas para automatizar la generación de un entorno de inyección basado en Emulación Autónoma, utilizando cualquiera de las tres técnicas propuestas. De esta forma, a tenor de los resultados obtenidos se puede concluir que el sistema de Emulación Autónoma es un método rápido y eficaz para la evaluación de circuitos grandes y complejos, aplicable a cualquier circuito cuya descripción esté disponible en VHDL o Verilog, de forma sencilla y semi-automática, permitiendo tasas de inyección del orden de millones de fallos en pocos segundos.

Por otra parte, se ha estudiado el problema de la inyección de fallos SEUs en circuitos microprocesadores por su importancia, en especial como componentes en SoCs y sistemas empotrados, cada vez más numerosos. El avance y evolución de las prestaciones de los circuitos microprocesadores permite el desarrollo de nuevas técnicas de inyección que explotan las nuevas capacidades. La solución propuesta en este trabajo

de tesis se aplica en componentes comerciales y hace uso de las **infraestructuras de depuración** integradas (OCD) en los microprocesadores actuales **mediante una interfaz JTAG** [Port07]. Cuando la inyección de fallos se realiza sobre una implementación *hardware* del circuito a evaluar, la controlabilidad y observabilidad de los recursos internos de dicho circuito son muy reducidas. El OCD de un microprocesador permite acceder a sus recursos internos. La novedad del método propuesto en esta tesis reside fundamentalmente en dos aspectos:

- El sistema de inyección **se implementa por completo en *hardware***, en una FPGA, lo que posibilita paralelizar tareas y en consecuencia acelerar el proceso con respecto a su implementación en *software* y evitar la comunicación con el PC que supone un cuello de botella en el proceso de evaluación.
- Es una solución con **amplia aplicabilidad** puesto que la interfaz JTAG es el mecanismo de acceso más general al OCD de los microprocesadores actúales. Además, la aproximación propuesta se basa en el uso de las capacidades básicas de depuración, disponibles en cualquier OCD, aunque pueden utilizarse capacidades avanzadas si están disponibles en el microprocesador bajo estudio para optimizar el proceso de inyección, como por ejemplo, la depuración en tiempo real. Este sistema de inyección es general, a expensas de las instrucciones JTAG necesarias para implementar cada acción de depuración, que dependen del microprocesador a evaluar.

Los resultados experimentales realizados sobre un microprocesador con arquitectura ARM demuestran la viabilidad de esta solución. El sistema se ha implementado en una FPGA de bajo coste y tamaño medio ocupando menos de la mitad de los recursos disponibles en términos de LUTs y biestables, por lo que se puede concluir que esta solución es efectiva en coste. Existen soluciones más rápidas en la literatura pero o son intrusivas, requiriendo la modificación del *software* o del *hardware*, o se basan en características avanzadas que no tienen todos los microprocesadores, por lo que no son suficientemente generales. Esta técnica es no intrusiva, puesto que no requiere la modificación de ningún componente del microprocesador, ni *hardware*, ni *software*. Por lo tanto, la técnica de inyección basada en el uso del OCD mediante una interfaz JTAG propuesta en esta tesis proporciona un compromiso entre velocidad, generalidad, sencillez, bajo coste y eficiencia.

Adicionalmente, se ha estudiado el caso en el que el microprocesador está integrado en un SoPC [Bern06b][Sonz06]. En ese caso el sistema propuesto para la inyección en microprocesadores implementados en *hardware* puede optimizarse utilizando la lógica programable para realizar ciertas tareas de inyección, como la clasificación de fallos o incluso el sistema completo de inyección. De esta forma se

aumenta la observabilidad y controlabilidad del microprocesador bajo estudio al tener acceso a todas las señales de su interfaz, como el bus de datos, y no únicamente a la interfaz JTAG. También mejora la velocidad de inyección al evitar los retardos de las señales y no ser necesario sincronizar el microprocesador con el sistema de inyección. Se han realizado experimentos con un SoPC complejo como es el circuito Virtex-II que contiene integrado un PowerPC. En dichas pruebas, el sistema de inyección no ha sido íntegramente implementado en *hardware*, si no que por simplicidad y puesto que otros experimentos muestran la validez y mejoras proporcionada por la implementación *hardware* (resultados con el microprocesador ARM7), se ha utilizado la herramienta *software* que proporciona el fabricante para controlar el OCD y así realizar la inserción de fallos. Los resultados experimentales ilustran el beneficio que se obtiene gracias a la mejora en la observabilidad del circuito. Además, al comparar estos resultados con los obtenidos en el caso general con el ARM7, se puede evaluar de forma cualitativa el aumento en la velocidad de inyección debida a la implementación *hardware* del sistema completo de inyección.

En resumen, se puede concluir que se ha logrado el objetivo de este trabajo de tesis, consistente en el desarrollo de nuevas técnicas de inyección que permitan la evaluación de la tolerancia a fallos SEU en circuitos complejos. Para ello es fundamental acelerar el proceso de inyección a la vez que se proporciona eficiencia, generalidad y bajo coste. Según los resultados obtenidos, la implementación *hardware* del sistema de inyección acelera significativamente el proceso de evaluación, siendo eficiente en coste gracias al avance de los dispositivos FPGA y su extendida comercialización.

El sistema de Emulación Autónoma permite realizar campañas de fallos exhaustivas o con un alto porcentaje de los posibles fallos, A partir del trabajo desarrollado en esta tesis doctoral se pueden evaluar circuitos reales concretos, de gran interés en aplicaciones críticas, para comprobar su comportamiento frente a fallos. Se propone evaluar incluso circuitos que ya fueron analizados mediante la inyección de un pequeño porcentaje de los posibles fallos para comparar los resultados obtenidos. De esta forma se puede valorar experimentalmente la importancia de realizar campañas de fallos exhaustivas. Para ello, sería interesante desarrollar una interfaz gráfica que facilite el manejo de las herramientas realizadas para la automatización de la generación del sistema de Emulación Autónoma, así como para el tratamiento de los resultados. Con respecto al análisis de resultados, se pueden añadir nuevas capacidades, como generaciones automáticas de bases de datos que muestren la clasificación de fallos en función de la zona del circuito en la que se inyectó el fallo, o realizar el análisis de otros resultados, como tiempos de latencias de fallos.

Por otro lado, se puede aplicar el sistema de Emulación Autónoma a la evaluación de la tolerancia a los fallos originados por otros fenómenos que afecten a elementos de memoria, como los producidos por interferencias electromagnéticas (EMI). Para ello, es necesario disponer de los modelos de fallos correspondientes, es decir, cómo se producen estos fallos y qué elementos de memoria se ven afectados. A partir, de los modelos de fallo, la inserción de fallos en elementos de memoria es inmediata con el sistema de Emulación Autónoma que se ha propuesto.

Otra línea futura de trabajo es la extensión del sistema de Emulación Autónoma a fallos transitorios en la lógica combinacional, tipo SET. Debido al incremento de la frecuencia de reloj, con el avance de la tecnología, está aumentando la probabilidad de que un SET se transmita hasta ser almacenado en un elemento de memoria. Por lo tanto, los SETs se están convirtiendo en efectos de gran interés que deben ser estudiados. El concepto de Emulación Autónoma puede se aplicado en este caso para evaluar los fallos transitorios en lógica combinacional de los circuitos actuales.

CAPÍTULO 8

CONCLUSIONS AND FUTURE WORK

8. Conclusions and Future Work

Fault injection techniques are the most accepted method by the scientific community to evaluate fault tolerance level in an integrated circuit (IC). Current circuits' complexity requires faster fault injection techniques to allow the evaluation of a high number of faults in a reasonable time. This PhD work presents new solutions for SEU injection in digital circuits, with the main objective of speeding-up the evaluation process, reaching a trade-off between efficiency, wide applicability and low cost.

A fault injection environment based on the emulation with FPGA has been proposed. In this environment, the injection system is implemented in hardware. According to the proposed environment, different fault injection techniques have been presented and developed. These techniques solve the current digital systems evaluation requirements, with different components embedded in the same chip, SoCs. First, a fault injection technique, applicable when the circuit description in a hardware description language (as VHDL) is available, is proposed. A second technique injects SEU faults in microprocessors when the circuit description is not available, or the fault injection in an IP (intellectual property) is necessary, so the injection is performed on the manufactured circuit.

The study of state of the art fault tolerance evaluation methods has showed that the different fault injection techniques could be classified into two categories. One category includes techniques that are applied on a circuit description and the other one includes techniques applicable on a final hardware implementation or commercial component. In this work two techniques have been presented. The first one is aimed at injecting faults in a circuit when its description in a high level language is available. It is based on fault emulation with FPGAs. One of the conclusions stated after studying the state of the art is that the fastest fault injection technique, when the circuit description is available, is FPGA-based fault emulation. However, emulation-based solutions proposed by other researchers, that provide injection rates around 100µs/fallo, have a speed bottleneck due to the necessary communication between the PC and the hardware platform. In those solutions, the PC is in charge of controlling the whole fault injection campaign. The first original contribution of this thesis work is the **Autonomous Emulation system based on minimizing as much as possible the required communication** between PC and FPGA [Lope07a][Lope07b][Lope05b][Lope05c]. Autonomous Emulation profits from hardware speed since all fault injection tasks, like controlling the injection campaign, observing the circuit behaviour to classify the faults or the stimuli application, are performed in hardware. One of the advantages of the fault injection system hardware implementation is that some tasks could be paralleled, specially, in case of calculation-based tasks. The proposed injection technique exploits the embedded hardware resources available in the FPGA platforms, like RAM memory

blocks in order to implement the complete fault injection environment. Furthermore, Autonomous Emulation provides more observability and controllability than other proposals. These features are exploited to optimise the injection process. Such optimisations reduce the necessary time to evaluate every fault, and then they speed-up the fault injection campaign execution. Experimental results show that Autonomous Emulation provides injection rates around **1µs/fallo,** that is, a speed-up of two orders of magnitude with respect to the state of the art. These rates allow the injection of millions of faults, and even, performing exhaustive fault injection campaigns. The injection of so large sets of faults is infeasible or a very slow process with other techniques.

Three possible injection techniques according to the Autonomous Emulation architecture have been developed. Such different implementation are named Time-Multiplexed [Lope05a], State-Scan [Port04] y Mask-Scan [Lope04][Garc04b]. These techniques are original contributions of this PhD thesis. Time-Multiplexed and Mask-Scan techniques use the same fault injection mechanism, but they differ in the way to observe the circuit behaviour and to classify the faults. In Time-Multiplexed technique, the executions with faults (faulty) and without faults (golden execution) are performed alternately, comparing on-line all the memory elements content. However, in Mask-Scan technique only the circuit outputs are compared on-line. Regarding State-Scan, this technique is based in the circuit injection state insertion by means of a scan chain. The three techniques involve different optimisations number, so they provide different trade-off between performance and required resources. A comparative analysis of the three implementations has been performed and then the experimental results have proved it. According to the obtained results, Time-Multiplexed technique provide the best performance, although it implies more area overhead than the other ones. Anyway, taking into account the FPGA platforms development, with a lot of available resources, the area overhead is not an important limitation. Indeed experiments with a LEON2 processor, that is a complex circuit of high interest in the industry (specially, in the space industry), has been performed, injecting millions of faults.

Autonomous Emulation is independent on the technology since it is not based on any particular vendor mechanism, and therefore it could be implemented in any FPGA device with enough logic resources.

Other original contribution consists **in memory models that support Autonomous Emulation** to inject fault in circuits with embedded memories [Garc06][Port06]. In general, fault emulation-based solutions that use circuit instrumentation to inject faults in the literature does not consider the fault emulation in circuits with embedded memories. The few existing proposals that study this kind of circuits are oriented to inject faults in microprocessors, are slow solutions with respect to the injection in flip-flops and with a poor capacity to analyse the circuit behaviour,

due to the limited accessibility in memories (a word memory per clock cycle). Embedded memories are more and more usual and large in modern designs, and therefore, the emulation of the embedded memories is a problem of rising importance. The proposed models presented in this thesis allow the fault emulation in embedded memories, injection faults and observing their effects in a fast way.

Applying Autonomous Emulation to a circuit with embedded memories, maintaining the high performance and injection process speed-up, is a difficult problem to solve, since memory blocks present important limitations in observability and controllability. The proposed solution is based on observing control signals, data and address memory bus to detect errors in memory. Therefore, accessing every memory word in order to check data is avoided. Regarding fault injection in memory, faults are injected in every clock cycle only in data bus, that is, the fault is injected in the word memory that is accessed in the fault injection instant. This solution avoids fault injection in unused memory words, which can be classified as latent faults without performing their emulation.

Two memory models have been proposed according to the fault injection technique Time-Multiplexed. Both models, named basic and ECAM model, present different features in terms of complexity and accuracy. Experiments with a LEON2 microprocessor, that includes embedded memories, have been developed to prove the feasibility and features of the proposed memory models. Results show the proposed models allows the fault injection environment to provide the Autonomous Emulation advantages in terms of process speed, studying how faults injected in flip-flops provoke errors in memories as well as how the faults injected in memories affect the circuit behaviour.

Besides, Autonomous Emulation system generation and application have been automated using any of the proposed techniques. Some tools have been developed to support the automation. As conclusion, and taking into account obtained results, the propose Autonomous Emulation system is a fast an efficient method to evaluate large and complex digital circuits in a simple and semi-automatic way, when the circuit description in VHDL or Verilog is available, providing fault injection rates around the order of millions of faults in a few seconds.

On the other hand, fault injection, according to SEU faults, in microprocessors has been studied. Microprocessors are essential components in modern circuits, in special, in SoCs and embedded systems that are more and more usual designs. The progress and enhancement of the current microprocessors capabilities allow the development of new fault injection techniques applicable to this kind of circuits based on such new capabilities. The proposed solution in this work is devised to evaluate

commercial components profiting from the **enhance debug capabilities integrated in the microprocessor under test (OCD) by means of a JTAG interface** [Port07]. When the fault injection is performed in a final hardware implementation of the circuit under test, the controllability and observability of the internal memory elements is very limited. The OCD available in a microprocessor allows the access to the internal resources to read and write data. The novelty of this approach is twofold:

- The injection system is **implemented completely in hardware,** in a FPGA, what allows some tasks paralleling execution. Therefore, hardware implementation speed-up the injection process with respect the software implementation and it avoids the bottleneck communication between a PC and the circuit.
- The proposed technique provides a wide applicability since the JTAG interface is the most general and common mechanism to access OCD in the current microprocessors. Besides, this proposal is based on using basic debugging capabilities, available in any OCD. In any case, if more enhance debugging capabilities are integrated in the microprocessor under test, they could be used to improve the injection process, like for example, real-time debugging. The developed injection system is general for any microprocessor, except for JTAG instruction set required for implementing each debugging action that depends on the microprocessor under study.

Experiments have been performed on an ARM-based microprocessor. The obtained results prove the feasibility of this solution to evaluate SEU tolerance in real processors. The system has been implemented in a low cost and medium size FPGA using less than half of all the available resources in terms of LUTs and flip-flops, so as conclusion, this solution is effective in cost. Other solutions in literature are faster but they are intrusive because they require the software or hardware modification, or they are based on enhanced debugging infrastructures that are not available in any microprocessor. Therefore, other techniques could not be applicable to a wide range of circuits. The technique proposed in this thesis, based on using an OCD with a JTAG interface, provides a trade-off between injection speed, wide applicability, simplicity, low cost and efficiency.

Moreover, fault injection in SoPC's embedded microprocessors has been studied [Bern06b][Sonz06]. In this particular case, the proposed solution to perform de fault injection in COTS (commercial of-the-shelf) microprocessors could be improved exploiting programmable logic in order to perform some injection tasks, like fault classification or even, the complete injection system. This optimisation increases the observability and controllability of the internal microprocessor elements, allowing the

access not only to the JTAG interface but also to the other microprocessor signals. Besides, this approach improves the injection speed since it avoids delays in the signals interface and it does not require synchronizing the microprocessor clock with the fault injection system one. Experiments with a complex real SoPC, like a Virtex-II from Xilinx® with a PowerPC processor, have been developed. In these experiments, a software tool from the vendor has been used to manage the OCD in order to inject faults and observe their effects. The OCD control has been implemented in software, in this case, since experiments with the ARM-based microprocessor show the features of implementing all the injection tasks in hardware. Experimental results prove the enhancement provided by the proposed optimisation. Besides, measuring the speed up in a completely hardware implemented injection system has been qualitatively made by comparing the results obtained in ARM-based processor with the PowerPC ones.

To summarize, the developed new fault injection techniques allow the SEU faults tolerance evaluation of complex circuits, what was the main objective of this PhD work. They speed-up the injection process, providing efficiency, wide applicability and low cost, at the same time. Taking into account obtained results, the injection system hardware implementation speed-up significantly the evaluation process, being cost efficient thanks to the FPGA devices development and the extended market.

Autonomous Emulation system performs exhaustive fault injection campaigns or with large set of faults. This PhD work can be useful to evaluate the fault tolerance of given real circuits, with high interest in safety-critical applications, in order to check their behaviour with SEU faults. Also this work could be applied in evaluating circuits that have already been evaluated with only a small percentage of the possible faults. The results of the injection campaign with a small set of faults could be compare with results of an exhaustive injection in order to measure experimentally the needed of injecting a very large set of faults.

To develop a graphical user interface would be very interesting to make easier the application of the tools in charge of generating the Autonomous Emulation system and the result analysis. Regarding analysis results, new capabilities could be added, like automatic generation of a database to study the fault classification according to the different elements affected by a fault in the circuit under test, or like considering other data analysis as latencies.

Furthermore, Autonomous Emulation could be applied to evaluate tolerance to faults provoked by other phenomenon in memory elements, like for example the electromagnetic interferences (EMI). In order to study other effects fault models are required representing how the fault affect the circuit, which memory elements are

modified and when the faults occur. Applying the suitable fault model, fault insertion in memory elements is simple using Autonomous Emulation.

Another future work is the application of the Autonomous Emulation architecture to faults in combinational logic, SET. With modern technologies the clock frequency is increasing and SETs are effects more and more probable in current circuits. A SET in combinational logic could be propagated through the circuit until be stored in a memory element, what is more probable with high clock frequencies. Therefore, SETs are becoming in very important effects what need to be studied. The Autonomous Emulation architecture could be applied, in that case, in order to evaluate SETs in combinational logic of current circuits.

BIBLIOGRAFÍA

[Agui04] M. Aguirre, J. N. Tombs, F. Muñoz, V. Baena, A. Torralba, A. Fernandez-Leon, F. Tortosa, D. Gonzalez-Gutierrez "An FPGA based hardware emulator for the insertion and analysis of Single Event Upsets in VLSI Designs" Radiation Effects on Components and Systems Workshop, Septiembre 2004.

[Alde03] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, A. Marmo, S. Pastore, G. R. Sechi "A Tool for Injectiong SEU-like Faults into the Configuration Control Mechanism of Xilinx Virtex FPGAs" 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003

[Altera] www.altera.com

[Anto02] L. Antoni, R. Leveugle, B. Feher. "Using Run-Time Reconfiguration for Fault Injection in HW Prototypes". IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 245-253, 2002.

[Arla90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. Fabre, J. Laprie, E. Martins, D. Powel "Fault Injection for Dependability Validation: A Methodology and Some Applications" IEEE Transactions on Software Engineering, Vol. 16, No. 2, pp. 166-181, Febrero 1990.

[ARM7] "ARM7TDMI-S. Technical Reference Manual" Rev 4. ARM, 2001

[ATMEL] www.atmel.com

[Autr06] J. Autran, P. Roche, J. Borel, C. Sudre, K. Castellani-Coulié, D. Munteanu, T. Parrassin, G. Gasiot, J. Schoellkopf "Altitude SEE Test European Platform (ASTEP): Project Overview, First Results in CMOS 130nm and Perspectives" Radiation Effects on Components and Systems, 2006 (RADECS'06).

[Aviz04] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, pp. 11-33, Enero-Marzo 2004.

[Bakh05] A. Bakhoda, S. G. Miremadi, H. R. Zarandi "Investigation of Transient Effects on FPGA-based Embedded Systems" 2nd IEEE Interanional Conference on Embedded Software and Systems, 2005.

[Bara05] J. C. Baraza, J. Gracia, D. Gil, P. J. Gil "Improvement of Fault Injection Techniques Based on VHDL Code Modification" 10th IEEE International High-Level Design Validation and Test Workshop, pp. 19-26, Diciembre 2005.

[Baum05] R. C. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies", IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 305-316, Septiembre 2005.

[Bell01] M. Bellato, M. Ceschia, M. Menichelli, A. Papi, J, Wyss, A. Paccagnella "Ion Beam Testing of SRAM-based FPGA's" 6th European on Radiation and Its Effects on Components and Systems, pp. 474-480, septiembre, 2001.

[Bens03] A. Benso, P. Prinetto (Eds.) "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation", Holanda, Kluwer Academic Publishers, 2003.

[Bens98] A. Benso, P. Prinetto, M. Rebaudengo, M. Sonza Reorda "A Fault Injection Environment for Microprocessor-based Boards" International Test Conference, pp. 18-23, Octubre 1998.

[Bens99] A. Benso, M. Rebaudengo, M. Sonza Reorda "Fault Injection for Embedded Microprocessor.based Systems" Journal of Universal Computer Science (Special Issue on Dependability Evaluation and Validation), Vol. 5, No. 5, pp. 693-711, 1999.

[Bern04] P. Bernardi, M. Sonza Reorda, L. Sterpone, M. Violante "On the evaluation of SEU sensitiveness in SRAM-based FPGAs" 10th IEEE International On-Line Testing Symposium, Julio 2004.

[Bern06a] P. Bernardi, L. M. Bolzani, M. Rebaudengo, M. Sonza Reorda, F. Vargas, M. Violante "A New Hybrid Fault Detection Technique for Systems-on-a-Chip" IEEE Transactions on Computers, Vol. 55, No.2, pp. 185-198, Febrero 2006.

[Bern06b] P. Bernardi, L. Sterpone, M. Violante, M. Portela-Garcia, "Hybrid Fault Detection Technique: A Case Study on Virtex-II Pro's PowerPC 405," IEEE Transactions on Nuclear Science, Vol. 53, No. 6, pp. 3550 – 3557, Diciembre 2006.

[Berr02a] L. Berrojo, F. Corno, L. Entrena, I. Gonzalez, C. Lopez, M. Sonza Reorda, G. Squillero. "New Techniques for Speeding-up Fault-injection Campaigns", Design, Automation and Test in Europe Conference, 2002.

[Berr02b] L. Berrojo, F. Corno, L. Entrena, I. Gonzalez, C. Lopez, M. Sonza Reorda, G. Squillero. "An Industrial Environment for High-Level Fault-Tolerant Structures Insertion and Validation", Proceedings VLSI Test Symposium, pp. 229-236, 2002.

[Boue98] J. Boue, P.Petillon, Y. Crouzet. "MEFISTO-L: A VHDL-Based Fault Injection Tool for the Experimental Assessment of Fault Tolerance", 28th International Symposium Fault Tolerant Computing, 1998.

[Cali96] T. Calin, M. Nicolaidis, R. Velazco "Upset Hardened Memory Design for Submicron CMOS Technology" IEEE Transactions on Nuclear Science, Vol. 43, No. 6, diciembre 1996.

[Card02] G. C. Cardarilli, F. Haddour, A. Leandri, M. Ottavi, S. Pontarelli, R. Velazco. "Bit flip injection in processor-based architectures: a case study", Proccedings International On-Line Testing Workshop, 2002.

[Carr98] J. Carreira, H. Madeira, J. G. Silva. "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", IEEE Transactions on Software Engineering, Vol 24, No. 2, pp. 125-136, Febrero 1998.

[Celoxica] Celoxica RC1000 Hardware Reference Manual, Version 2.3, 2004

[Chen95] K. T. Cheng, S. Y. Huang, W. J. Dai. “Fault Emulation: A New Approach to Fault Grading” Proc. International Conference on Computer-Aided Design, pp. 681-686, 1995.

[Chey00] P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M. Sonza Reorda, M. Violante, “Experimentally Evaluating an Automatic Approach for Generating Safety-Critical Software with Respect to Transient Errors”, IEEE Transactions on Nuclear Science, Vol. 47, No. 6, pp. 2231-2236, diciembre 2000.

[Choi92] G. S. Choi, R. K. Iyer “FOCUS: An Experimental Environment for Fault Sensitivity Analysis” IEEE Transactions on Computers, Vol. 41, No. 12, pp. 1515-1526, Diciembre 1992.

[Cive01a] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante. “Exploiting Circuit Emulation for Fast Hardness Evaluation” IEEE Transactions on Nuclear Science, Vol 48, nº 6, 2001

[Cive01b] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante “FPGA-based Fault Injection for Microprocessor Systems”, IEEE Asian Test Symposium, pp. 304-309, 2001.

[Corn00] F. Corno, M. Sonza Reorda, G. Squillero. “RT-Level ITC’99 benchmarks and first ATPG results”. IEEE T. on Design and Test of Computers, pp 44-53, 2000.

[Delo96] T. A. Delong, B. W. Johnson, J. A. Profeta III “A Fault Injection Technique of VHDL Behavioral-Level Models” IEEE Design and Test Computers, Vol. 13, No. 4, pp. 24-33, 1996.

[EDN05] M. Santarini “CosmicRadiation Comes to ASIC and SOC Design” EDN Europe, May 2005 (www.edn.com).

[EETimes] David Lammers, Ron Wilson, “Soft errors become hard truth for logic”, EE Times, 3 de Marzo de 2004.

[Ejla05] A. Ejlali, B. M, Al-Hashimi. S, Ghassem Miremadi. “Fast observation architecture for FPGA-based SEU analysis" 10th European Test Symposium (ETS’05), Talinn, Estonia, Mayo 2005.

[Entr01] L. Entrena, C. López, E. Olías "Automatic Insertion of Fault-Tolerant Structures at the RT Level" 7th IEEE International On-Line Testing Wrokshop, pp. 48-50, 2001.

[Facc07] F. Faccio, "Design Hardening Methodologies for ASICs". En R. Velazco, P. Fouillat, R. Reis (Eds.) "Radiation Effects on Embedded Systems", Holanda, Springer, 2007, pp. 143-181.

[Faur02] F.Faure, P. Peronnard, R. Velazco and R. Ecoffet, "THESIC+,a flexible system for SEE testing", Proceedings of RADECS Workshop, pp. 231-234, Septiembre 2002.

[Faur03] F. Faure, R.Velazco, M. Violante, M. Rebaudengo, M. Sonza Reorda "Impact of Data Cache Memory on the Single Event Upset-Induced Error Rate of Microprocessors" IEEE Transactions on Nuclear Science, Vol. 50, N 6, Diciembre, 2003.

[Ferr01] P. A. Ferreira, C. A. Marqués, R. Velazco, O. Calvo "Injecting Single Event Upsets in a Digital Signal Processor by jeans of Direct Memory Access Requests : A New Method for Generating Bit Flips" 6th European Conference on Radiation and Its Effects on Components and Systems (RADECS), pp. 248-252, 2001.

[Fida06] A. V. Fidalgo, G. R. Alves, J. M. Ferreira "Real Time Fault Injection Using Enhanced OCD – A Performance Analysis" 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT), 2006.

[Folk98] P. Folkesson, S. Svensson, J. Karlsson "A Comparison of Simulation Based and Scan Chain Implemented Fault Injection" 28th Annual International Symposium Fault-Tolerant Computing, pp.248-293, Junio 1998.

[Full99] E. Fuller, M. Caffrey, P. Blain, C. Carmichel "Radiation Test Results of the Virtex FPGA and ZBT SRAM for Space Based Reconfigurable Computing" Military and Aerospace Programmable Logic Device (MAPLD) International Conference, 1999.

[Gaisler] http://www.gaisler.com/products/leon2/leon.html

[Garc04a] M. García Valderas, "Emulación en Prototipos Basados en FPGAs: Métodos de Depuración Mediante Inserción de Lógica Compatible con el Estándar IEEE-1149.1", Tesis doctoral, dirigida por Eduardo de la Torre Arnanz. Universidad Politécnica de Madrid, 2004.

[Garc04b] M. García Valderas, C. López Ongil, M. Portela García, L. Entrena, "Transient Fault Emulation of Hardened Circuits in FPGA Platforms", 10th International On-Line Testing Symposium (IOLTS'04), Julio 2004.

[Garc06] M. García-Valderas, M. Portela-Garcia, C. Lopez-Ongil, L. Entrena, "An Extension of Transient Fault Emulation Techniques to Circuits with Embedded Memories", Poster, Design & Diagnostics of Electronic Circuits & Systems (DDECS'06), Abril 2006.

[Gosw97] K. K. Goswami, R. K. Iyer, L. Young "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis" IEEE Transactions on Computers, Vol. 46, No. 1, pp. 60-74, Enero 1997.

[Hong96] J. H. Hong, S. A. Hwang, C. W. Wu. "An FPGA-Based *Hardware* Emulator For Fast Fault Emulation" MidWest Symposium on Circuits and Systems, Ames-Iowa (EEUU), 1996.

[Hsue97] M. C. Hsueh, T. K. Tsai, R. K. Iyer, "Fault Injection Techniques and Tools" IEEE Computer, Vol. 30, No. 4, pp. 75-82, Abril 1997.

[iRoC] www.iroctech.com

[Irom04] F. Irom, F. F. Farmanesh "Frequency Dependence of Single-Event Upset in Advanced Commercial PowerPC Microprocessors" IEEE Transactions on Nuclear Science, Vol. 51, No. 6, pp. 3505-3509, Diciembre 2004.

[ITRS] International Technology Roadmap for Semiconductors (ITRS). Ediciones 2001-2005.

[JEDEC] www.jedec.org

[Jenn94] E. Jenn, J. Arlat, M. Rimén, J, Ohlsson, J. Karlsson "Fault Injection into VHDL Models: The MEFISTO Tool" 24th International Symposium on Fault Tolerant Computing, pp. 66-75, USA, 1994.

[John89] B. Johnson “Design and analysis of Fault-Tolerant Digital Systems” Addison-Wesley, 1989.

[JTAG] “IEEE Standard Test Access Port and Boundary-Scan Architecture” IEEE Std 1141.1-2001.

[Kana95] G. Kanawati, N. A. Kanawati, J. A. Abraham “FERRARI: A Flexible Software-Based Fault and Error Injection System” IEEE Transactions on Computers, Vol. 44, No. 2, pp. 248-260, Febrero 1995.

[Karl94] J. Karlsson, P. Lidén, P. Dahigren “Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms” IEEE Micro, Vol. 14, Issue 1, pp. 8-23, Febrero 1994.

[Kent06] P. Kenterlis, N. Kranitis, A. Paschalis, D. Gizopoulus, M. Psarakis “A Low-Cost SEU Fault Emulation Platform for SRAM-Based FPGAs” 12th IEEE International On-Line Testing Symposium, pp.235-241, Julio 2006.

[Lamb05] D. Lambert, J. Baggio, G. Hurbert, V. Ferlet-Cavrois, O. Flament, F. Saigné, F. Wrobel, H. Duarte, J. Boch, B. Sagnes, N. Buard, T. Carrière “Neutron-Induced SEU in SRAMs: Simulatinos With n-Si and n-O Interactions” IEEE Transactions on Nuclear Science, Vol. 52, No. 6, pp. 2332-2339, Diciembre 2005.

[Lese05] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael, P. Alfke “The Rosetta Experiment: Atmospheric Soft Error Rate Testing in Differing Technology FPGAs” IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 317-328, Septiembre 2005.

[Leve00] R. Leveugle. “Fault Injection in VHDL Descriptions and Emulation” IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, 2000.

[Lima01a] F. Lima, S. Rezgui, L. Carro, R. Velazco, R. Reis, “On the use of VHDL simulation and emulation to derive error rates”, Proc. of 6th Conference on Radiation and its Effects on Components and Systems (RADECS' 01), Grenoble, Septiembre 2001.

[Lima01b] F. Lima, C. Carmichael, J. Fabula, R. Padovani, R. Reis "A fault injection analysis of Virtex® FPGA TMR design methodology" European Conference on Radiation and its Effects on Components and Systems (RADECS), pp. 275-282, 2001.

[Lima06] F. Lima Kastensmidt, L. Carro, R. Reis "Fault-Tolerance Techniques for SRAM-based FPGAs" Springer, 2006.

[Lope04] C. López Ongil, M. García Valderas, M. Portela García, L. Entrena, "Técnicas para Emulación de Fallos Transitorios en Plataformas FPGAs", Jornadas de Computación Reconfigurable (JCRA'04), Septiembre 2004.

[Lope05a] C. López Ongil, M. García Valderas, M. Portela García, L. Entrena, "Techniques for Fast Fault Grading Based on Autonomous Emulation", Conference on Design Automation and Test in Europe (DATE'05), Marzo 2005.

[Lope05b] C. López Ongil, M. García Valderas, M. Portela García, L. Entrena, "Autonomous transient fault emulation on FPGAs for accelerating fault grading", IEEE 11th International On-Line Testing Symposium (IOLTS'05), Julio 2005.

[Lope05c] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, L. Entrena-Arrontes, "An Autonomous FPGA-based Emulation System for Fast Fault Tolerance Evaluation", IEEE 15th International Conference on Field Programmable Logic and Applications (FPL'05), agosto 2005.

[Lope07a] C. López-Ongil, M. García-Valderas, M. Portela-García, L. Entrena, "Autonomous Fault Emulation: A New FPGA-based Acceleration System for Hardness Evaluation", IEEE Transactions on Nuclear Science, Vol. 54, Issue 1, Part 2, pp. 252-261, febrero 2007.

[Lope07b] C. López-Ongil, L. Entrena, M. García-Valderas, M. Portela, M. A. Aguirre, J. Tombs, V. Baena, F. Muñoz, "A Unified Environment for Fault Injection at Any Design Level Based on Emulation", IEEE Transactions on Nuclear Science, Vol. 54, No. 4, pp. 946-950, Agosto 2007.

[LPC2129] "LPC2119/2129/2194/2292/2294 User Manual" Philips Semiconductors, 2004.

[Mill04] F. Miller, N. Buard, T. Carrière, R. Dufayel, R. Gaillard, P. Poirot, J. M. Palau, B. Sagnes, P. Fouillat "Effects of Beam Spot Size on the Correlation Between Laser and Heavy Ion SEU Testing" IEEE Transactions on Nuclear Science, Vpl. 15, No. 6, pp. 3708-3715, diciembre 2004.

[Mitr05] S. Mitra, N. Seifert, M. Zhang, Q. Shi, K. S. Kim, "Robust System Design with Built-In Soft Error Resilience", IEEE Computer, pp. 43-52, Febrero 2005.

[Nexus] IEEE-ISTO 5001-2003, "The Nexus Forum™ standard for a global embedded processor debug interface", version 2.0, 2003.

[Nico05] M. Nicolaidis "Design for Soft Error Mitigation" IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, septiembre 2005.

[Nico99] M. Nicolaidis "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies" IEEE 17th VLSI Test Symposium, pp. 86-94, abril 1999.

[Oh02] N. Oh, P. P. Shirvani, E. J. McCluskey, "Control-Flow Checking by Software Signatures", IEEE Transactions on Reliability, Vol. 51, No. 2, pp. 111-122, Marzo 2002.

[OpenCore] www. opencores.org

[Parr00] B. Parrotta, M. Rebaudengo, M. Sonza Reorda, M. Violante. "New Techniques for Accelerating Fault Injection in VHDL descriptions ", IEEE International On-Line Test Workshop, pp. 61-66, 2000.

[Peng06] J. Peng, J. Ma, B. Hong, C. Yuan "Validation of Fault Tolerance Mechanisms of an Onboard System" 1st International Symposium on Systems and Control in Aerospace and Astronautics (ISSCAA), pp.1230-1234, Enero 2006.

[Port04] M. Portela García, C. López Ongil, M. García Valderas, L. Entrena, "Analysis of Transient Fault Emulation Techniques in Platform FPGAs", Conference on Design of Integrated Circuits and Systems (DCIS'04), Noviembre 2004.

[Port06] M. Portela-García, M. García-Valderas, C. López-Ongil, L. Entrena, "An Efficient Solution to Evaluate SEU Sensitivity in Digital Circuits with Embedded RAMs", XXI Conference on Design of Circuits and Integrated Systems (DCIS'06), noviembre 2006

[Port07] M. Portela-García, C. López-Ongil, M. García-Valderas, L. Entrena "A Rapid Fault Injection Approach for Measuring SEU Sensitivity in Complex Processors", IEEE International On-Line Testing Symposium (IOLTS'07), Julio 2007.

[Poug04] V. Pouget, D. Lewis, P. Fouillat "Time-resolved scanning of integrated circuits with a pulsed laser: application to transient fault injection in an ADC" IEEE Transactions on Instrumentation and Measurement, Vol. 53, N 4, pp. 1227-1231, Agosto, 2004.

[Poup05] A. L. Pouponnot "Strategic Use of SEE Mitigation Techniques for the Development of the ESA Microprocessors: Past, Present, and Future" 11th IEEE International On-Line Testing Symposium, 2005.

[Powe95] D. Powell, E. Martins, J. Arlat, Y. Crouzet "Estimators for fault tolerance coverage evaluation" IEEE Transactions on Computers, Vol. 44, Issue 2, pp. 261-274, Febrero, 1995.

[PowerPC] Xilinx Reference Guide, "PowerPC Processor", EDK 6.1, September 2, 2003.

[Prad96] D. K. Pradhan "Fault-Tolerant Computer System Design" Prentice Hall, 1996.

[Reba02] M. Rebaudengo, M. Sonza Reorda, M. Violante "Analysis of SEU effects in a pipelined processor" IEEE International On-line Testing Workshop, pp. 206-210, 2002.

[Reba99] M. Rebaudengo, M. Sonza Reorda "Evaluating the Fault Tolerance Capabilities of Embedded Systems via BDM" 17th IEEE VLSI Test Symposium, pp. 452-457, Dana Point, USA, Abril, 1999.

[Rezg01] S. Rezgui, R. Velazco, R. Ecoffet, S. Rodriguez, J. R. Mingo "Estimating error rates in processor-based architectures" IEEE Transactions on Nuclear Science, Vol. 48, Issue 5, pp. 1680-1687, Octubre, 2001.

[Schr07] R. D. Schrimpf, "Radiation Effects in Microelectronics". En R. Velazco, P. Fouillat, R. Reis (Eds.) "Radiation Effects on Embedded Systems", Holanda, Springer, 2007, pp. 11-29.

[Sieh97] V, Sieh, O. Tschäche, F. Balbach. "VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions", 27th International Symposium Fault Tolerant Computing, 1997.

[Sonz06] M. Sonza Reorda, L. Sterpone, M. Violante, M. Portela-García, C. Lopez-Ongil, L. Entrena "Fault Injection-based Reliability Evaluation of SoPCs" 11th IEEE European Test Symposium, pp. 75-82, Mayo 2006.

[Ster06] L. Sterpone, M. Violante "A new reliability-oriented place and route algorithm for SRAM-based FPGAs" IEEE Transactions on Computers, Vol. 55, No. 6, pp. 732-744, Junio, 2006.

[Ster07] L. Sterpone, M. Sonza Reorda, M. Violante, F. Lima Kastensmidt, L. Carro "Evaluating Different Solutions to Design Fault Tolerant Systems with SRAM-based FPGAs" Journal of Electronic Testing: Theory and Applications, 23, pp. 47-54, 2007.

[V2 Pro] Xilinx Product Specification, "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet", DS083 v4.5, October 10, 2005

[Varg00] F. Vargas, A. Amory, R. Velazco "Estimating Circuit Fault Tolerance by Means of Transient-Fault Injection in VHDL" 6th IEEE International On-Line Testing Workshop, pp. 67-72, Julio 2000.

[Varg05] F. Vargas, D.L. Cavalcante, E. Gatti, D. Prestes, D. Lupi, "On the proposition of an EMI-based fault injection approach", 11th IEEE International On-Line Testing Symposium, pp. 207-208, Julio 2005.

[Vela00] R. Velazco, S. Rezgui, R. Ecoffet, "Predicting Error Rate for Microprocessor-Based Digital Architectures Through C.E.U. (Code Emulating Upsets) Injection", IEEE Transactions on Nuclear Science, vol. 47, No. 6, pp. 2405-2411, Diciembre 2000.

[Vela92_a] R. Velazco, B. Martinet, G. Auvert "Laser Injection of Spot Defects on Integrated Circuits" 1st Asian Test Symposium, pp. 158-163, Noviembre 1992.

[Vela92_b] R. Velazco, S. Karoui, T. Chapuis, D. Benezech; L.H. Rosier; "Heavy ion test results for the 68020 microprocessor and the 68882 coprocessor" IEEE Transactions on Nuclear Science, Volume 39, Issue 3, Part 1-2, pp. 436-440, Junio 1992.

[Vela96] R. Velazco, T. Calin, M. Nicolaidis, S.C. Moss, S.D. LaLumondiere, V.T. Tran, R. Koga, "SEU-hardened storage cell validation using a pulsed laser" IEEE Transactions on Nuclear Science, Volume 43, Issue 6, Part 1, pp. 2843 – 2848, Diciembre 1996.

[WebServer] M. Muggli, M. Ouellette, S. Thammanur, "Web Server Reference Design Using a PowerPC-Based Embedded System", Xilinx Application Notes XAPP434 v.1.1, 3 de Noviembre, 2004.

[Xilinx] www.xilinx.com

[Zara03] H. R. Zarandi, S. G. Miremadi, A. Ejlali "Fault Injection into Verilog Models for Dependability Evaluation of Digital Systems" 2nd International Symposium on Parallel and Distributed Computing, 2003.