UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

DISCA

Fallos intermitentes: análisis de causas y efectos, nuevos modelos de fallos y técnicas de mitigación

Tesis Doctoral

Presentada por: **D. Luis José Saiz Adalid**

Dirigida por: **Dr. D. Pedro Gil Vicente**
Dr. D. Juan Carlos Baraza Calvo
Dr. D. Joaquín Gracia Morán

Valencia, 2015

Para Antonio, por retrasar esta tesis de esa forma tan dulce

Para Pepi, por sufrir esta tesis tanto como yo (o más),
y por darme a Antonio

"…pero no te abandonaremos, ni aun cuando la muerte nos lleve. La riqueza de nuestras vidas pasará a ti. Todo lo que tengo, lo que he aprendido, mis sentimientos, todo eso y más, pasará a ti, hijo mío. Seré tu compañero todos los días de mi vida. Harás de mi fuerza la tuya. Verás mi vida a través de tus ojos, y yo la tuya a través de los míos. El hijo se convertirá en padre y el padre, en hijo. Éste es mi legado, todo lo que puedo darte."

Jor-El (Marlon Brando)
Superman, 1975

"...el destino no está marcado al nacer,
yo he elegido ser lo que siempre seré..."

José Luis Campuzano
Hijos de Caín, En un lugar de la marcha
Barón Rojo, 1985

Prólogo

"Existen tres tipos de personas:
los vivos, los muertos y los marinos"
Anónimo

Durante la realización de esta tesis, y especialmente en la fase final, he pensado varias veces en la cita que encabeza este prólogo. La idea ha sido atribuida a distintos autores (Anacarsis, Sócrates, Platón...), y hace referencia a que, antiguamente, los que se aventuraban a navegar los mares no se sabía si estaban en el mundo de los vivos o de los muertos hasta que regresaban de sus travesías, que podían durar meses o años.

Y me parecía especialmente conveniente para este momento por dos motivos, uno más personal y otro más científico. En un aspecto más personal pensaba que, en la frase, se podría sustituir perfectamente a los marinos por los doctorandos en fase de elaboración de la tesis. Supongo que todos los doctores que lean esto entenderán perfectamente a qué me refiero. Cuántas veces no he estado "en el mundo de los vivos" por trabajar en esta tesis, cuántas renuncias personales, en cuántas ocasiones he tenido la sensación de ser un mal amigo, un mal hijo, un mal hermano, un mal marido, incluso un mal padre. Escribiendo estas líneas, he tenido que sacar a mi hijo llorando del despacho porque quería ir a Manzaneruela (mi pueblo). Espero llegar pronto a tierra, volver al mundo de los vivos, y poder compensar a todos.

Y ya en un plano más científico, hablemos del *hardware* de los sistemas informáticos y de los fallos que puede sufrir. Según su comportamiento temporal, los fallos se han clasificado históricamente en permanentes y transitorios. Cuando se produce un fallo permanente, el *hardware* está "muerto" y es necesario reemplazarlo para que el sistema pueda reanudar su servicio. Los fallos transitorios se producen en un *hardware* "vivo", que funciona bien, pero que falla por condiciones del entorno. Para completar la analogía, se puede decir que el *hardware* afectado por los fallos intermitentes (cada vez más frecuentes en los sistemas digitales actuales) no está "muerto" ya que, en general, funciona correctamente, pero tampoco está completamente "vivo", ya que hay un problema interno que, combinado con las condiciones del entorno, puede hacer que el sistema no funcione adecuadamente.

Sobre "no vivos" y "no muertos" versa esta tesis. O sea, sobre los fallos intermitentes en el *hardware* de los sistemas informáticos, sus causas, sus efectos y posibles técnicas de mitigación. Ésta ha sido mi travesía por el océano.

Agradecimientos

"No consideres como amigo al que siempre te alaba y no tiene valor para decirte tus defectos"
San Juan Bosco

En primer lugar, quiero expresar mi gratitud con mis directores de tesis, los doctores Pedro Gil, Juan Carlos Baraza y Joaquín Gracia, por toda la ayuda que me han prestado para permitirme alcanzar el doctorado. Sus consejos, su orientación y su paciencia han hecho que este "viejo novato" haya podido aprender algo acerca de la investigación científica. Si no he aprendido más, no es culpa suya.

Junto a ellos, el resto de componentes (presentes o pasados) del equipo de la línea de investigación sobre Sistemas Tolerantes a Fallos con los que he coincidido. En especial, a los doctores Daniel Gil, Juan Carlos Ruiz y David de Andrés, pero también a otros muchos (Pepe, Juan, Jesús, Jaume, Miquel…) con los que he compartido artículos, viajes a congresos, reuniones, etc. Todos ellos me han aportado conocimientos y nuevas experiencias. Gracias, compañeros y amigos.

También debo agradecer a la Universitat Politècnica de València y, especialmente, al Departamento de Informática de Sistemas y Computadores, por permitirme trabajar como docente e investigador. Este trabajo me apasiona y espero retornar la confianza depositada en mí con una buena labor docente y producción científica.

Me gustaría recordar a los amigos a los que he tenido un poco en el olvido por culpa de la tesis. Ya he comentado antes que, en ocasiones, he tenido la sensación de ser un mal amigo. Vicente, Pep, Abel, José Antonio, Rafa, Armando, Julián, Miguel… espero que después de defender la tesis podamos vernos más a menudo. Gracias por vuestra paciencia conmigo. Manzaneruela, espero empezar a pasar más tiempo allí.

No me puedo olvidar de mi familia. Gracias infinitas a mis padres por la educación que me han dado, y que me ha permitido llegar hasta donde he llegado; a mi mujer, Pepi, por cargar sobre sus espaldas el peso de nuestro hogar y la crianza de nuestro hijo, demostrando ser una gran mujer a pesar de las dificultades; a Antonio, la personita que llenó nuestras vidas y que ha ralentizado un poco el desarrollo de esta tesis, pero ha conseguido mantenerme pegado a la realidad y que no me volviese loco en alguna ocasión; y a todos los demás, porque de una forma u otra me han ayudado a ser como soy, a tener un entorno en el que poder sentirme vivo, o nos han ayudado a Pepi y a mí con el cuidado de Antonio. Ana, vosotros también sois parte de nuestra familia.

A todos ellos, y a quienes no he nombrado expresamente pero me aprecian y apoyan, muchísimas gracias de todo corazón.

Índice

Resumen

Desde la invención del primer circuito integrado hasta la tecnología de muy alta escala de integración (VLSI), el *hardware* de los sistemas informáticos ha evolucionado enormemente. La Ley de Moore, que vaticina que el número de transistores que se pueden integrar en un chip se duplica cada año, se ha venido cumpliendo durante décadas gracias a la agresiva reducción del tamaño de los transistores. Esto ha permitido aumentar su frecuencia de trabajo, logrando mayores prestaciones con menor consumo, pero a costa de penalizar la confiabilidad, ya que aumentan los defectos producidos por variaciones en el cada vez más complejo proceso de fabricación.

En la presente tesis se aborda el estudio de uno de los problemas fundamentales que afectan a la confiabilidad en las actuales y futuras tecnologías de circuitos integrados digitales VLSI: los fallos intermitentes. En el pasado, los fallos intermitentes se consideraban el preludio de fallos permanentes. En la actualidad, ha aumentado la aparición de fallos intermitentes provocados por variaciones en el proceso de fabricación que no afectan permanentemente. Los errores inducidos por fallos intermitentes se manifiestan de forma similar a los provocados por fallos transitorios, salvo que los fallos intermitentes suelen agruparse en ráfagas y se activan repetitivamente y de forma no determinista en el mismo lugar. Además, los fallos intermitentes se pueden activar y desactivar por cambios de temperatura, tensión y frecuencia.

En esta tesis se han analizado los efectos de los fallos intermitentes en sistemas digitales utilizando inyección de fallos basada en simulación, que permite introducir fallos en el sistema de forma controlada. Tras un amplio estudio bibliográfico para entender los mecanismos físicos de los fallos intermitentes, se han propuesto nuevos modelos de fallo en los niveles de puerta lógica y de transferencia de registros, que se han utilizado para analizar los efectos de los fallos intermitentes y la influencia de diversos factores.

Para mitigar esos efectos, en esta tesis se han estudiado distintas técnicas de tolerancia a fallos, con el objetivo de determinar si son adecuadas para tolerar fallos intermitentes, ya que las técnicas existentes están generalmente diseñadas para tolerar fallos transitorios o permanentes. Los resultados muestran que los mecanismos de detección funcionan adecuadamente, pero hay que mejorar los de recuperación.

Una técnica de tolerancia a fallos existente son los códigos correctores de errores (ECC). Esta tesis propone nuevos ECC diseñados para tolerar fallos cuando su tasa no es la misma en todos los bits de una palabra, como en el caso de los fallos intermitentes. Éstos, además, pueden presentar una tasa de fallo variable en el tiempo, por lo que sería necesario un mecanismo de tolerancia a fallos cuyo comportamiento se adapte a la evolución temporal de las condiciones de error, y que utilice los nuevos ECC propuestos.

Resum

Des de la invenció del primer circuit integrat fins a la tecnologia de molt alta escala d'integració (VLSI), el maquinari dels sistemes informàtics ha evolucionat enormement. La Llei de Moore, que vaticina que el nombre de transistors que es poden integrar en un xip es duplica cada any, s'ha vingut complint durant dècades gràcies a l'agressiva reducció de la mida dels transistors. Això ha permès augmentar la seua freqüència de treball, aconseguint majors prestacions amb menor consum, però a costa de penalitzar la fiabilitat, ja que augmenten els defectes produïts per variacions en el cada vegada més complex procés de fabricació.

En la present tesi s'aborda l'estudi d'un dels problemes fonamentals que afecten la fiabilitat en les actuals i futures tecnologies de circuits integrats digitals VLSI: les fallades intermitents. En el passat, les fallades intermitents es consideraven el preludi de fallades permanents. En l'actualitat, ha augmentat l'aparició de fallades intermitents provocades per variacions en el procés de fabricació que no afecten permanentment. Els errors induïts per fallades intermitents es manifesten de forma similar als provocats per fallades transitòries, llevat que les fallades intermitents solen agrupar-se en ràfegues i s'activen repetidament i de forma no determinista en el mateix lloc. A més, les fallades intermitents es poden activar i desactivar per canvis de temperatura, tensió i freqüència.

En aquesta tesi s'han analitzat els efectes de les fallades intermitents en sistemes digitals utilitzant injecció de fallades basada en simulació, que permet introduir errors en el sistema de forma controlada. Després d'un ampli estudi bibliogràfic per entendre els mecanismes físics de les fallades intermitents, s'han proposat nous models de fallada en els nivells de porta lògica i de transferència de registres, que s'han utilitzat per analitzar els efectes de les fallades intermitents i la influència de diversos factors.

Per mitigar aquests efectes, en aquesta tesi s'han estudiat diferents tècniques de tolerància a fallades, amb l'objectiu de determinar si són adequades per tolerar fallades intermitents, ja que les tècniques existents estan generalment dissenyades per tolerar fallades transitòries o permanents. Els resultats mostren que els mecanismes de detecció funcionen adequadament, però cal millorar els de recuperació.

Una tècnica de tolerància a fallades existent són els codis correctors d'errors (ECC). Aquesta tesi proposa nous ECC dissenyats per tolerar fallades quan la seua taxa no és la mateixa en tots els bits d'una paraula, com en el cas de les fallades intermitents. Aquests, a més, poden presentar una taxa de fallada variable en el temps, pel que seria necessari un mecanisme de tolerància a fallades on el comportament s'adapte a l'evolució temporal de les condicions d'error, i que utilitze els nous ECC proposats.

Abstract

From the first integrated circuit was developed to very large scale integration (VLSI) technology, the hardware of computer systems has had an immense evolution. Moore's Law, which predicts that the number of transistors that can be integrated on a chip doubles every year, has been accomplished for decades thanks to the aggressive reduction of transistors size. This has allowed increasing its frequency, achieving higher performance with lower consumption, but at the expense of a reliability penalty. The number of defects are raising due to variations in the increasingly complex manufacturing process.

Intermittent faults, one of the fundamental issues affecting the reliability of current and future digital VLSI circuits technologies, are studied in this thesis. In the past, intermittent faults have been considered the prelude to permanent faults. Nowadays, the occurrence of intermittent faults caused by variations in the manufacturing process not affecting permanently has increased. Errors induced by intermittent and transient faults manifest similarly, although intermittent faults are usually grouped in bursts and they are activated repeatedly and non-deterministically in the same place. In addition, intermittent faults can be activated and deactivated by changes in temperature, voltage and frequency.

In this thesis, the effects of intermittent faults in digital systems have been analyzed by using simulation-based fault injection. This methodology allows introducing faults in a controlled manner. After an extensive literature review to understand the physical mechanisms of intermittent faults, new intermittent fault models at gate and register transfer levels have been proposed. These new fault models have been used to analyze the effects of intermittent faults in different microprocessors models, as well as the influence of several parameters.

To mitigate these effects, various fault tolerance techniques have been studied in this thesis, in order to determine whether they are suitable to tolerate intermittent faults. Results show that the error detection mechanisms work properly, but the error recovery mechanisms need to be improved.

Error correction codes (ECC) is a well-known fault tolerance technique. This thesis proposes a new family of ECCs specially designed to tolerate faults when the fault rate is not equal in all bits in a word, such as in the presence of intermittent faults. As these faults may also present a fault rate variable along time, a fault tolerance mechanism whose behavior adapts to the temporal evolution of error conditions can use the new ECCs proposed.

1

Presentación

1.1 Fundamentos y motivación

Los sistemas electrónicos e informáticos ocupan un papel muy importante en el día a día de nuestra sociedad: desde los hábitos personales cotidianos (uso de teléfonos inteligentes, tabletas, computadores personales, etc.) hasta la gestión de empresas (públicas y privadas), pasando por el control industrial (proceso de fabricación, plantas de energía, etc.), gestión de sistemas de transporte (terrestre, marítimo y aeroespacial), previsión de catástrofes (control meteorológico, predicción de terremotos, etc.), actividades militares, etc. Todo está controlado por sistemas informáticos. Por ello, es prioritario conseguir que estos sistemas duren el mayor tiempo posible y con un elevado nivel de eficacia, especialmente aquellos en los que un mal funcionamiento puede causar impacto económico, suspensión de servicios básicos o pérdida de vidas humanas.

Los **Sistemas Tolerantes a Fallos** son sistemas que disponen de mecanismos especiales que proporcionan una cierta inmunidad a la ocurrencia de fallos que puedan causar un cese o deterioro del servicio prestado. Se denomina **confiabilidad** al conjunto de atributos que permiten cuantificar la calidad del servicio prestado y el grado de confianza que el usuario puede depositar en el sistema. Las amenazas de la Confiabilidad son los fallos, errores y averías, que son circunstancias no deseadas que pueden tener como resultado la pérdida de la confianza en el sistema. Para evitarla, se dispone de distintos medios para conseguir la Confiabilidad: prevención de fallos, tolerancia a fallos, eliminación de fallos y predicción de fallos [Laprie92, Avizienis04].

Una de las amenazas de la Confiabilidad que está cobrando especial importancia en los últimos años son los **fallos intermitentes** [Constantinescu03]. Según su comportamiento temporal, los fallos se han clasificado históricamente en **fallos transitorios**, causados por circunstancias ambientales, y **fallos permanentes**, producidos por un defecto irreversible en el *hardware*. Los fallos intermitentes se producen por un *hardware* defectuoso que, en ocasiones, funciona correctamente, pero que ante determinadas variaciones del entorno, como alta temperatura o carga de trabajo elevada, puede no hacerlo. Dado que uno de los mecanismos que pueden provocar esta situación es el desgaste y envejecimiento de los dispositivos electrónicos, y que normalmente termina en un fallo permanente, los fallos intermitentes se han considerado un preludio de éstos. Por ello, apenas se hacían estudios específicos para los fallos intermitentes, y los mecanismos de tolerancia a fallos aplicados eran los mismos que para los fallos permanentes.

En los últimos años, con el aumento de la escala de integración, y la consiguiente reducción de tamaño de los dispositivos electrónicos, están apareciendo nuevos mecanismos físicos que pueden provocar fallos intermitentes que no desembocan en fallos permanentes. Numerosos estudios recientes confirman la importancia de los fallos intermitentes en los sistemas informáticos. En [Constantinescu08], por ejemplo, se comprueba que el 6,2% de los errores producidos en los subsistemas de memoria de los sistemas estudiados son debidos a fallos intermitentes. En [Schroeder09] se confirma la prevalencia de los fallos en el *hardware* de las memorias RAM dinámicas, pero esos fallos no son, en buena parte, permanentes, sino que se activan y desactivan repetitivamente en el mismo sitio. En [Nightingale11] se estima que el 39% de las máquinas que han sufrido una avería en la CPU han tenido como causa un fallo intermitente, siendo también importantes los porcentajes que se obtienen para las averías en memoria y discos duros. Estos tres estudios tienen en común que obtienen sus conclusiones tras la observación de sistemas reales y con una cantidad elevada de datos (el primero observa 257 servidores, acumulando en total 310,7 años de trabajo; el segundo analiza los servidores de *Google* durante 2,5 años; y el tercero analiza los datos del *Windows Error Reporting system*, con 755.539 registros).

Esta tesis se enmarca dentro del trabajo realizado en la línea de investigación sobre Sistemas Tolerantes a Fallos del instituto ITACA, y culmina los estudios de Doctorado del autor, dentro del programa "Arquitectura de los sistemas informáticos en red y sistemas empotrados", ofertado por el Departamento de Informática de Sistemas y Computadores (DISCA). Tanto el instituto ITACA como el departamento DISCA pertenecen a la Universitat Politècnica de València.

1.2 Objetivos

El objetivo general de esta tesis es el estudio de los fallos intermitentes, sus mecanismos físicos, sus efectos en los sistemas informáticos actuales, el comportamiento de las técnicas de mitigación existentes y la propuesta de una nueva metodología que permita diseñar mecanismos de tolerancia a fallos dirigidos principalmente a tolerar fallos intermitentes. Para conseguir este objetivo general es necesario considerar varios objetivos parciales, que en su conjunto permiten alcanzar una visión completa del problema que suponen los fallos intermitentes en los sistemas informáticos actuales.

El primer paso es estudiar las causas y mecanismos físicos que pueden provocar fallos intermitentes. Este estudio hace posible entender mejor en qué lugares pueden aparecer estos fallos y cuál puede ser el comportamiento de los componentes afectados.

La segunda etapa consiste en, a partir de las causas y mecanismos físicos, obtener una serie de modelos de fallos intermitentes. Para realizar el estudio de los efectos de los fallos intermitentes en los sistemas informáticos, se utilizará la inyección de fallos mediante simulación. Para ello, además de los modelos de los sistemas bajo estudio, es necesario disponer de unos modelos de fallo representativos, que permitan simular el comportamiento del sistema ante fallos intermitentes de la forma más cercana posible a la realidad.

En la tercera fase se estudian los efectos de los fallos intermitentes en sistemas informáticos. Para ello, se utiliza la inyección de fallos mediante simulación, por lo que se necesita el modelo del sistema bajo estudio, modelos de los fallos a los que se hace referencia en la etapa anterior, y una herramienta que permita simular la inyección de fallos. Estas simulaciones permiten valorar el impacto de los fallos intermitentes y comprobar que sus efectos son significativos en los sistemas informáticos actuales.

A continuación, hay que analizar las principales técnicas de mitigación existentes actualmente, y comprobar si responden adecuadamente ante los fallos intermitentes. Para ello se han realizado nuevas simulaciones, inyectando fallos en un sistema microprocesador tolerante a fallos, es decir, equipado con diversas técnicas de mitigación. Este análisis permite valorar si el comportamiento de las diversas técnicas es adecuado ante los fallos intermitentes.

Por último, y dado que del análisis se ha constatado que las técnicas de tolerancia a fallos actuales no responden de manera adecuada ante los fallos intermitentes, o que su comportamiento podría ser mejorable, el objetivo final de esta tesis es hacer propuestas que permitan aumentar la confiabilidad de un sistema informático en el que puedan aparecer fallos intermitentes.

1.3 Desarrollo

Esta tesis recoge el trabajo realizado para la consecución de los objetivos anteriormente expuestos, y describe los resultados obtenidos. A continuación se especifica la organización de esta tesis.

El capítulo 2, titulado "Conceptos básicos sobre Tolerancia a Fallos", está dedicado a la terminología utilizada en el campo de la Tolerancia a Fallos. En él se establece la relación entre fallo, error y avería, y se define el concepto de Confiabilidad y sus atributos. En este capítulo también se enumeran los diferentes métodos existentes para evaluar la Confiabilidad de un sistema, haciendo especial hincapié en la validación de forma experimental y, en particular, en la validación mediante inyección de fallos. Por su importancia para esta tesis, se incluye un apartado que introduce la teoría de codificación de la información y los códigos detectores y correctores de errores, especialmente aquellos que son adecuados para codificar la información utilizada en el interior de un procesador, en los buses del sistema o en la memoria.

En el capítulo 3, "Efectos de los fallos intermitentes", se realiza un estudio bibliográfico de las causas y mecanismos físicos de los fallos intermitentes, con el fin de obtener una serie de modelos de fallo que sean representativos de los fallos intermitentes. A continuación se utilizan los modelos obtenidos para simular su aparición en sistemas informáticos no tolerantes a fallos y poder evaluar sus efectos, la influencia de los distintos parámetros de los modelos, y llevar a cabo diversos análisis comparativos. Finalmente, se estudian las técnicas de mitigación existentes, realizando nuevas simulaciones sobre un sistema tolerante a fallos equipado con diversas técnicas de mitigación representativas, y se analiza su comportamiento. Finalmente, se presentan las conclusiones obtenidas, que dan pie a la búsqueda de nuevas técnicas de tolerancia a fallos que respondan mejor a la aparición de fallos intermitentes.

El capítulo 4, "*Flexible Unequal Error Control* y otros códigos correctores de errores", motiva el uso de los códigos detectores y correctores de errores como herramienta que permite proteger la información manejada en un sistema informático, propone una metodología para diseñar códigos correctores de errores en función de los vectores de errores a corregir o detectar. Utilizando esta metodología se han obtenido unos nuevos códigos, denominados *Flexible Unequal Error Control* (FUEC), que son especialmente útiles cuando la tasa de error varía de unos bits a otros, como sucede ante la aparición de fallos intermitentes. También se expone cómo se podrían utilizar estos códigos en un sistema tolerante a fallos adaptativo, capaz de detectar la aparición de un fallo intermitente y reconfigurarse dinámicamente para una mejor tolerancia de dicho fallo.

La metodología presentada ha permitido obtener otros códigos, especialmente indicados para tolerar fallos múltiples adyacentes, cada vez más frecuentes en sistemas VLSI. Estos códigos también se presentan en el capítulo 4.

Para finalizar, en el capítulo 5, "Conclusiones y trabajo futuro", se hace una recapitulación del trabajo presentado, se extraen las conclusiones más interesantes a las que se ha llegado durante el desarrollo de la presente tesis, y se explican las líneas de investigación abiertas, que constituyen el trabajo ya en curso y futuro.

2

Conceptos básicos sobre Tolerancia a Fallos

2.1 Introducción

En este capítulo se van a introducir una serie de conceptos que se utilizan en el campo de la Tolerancia a Fallos. A veces, estos conceptos se utilizan en la bibliografía con significados diferentes, y a menudo contradictorios. Estas divergencias en la interpretación surgen de las diferentes traducciones al castellano que se hacen de los términos ingleses o franceses. En este sentido, la terminología y definiciones que se aportan en este trabajo siguen las tendencias mostradas en [Avizienis04] y [PGil06] como referentes en el campo de la Tolerancia a Fallos.

Al final del capítulo, y por la especial relevancia que tienen para esta tesis, se incluyen dos apartados específicos. El primero de ellos resume las técnicas de inyección de fallos con mayor aceptación dentro de la comunidad científica, enfatizando las técnicas de inyección de fallos basadas en simulación de modelos. El último apartado introduce la teoría de codificación de la información y los códigos detectores y correctores de errores, especialmente aquellos que son adecuados para codificar la información utilizada en el interior de un procesador, en los buses del sistema o en la memoria.

2.2 Definiciones básicas

La **confiabilidad** (del inglés *dependability*) se define en [Laprie92] como "la propiedad de un sistema informático que permite depositar una confianza justificada en el servicio que proporciona. El servicio proporcionado por un sistema es el comportamiento percibido por su o sus usuarios; un usuario es otro sistema (físico o humano) que interacciona con el primero". Asimismo, se dice que la confiabilidad de un sistema es la capacidad de evitar averías de servicio que sean más frecuentes y más graves de lo aceptable [Avizienis04].

La confiabilidad puede ser vista de acuerdo a diferentes (aunque complementarias) propiedades, lo que permite la definición de sus atributos:

- La disposición para un servicio correcto se denomina **disponibilidad** (del inglés *availability*).
- La continuidad de un servicio correcto se denomina **fiabilidad** (del inglés *reliability*).
- La no ocurrencia de consecuencias catastróficas en el entorno se denomina **inocuidad** (del inglés *safety*).
- La no ocurrencia de revelaciones no autorizadas de información se denomina **confidencialidad** (del inglés *confidentiality*).
- La ausencia de alteraciones impropias del sistema se denomina **integridad** (del inglés *integrity*).
- La capacidad para someterse a reparaciones y modificaciones se denomina **mantenibilidad** (del inglés *maintainability*).
- La asociación de la integridad y disponibilidad respecto a acciones autorizadas, junto con la confidencialidad, se denomina **seguridad** (del inglés *security*).

Según las aplicaciones a las que está destinado el sistema, se pone énfasis en diferentes facetas de la confiabilidad.

En la figura 2.1 se puede ver un resumen de las relaciones entre confiabilidad y seguridad en términos de sus principales atributos. La figura indica dónde cae en cada caso el mayor balance de interés y actividad.

La especificación de la confiabilidad y la seguridad de un sistema debe incluir los requisitos relativos a los atributos, en términos de la gravedad y frecuencia aceptable para

las averías de servicio, para las clases de fallos especificadas y para un entorno de uso dado.

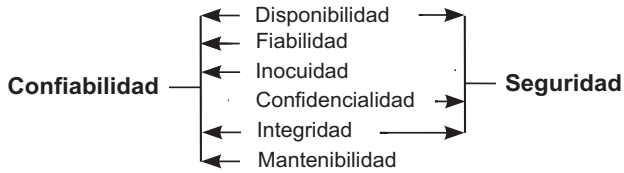

Figura 2.1: Atributos de la confiabilidad y la seguridad.

Una avería del sistema ocurre cuando el servicio entregado se desvía del cumplimiento de la función del sistema, siendo esta función a la que el sistema está destinado. Un error es la parte del estado del sistema responsable de llevar a éste a una avería: un error que afecta al servicio es una indicación de que la avería está ocurriendo o ha ocurrido. Un fallo es la causa, justificada o hipotética, de un error.

El desarrollo de sistemas confiables requiere la utilización combinada de un conjunto de medios que pueden ser clasificados como:

- **Prevención de fallos**: cómo prevenir la ocurrencia o la introducción de fallos.
- **Tolerancia a fallos**: cómo evitar la ocurrencia de averías de servicio en presencia de fallos.
- **Eliminación de fallos**: cómo reducir la presencia (número, gravedad) de los fallos.
- **Predicción de fallos**: cómo estimar el número actual, la incidencia futura, la probabilidad y las consecuencias de los fallos.

La prevención y la tolerancia a fallos tienen como objetivo proveer al sistema de la capacidad de entregar un servicio en el que se pueda confiar, mientras que la eliminación y la predicción de fallos tienen el objetivo de alcanzar la confianza en esta capacidad, justificando que las especificaciones funcionales, de confiabilidad y de seguridad son las adecuadas, y que el sistema las cumple con una determinada probabilidad.

Los conceptos que se acaban de definir pueden ser agrupados en tres categorías, como se ve en la figura 2.2, que define el esquema de la taxonomía completa de la computación confiable y segura:

- Las **amenazas** de la confiabilidad: fallos, errores y averías. Son circunstancias no deseadas, pero no inesperadas en principio, que causan la no confiabilidad.

- Los **medios** para conseguir la confiabilidad: prevención de fallos, tolerancia a fallos, eliminación de fallos, predicción de fallos. Son métodos y técnicas para dar al sistema capacidad para entregar un servicio en el que se pueda confiar, y tener confianza en esa capacidad.
- Los **atributos** de la confiabilidad: disponibilidad, fiabilidad, inocuidad, confidencialidad, integridad y mantenibilidad. Permiten expresar las propiedades que se esperan de un sistema y valorar la calidad del servicio entregado.

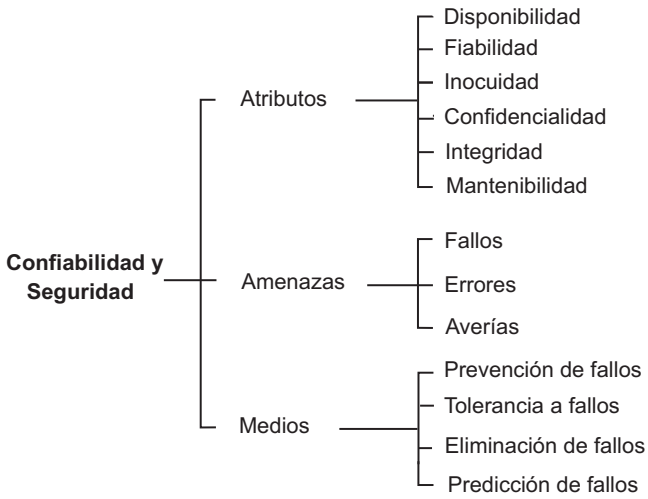

Figura 2.2: Árbol de la confiabilidad.

2.3 Atributos de la confiabilidad

Los atributos de la confiabilidad se han definido en la sección anterior de acuerdo con diversas propiedades sobre las que se puede poner más o menos atención según la aplicación a la que esté destinado el sistema informático considerado:

- La disponibilidad se requiere siempre, aunque a niveles variables dependientes de la aplicación.
- La fiabilidad, la inocuidad y la confidencialidad se pueden requerir o no, dependiendo de la aplicación.

La integridad se puede definir como la ausencia de alteraciones indebidas de la información.

La mantenibilidad es una medida del tiempo de reparación o restauración del sistema desde la última avería ocurrida, referida al tiempo transcurrido entre el estado de servicio inadecuado y el de servicio adecuado.

Según sus definiciones, fiabilidad y disponibilidad enfatizan la capacidad de evitar las averías, mientras que la inocuidad enfatiza la capacidad de evitar una clase específica de averías (las averías catastróficas).

El énfasis sobre los diferentes atributos de la confiabilidad incidirá directamente en el tipo de técnicas que se implementen para conseguir que el sistema resultante sea confiable. Sobre todo, hay que tener en cuenta que algunos de los atributos son antagónicos, así que se hace necesario encontrar una solución de compromiso. Cuando se consideran las tres dimensiones principales del desarrollo de un sistema informático (costes, prestaciones y confiabilidad) el problema incluso empeora al no tener tanto dominio de la confiabilidad como de las otras dos.

2.4 Amenazas de la confiabilidad

En esta sección se examinan los conceptos de avería, error y fallo, así como sus mecanismos de manifestación, es decir, la patología de los fallos.

2.4.1 Averías

La ocurrencia de averías se ha definido con respecto a la función del sistema, no respecto a su especificación. Esto es debido a que, en realidad, si se identifica como avería a un comportamiento inaceptable debido a la no conformidad con la especificación, puede suceder que este comportamiento cumpla con la especificación y, sin embargo, sea inaceptable para los usuarios del sistema, dejando al descubierto, por tanto, un fallo de especificación. Además, reconocer que el evento es indeseable (y es de hecho una avería), solamente puede ser llevado a cabo después de su ocurrencia, por ejemplo, a través de sus consecuencias.

Generalmente, un sistema no se avería siempre de la misma forma. Las formas según las que un sistema puede averiarse son sus modos de avería, que pueden describirse

desde cuatro puntos de vista: el dominio de la avería, la detectabilidad de las averías, la coherencia de las averías y las consecuencias de éstas sobre el entorno.

El punto de vista del dominio de la avería conduce a distinguir entre **averías de valor** (en las que el valor del servicio entregado no cumple con la función del sistema) y **averías de tiempo** (en las que el tiempo en el que se entrega el servicio no cumple con la función del sistema).

Estas definiciones generales deben refinarse más. Por ejemplo, la noción de avería de tiempo puede refinarse en **averías con adelanto** y **averías con retraso**, dependiendo de si el servicio se ha entregado demasiado pronto o demasiado tarde. Una clase de averías relacionadas a la vez con el valor y el tiempo son las **averías con parada**, que ocurren cuando la actividad del sistema, si es que hay alguna, no es perceptible por sus usuarios.

En relación a cómo el sistema interacciona con sus usuarios, tal ausencia de actividad puede tomar forma de:

a) **Salidas congeladas**: se entrega un servicio de valor constante; este valor constante puede variar según la aplicación, pudiendo ser el último valor correcto, algún valor por defecto, etc.

b) **Silencio**: no se entrega servicio alguno en la interfaz de servicio. Por ejemplo, no se envía ningún mensaje en un sistema distribuido.

Un sistema cuyas averías son solamente averías con parada es un **sistema con parada tras avería**. Las situaciones de salidas congeladas o de silencio dan lugar, respectivamente a los **sistemas pasivos tras avería** y a los **sistemas con silencio tras avería**.

En contraposición a las averías con parada, las **averías erráticas** se producen cuando existe entrega de servicio, pero se generan valores o mensajes incorrectos.

El punto de vista de la detectabilidad indica la señalización de las averías al usuario. La señalización en el interfaz del servicio se origina en los mecanismos de detección del sistema, que comprueban la corrección del servicio ofrecido. Cuando se detecta un servicio incorrecto y se alerta con una señal de aviso, suceden las **averías señaladas**. Si se produce un servicio incorrecto y no es detectado, se producen **averías no señaladas**.

El punto de vista de la coherencia de la avería lleva a distinguir, en caso de varios usuarios, entre **averías coherentes** (en las que todos los usuarios del sistema tienen la misma percepción de las averías) y **averías incoherentes** (en las que los usuarios del sistema pueden percibir una avería dada de forma diferente). Las averías incoherentes son a menudo clasificadas como **averías bizantinas**. Hay que resaltar que las averías de un

sistema con silencio tras avería son coherentes, mientras que en un *sistema pasivo tras avería* pueden no serlo.

Por otra parte, la gravedad de las averías es el resultado de clasificar las consecuencias de las averías sobre el entorno del sistema, por medio de la ordenación de los modos de avería según diferentes niveles de gravedad, a los que se asocian generalmente las probabilidades máximas de ocurrencia permisibles. El número, el etiquetado y la definición de los niveles de gravedad, así como las probabilidades de ocurrencia admisibles son, en gran parte, dependientes de las aplicaciones. Sin embargo, pueden definirse dos niveles extremos, de acuerdo con la relación entre el beneficio proporcionado por el servicio entregado por el sistema en ausencia de avería y las consecuencias de las averías:

- **Averías menores**, en las que las consecuencias son de un orden de magnitud igual que el beneficio obtenido por el servicio entregado en ausencia de averías.
- **Averías catastróficas**, en las que las consecuencias son inconmensurablemente superiores al beneficio obtenido por el servicio entregado en ausencia de averías.

Un sistema en donde todas las averías son, en una medida aceptable, menores o benignas, es un **sistema inocuo tras avería**.

La figura 2.3 resume esquemáticamente las clases de averías.

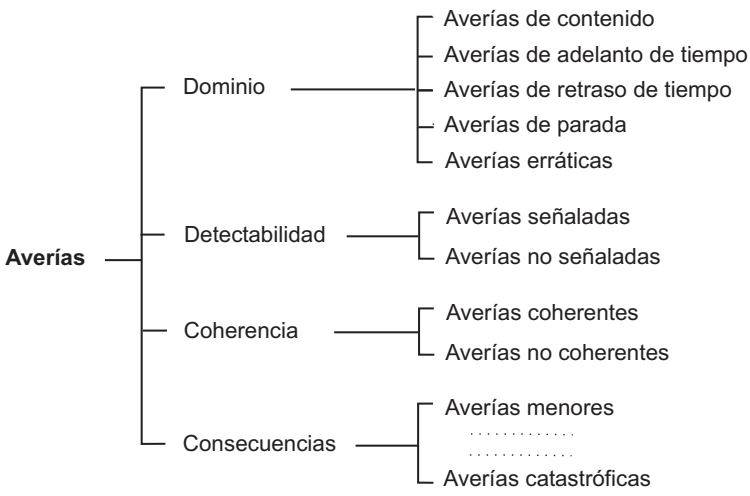

Figura 2.3: Clases de averías [Avizienis04].

2.4.2 Errores

El error se define como la parte del estado total de un sistema que puede conducir a una avería. Así pues, una avería ocurre cuando el error hace que el servicio entregado se desvíe del servicio correcto. A la causa del error se le denomina fallo. Un error es detectado si se señala su presencia por medio de un mensaje o una señal de error. Los errores que están presentes, pero que no son detectados, se denominan **errores latentes**. Puesto que un sistema consiste en una serie de componentes que interaccionan, el estado total del sistema es el conjunto de estados de sus componentes. La definición implica que un fallo causa originalmente un error dentro del estado de uno o más componentes, pero la avería del servicio no ocurrirá mientras el estado externo de ese componente no forme parte del estado externo del sistema. Una vez que el error se convierta en parte del estado externo del componente, ocurrirá una avería de servicio de ese componente, pero el error todavía será interno al sistema completo. El hecho de que un error conduzca o no a una avería depende de tres factores principales:

a) De la composición del sistema y, particularmente, de la naturaleza de la redundancia existente:

   - **Redundancia extrínseca o intencionada** es la introducida para tolerar los fallos. Está destinada explícitamente a evitar que un error dé lugar a una avería.

   - **Redundancia intrínseca o no intencionada** que puede tener el mismo efecto (aunque inesperado) que la redundancia intencionada. En la práctica es muy difícil, si no imposible, construir un sistema sin alguna forma de redundancia.

b) De la actividad del sistema, ya que un dato erróneo puede ser sobrescrito con un valor correcto antes de que produzca daños.

c) De la definición de avería desde el punto de vista del usuario, ya que lo que es una avería para un usuario dado puede no ser más que una molestia soportable para otro.

Una clasificación conveniente de errores consiste en describirlos en términos de las averías elementales de servicio que causa. Así, se tienen errores de contenido frente a errores de temporización, errores detectados contra errores latentes, errores coherentes frente a errores incoherentes (cuando el servicio va dirigido a dos o más usuarios), o errores menores contra errores catastróficos.

2.4.3 Fallos

Todos los fallos que pueden afectar a un sistema durante su vida se pueden clasificar de acuerdo a ocho puntos de vista básicos, que dan lugar a las clases de **fallos elementales**, mostradas en la figura 2.4.

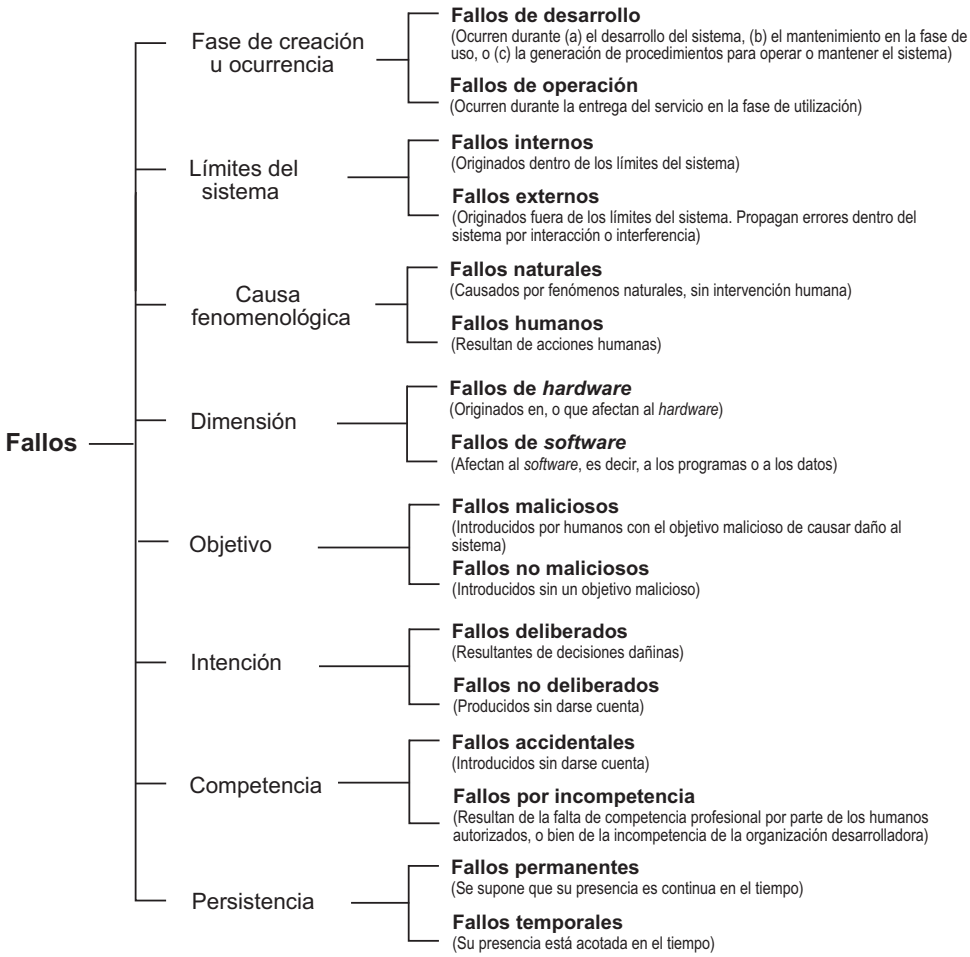

Figura 2.4: Clases de fallos elementales. Basada en [Avizienis04].

El momento de su introducción, desde el punto de vista de la vida del sistema, lleva a distinguir entre:

- **Fallos de desarrollo**, resultantes de imperfecciones originadas, bien a lo largo del desarrollo del sistema (desde la especificación de las necesidades hasta la implementación), o durante las modificaciones posteriores, o bien al establecer los procedimientos de explotación y mantenimiento del sistema.
- **Fallos de operación**, que aparecen durante la explotación del sistema.

Las fronteras del sistema llevan a distinguir entre:

- **Fallos internos**, originados dentro de los límites del sistema.
- **Fallos externos**, que son el resultado de interferencias o de la interacción con su entorno físico.

Las causas fenomenológicas llevan a distinguir entre:

- **Fallos naturales**, debidos a fenómenos naturales adversos.
- **Fallos humanos**, que resultan de acciones humanas.

Según la dimensión de los fallos se tiene:

- **Fallos de *hardware***: originados en el propio *hardware* o que afectan a éste.
- **Fallos de *software***: que afectan al *software* (a los programas o a los datos).

El objetivo de los fallos lleva a distinguir entre:

- **Fallos maliciosos**: cuando son introducidos por humanos, bien durante el desarrollo del sistema o directamente durante su uso, con el objetivo de causar algún daño al sistema.
- **Fallos no maliciosos**: cuando son introducidos sin el objetivo de dañar.

La naturaleza de los fallos conduce a distinguir entre:

- **Fallos no deliberados**, que aparecen o son creados de manera fortuita.
- **Fallos deliberados**, que son debidos a malas decisiones, es decir, acciones intencionadas que son erróneas y causan fallo.

La competencia profesional de los humanos que desarrollan o manejan un sistema nos lleva a distinguir entre:

- **Fallos accidentales**, que son introducidos sin darse cuenta, accidentalmente.
- **Fallos por incompetencia**, que resultan de una falta de competencia profesional de los humanos autorizados.

La persistencia temporal de los fallos conduce a distinguir entre:

- **Fallos permanentes**, cuya presencia no está ligada a condiciones puntuales, sean internas (como la actividad computacional) o externas (como el entorno).
- **Fallos temporales**, cuya presencia está ligada a aquellas condiciones y están, por tanto, presentes un tiempo limitado. Bajo esta denominación se incluyen los fallos temporales externos, denominados **fallos transitorios** (originados por condiciones ambientales), y los fallos temporales internos o **fallos intermitentes**, que son el resultado de la combinación de defectos internos con condiciones de operación internas o externas.

Esta tesis se centra en el estudio de los fallos intermitentes en el *hardware* de los sistemas informáticos, sus causas y mecanismos físicos, sus efectos en dichos sistemas y la efectividad de distintas técnicas de tolerancia a fallos.

2.4.4 Patología de los fallos

Los mecanismos de creación y manifestación de los fallos, errores y averías se ilustran en la figura 2.5, y se pueden resumir como sigue:

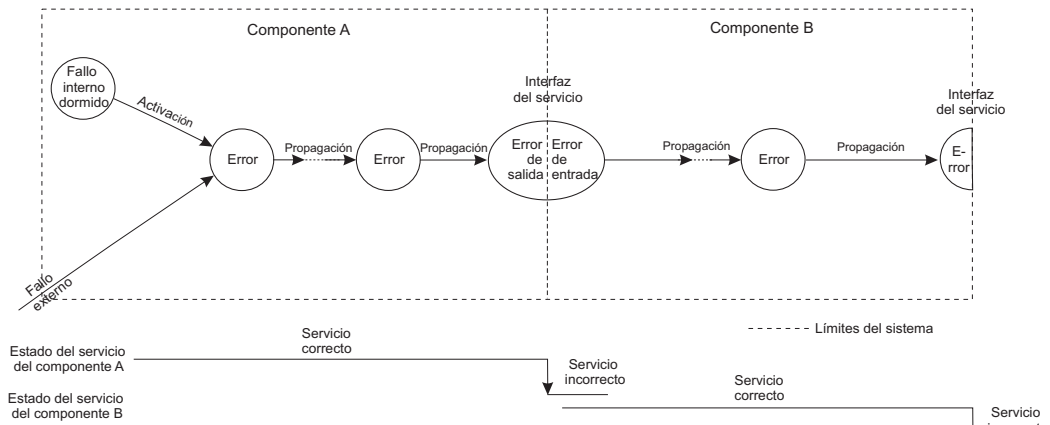

Figura 2.5: Propagación de los errores.

1. Un fallo se denomina **activo** cuando produce un error. Si no, es un fallo **inactivo**. Un fallo activo puede ser i) un fallo interno que era previamente inactivo y que ha sido activado por el proceso de cómputo o por condiciones ambientales, o bien ii) un fallo externo. La activación del fallo es la aplicación de una entrada (el patrón de la activación) a un componente que hace que el fallo inactivo se convierta en activo. La mayoría de los fallos internos están conmutando entre sus estados inactivo y activo.
2. La propagación del error dentro del componente dado (es decir, la propagación **interna**) es causada por el proceso de cómputo: el error se transforma sucesivamente en otros errores. La propagación del error de un componente A a otro componente B que recibe servicio de A (es decir, la propagación **externa**) ocurre cuando, a través de la propagación interna, un error alcanza la interfaz de servicio del componente A. En este momento, el servicio entregado por A a B pasa a ser incorrecto, y la avería de servicio de A que sobreviene aparece como un fallo externo en B, propagándose el error dentro de B a través de su interfaz de utilización.
3. Una **avería de servicio** ocurre cuando el error se propaga a la interfaz de servicio y causa que el servicio entregado por el sistema se desvíe del servicio correcto. La avería de un componente causa un fallo permanente o transitorio en el sistema que contiene este componente. La avería de servicio de un sistema causa un fallo externo permanente o transitorio para el otro sistema que recibe servicio del sistema dado.

Estos mecanismos permiten completar la **cadena fundamental** de amenazas de la confiabilidad y la seguridad, mostrada en la figura 2.6:

Figura 2.6: Cadena fundamental de amenazas de la confiabilidad y la seguridad.

Las flechas de esta cadena expresan la relación de causalidad entre fallos, errores y averías. No deben ser interpretadas de forma restringida, ya que mediante propagación pueden producirse varios errores antes de que se ocasione una avería, y un error puede provocar un fallo sin que se haya observado una avería (en caso de que no se realice esta observación), ya que una avería es un evento que ocurre en el interfaz entre dos componentes. La transformación entre los estados de fallo, error y avería no se produce

de manera simultánea en el tiempo. Así, desde que ocurre el fallo hasta que se manifiesta el error existe un tiempo de inactividad, llamado **latencia del error**. Durante este tiempo se dice que el fallo no es efectivo y que el error está latente. De forma análoga, se puede definir la latencia de detección del error y la latencia de producción de la avería.

Con frecuencia se encuentran situaciones donde están implicados múltiples fallos y/o averías. El tener en cuenta estos casos lleva a distinguir entre **fallos independientes**, que son atribuidos a diferentes causas, y **fallos conexos**, atribuidos a una causa común. Los fallos conexos se manifiestan con errores similares, mientras que los fallos independientes causan normalmente errores distintos, aunque a veces puede suceder que estos últimos produzcan errores similares. Los errores similares causan averías de modo común. La generalización de la noción de errores parecidos da lugar a la noción de errores coincidentes, es decir, errores creados a partir de la misma entrada. La relación temporal entre las averías múltiples lleva a distinguir entre las averías simultáneas, que ocurren en la misma ventana de tiempo predefinida, y averías secuenciales, que no ocurren en la misma ventana de tiempo predefinida.

2.5 Medios para conseguir confiabilidad

De los cuatro medios definidos en el apartado 2.2, en este apartado se van a examinar la tolerancia a fallos, la eliminación de fallos y la predicción de fallos. La prevención de fallos no se va a tratar, ya que claramente se refiere a la ingeniería general de sistemas.

2.5.1 Tolerancia a fallos

La tolerancia a fallos se lleva a cabo mediante el **procesamiento de los errores** y el **tratamiento de los fallos**. El procesamiento de los errores está destinado a eliminar los errores del estado computacional, a ser posible antes de que ocurra una avería. El tratamiento de los fallos está destinado a prevenir que se activen uno o varios fallos de nuevo. El procesamiento de los errores se puede realizar por medio de tres principios de diseño:

- Mediante la **detección de errores**, que permite identificar como tal a un estado erróneo.
- Mediante el **diagnóstico de errores**, que permite apreciar los daños producidos por el error detectado, o por los propagados antes de la detección.
- Mediante la **recuperación de errores**, donde se sustituye el estado erróneo por otro libre de errores. Esta sustitución puede hacerse de tres formas:
  a) Mediante la **recuperación hacia atrás**, donde el estado erróneo se sustituye por otro correcto sucedido antes de la ocurrencia del error. Este método requiere el establecimiento de puntos de recuperación, que son instantes durante la ejecución de un proceso donde se salvaguarda el estado libre de errores, pudiendo éste posteriormente ser restaurado tras la ocurrencia de un error.
  b) Mediante la **recuperación hacia adelante**, donde la transformación del estado erróneo consiste en encontrar un nuevo estado a partir del cual el sistema pueda seguir funcionando (frecuentemente en modo degradado).
  c) Mediante la **compensación**, donde el estado erróneo contiene suficiente redundancia como para permitir su transformación en un estado libre de errores. Un ejemplo de uso de este método son los códigos correctores de errores que, mediante redundancia en la información transferida o almacenada, permiten corregir determinados errores. Por su relevancia para esta tesis, en el apartado 2.9 se introducen los conceptos básicos sobre la teoría de codificación de la información, y en el capítulo 4 se presenta una nueva familia de códigos correctores de errores especialmente diseñados para el tratamiento de fallos intermitentes.

La sobrecarga temporal (en tiempo de ejecución) necesaria para el procesamiento de los errores puede ser muy diferente según la técnica de recuperación de errores adoptada. En la recuperación de errores hacia delante o hacia atrás, la sobrecarga temporal es más importante cuando ocurre un error que en ausencia del mismo. En el caso de la recuperación hacia atrás, este tiempo es consumido por el establecimiento de puntos de recuperación, es decir, en preparar al sistema para el procesamiento de los errores. Por el contrario, en la compensación de errores la sobrecarga temporal es la misma, o prácticamente la misma, en presencia o ausencia de errores. Además, la duración de la compensación de un error es mucho menor que la de una recuperación de errores hacia adelante o hacia atrás, debido a la mayor cantidad de redundancia estructural.

El primer paso en el tratamiento de los fallos es el **diagnóstico de errores**, que consiste en la determinación de las causas de los errores en términos de su localización y naturaleza. Posteriormente vienen las acciones destinadas a cumplir el objetivo principal del tratamiento de los fallos: impedir una nueva activación de los mismos, o sea, hacerlos pasivos (este procedimiento se denomina **pasivación** de los fallos). Este objetivo se lleva a cabo eliminando del proceso de ejecución posterior los elementos considerados como defectuosos. Si el sistema no es capaz de seguir dando el servicio anterior, puede tener lugar una **reconfiguración**, que consiste en la modificación de la estructura del sistema para que los componentes no averiados del mismo permitan la entrega de un servicio aceptable, aunque degradado. Una reconfiguración puede implicar la renuncia a algunas tareas, o una reasignación de éstas entre los componentes no averiados.

La pasivación del fallo no será precisa si se estima que el procesamiento del error ha podido eliminar el fallo directamente, o si su probabilidad de reaparición es lo suficientemente pequeña. En caso de no tener que realizarse la pasivación, el fallo se considera como un **fallo blando**. Si se realiza la pasivación, el fallo se considera un **fallo duro**. Las nociones de fallo blando y duro están relacionadas con las de fallo temporal y permanente. Los fallos permanentes siempre son fallos duros. Los fallos temporales pueden ser blandos o duros: cuando son externos (fallos transitorios) son fallos blandos, mientras que los internos (fallos intermitentes) son fallos duros. Efectivamente, la tolerancia a fallos transitorios no precisa del tratamiento de los fallos, ya que en este caso la recuperación del error deberá eliminar los efectos del fallo, que ha desaparecido por sí mismo siempre que no se haya generado un fallo permanente por la propagación del transitorio. Las nociones de fallo blando y duro son útiles por los siguientes motivos:

- Distinguir un fallo temporal de uno permanente es una tarea compleja, ya que un fallo temporal desaparece después de un cierto intervalo de tiempo, normalmente antes de que se haya llevado a cabo el diagnóstico del fallo, y fallos de diferentes clases pueden dar lugar a errores similares. Por lo tanto, la noción de fallo blando y duro incorpora la subjetividad asociada a estas dificultades, incluyendo el hecho de que un fallo puede ser declarado como blando cuando su diagnóstico no ha tenido éxito.

- Por la capacidad de estas nociones de incorporar sutilezas en los modos de acción de algunos fallos transitorios, por ejemplo ¿se puede decir que un fallo dormido resultante de la acción de partículas α (debida a la ionización residual de los encapsulados de los circuitos), o de iones pesados sobre elementos de memoria, es un fallo temporal? Sin embargo, un fallo de este tipo es claramente un fallo blando.

La figura 2.7 muestra un esquema resumido de las técnicas de tolerancia a fallos.

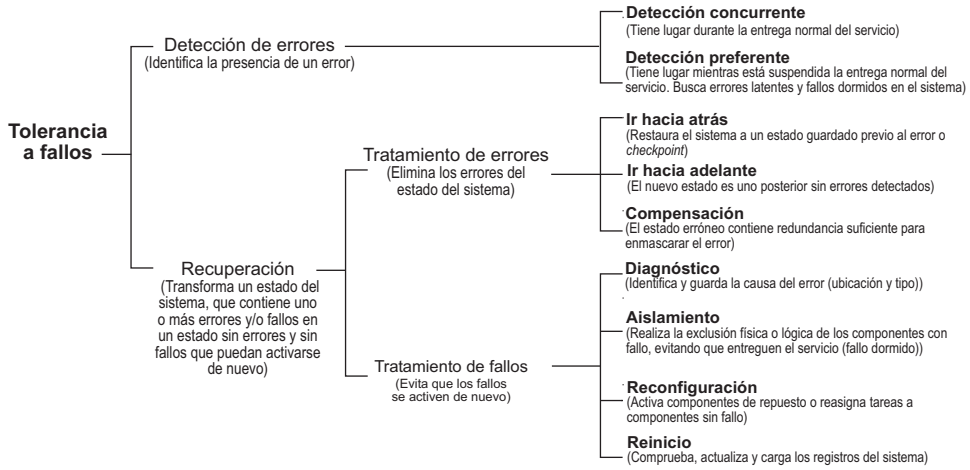

Figura 2.7: Técnicas de tolerancia a fallos.

2.5.2 Eliminación de fallos

La eliminación de fallos está constituida por tres etapas: **verificación**, **diagnóstico** y **corrección**. La verificación consiste en determinar si el sistema satisface unas propiedades, llamadas condiciones de verificación. En caso de que no sea así, se deben realizar las otras dos etapas: diagnosticar el o los fallos que impidieron que se cumpliesen las condiciones de verificación, y posteriormente realizar las correcciones necesarias. Después de la corrección, el proceso debe comenzar de nuevo, con el fin de que la eliminación de fallos no haya tenido consecuencias indeseables. Esta última verificación se denomina de no regresión. Las condiciones de verificación pueden ser de dos formas:

- Condiciones generales, que se aplican a una clase de sistema dado y que son, en consecuencia, independientes (relativamente) de las especificaciones. Por ejemplo, ausencia de bloqueos o conformidad con las reglas de diseño y de realización.
- Condiciones específicas del sistema considerado, deducidas directamente de su especificación.

Las técnicas de verificación se pueden clasificar según si implican o no la activación del sistema. La verificación de un sistema sin su activación real se denomina **verificación estática**, que se puede realizar de las siguientes formas:

- En el propio sistema, en forma de:
  a) **Análisis estático**, por ejemplo inspecciones o ensayos, análisis del flujo de los datos, análisis de complejidad, verificaciones efectuadas por los compiladores, etc.
  b) **Pruebas de exactitud** (aserciones inductivas).
- En un modelo de comportamiento del sistema (basado, por ejemplo, en redes de Petri o autómatas de estados finitos), dando lugar a un análisis del comportamiento.

La verificación de un sistema previamente activado se denomina **verificación dinámica**. En este caso, las entradas suministradas al sistema pueden ser simbólicas, como en el caso de la **ejecución simbólica**, o con un determinado valor como en el caso del test de verificación, también llamado simplemente **test**.

Todos estos enfoques se resumen en la figura 2.8.

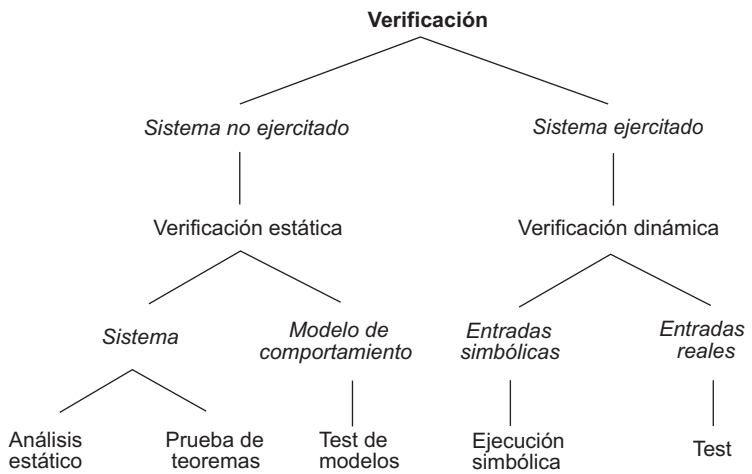

Figura 2.8: Enfoques de verificación.

2.5.3 Predicción de fallos

La predicción de fallos se lleva a cabo realizando una evaluación del comportamiento del sistema respecto a la ocurrencia de los fallos y a su activación. Esta evaluación tiene dos facetas:

- Una **evaluación cualitativa**, destinada en primer lugar a identificar, clasificar y ordenar los modos de avería. Y, en segundo lugar, a identificar las combinaciones de eventos (averías de componentes o condiciones del entorno) que dan lugar a sucesos no deseados.

- Una **evaluación cuantitativa**, destinada a la evaluación, en términos de probabilidades, de algunos de los atributos de la confiabilidad, que pueden por tanto verse como medidas de esta última.

Los métodos y herramientas que permiten la evaluación cualitativa y la cuantitativa son específicos de cada modo de evaluación (por ejemplo, análisis de modos y efectos de las averías para la evaluación cualitativa o cadenas de Markov para la cuantitativa) o sirven para los dos modos en conjunto (por ejemplo, los diagramas de bloques de la fiabilidad y los árboles de fallos).

La definición de las medidas de la confiabilidad precisa, en primer lugar, de las nociones de servicio correcto e incorrecto. **Servicio correcto** es aquél en el que el servicio entregado cumple con la función del sistema. Si el servicio entregado no cumple con la función del sistema, es un **servicio incorrecto**.

Una avería es, por tanto, una transición entre el servicio correcto y el incorrecto. A la transición entre el servicio incorrecto y el correcto se le denomina **restauración**. La cuantificación de la alternancia entre servicio correcto e incorrecto permite definir la fiabilidad y la disponibilidad como medidas de la confiabilidad:

- **Fiabilidad**: medida de la entrega continua de un servicio correcto, o de manera equivalente, del tiempo hasta la avería.

- **Disponibilidad**: medida de la entrega de un servicio correcto, respecto a la alternancia entre servicio correcto y servicio incorrecto.

Los dos principales métodos de evaluación cuantitativa, destinados a la obtención de estimadores cuantificados de las medidas de confiabilidad, son el modelado y el test (de evaluación). Estos métodos son complementarios directamente, en el sentido de que el modelado precisa de datos relativos a los modelos de procesos elementales (procesos

de avería, de mantenimiento, de activación del sistema, etc.), que se pueden obtener mediante test.

Dada la fuerte interacción existente entre la eliminación y la predicción de fallos, ambas se incluyen en el término general **validación**.

La validación de un sistema se puede llevar a cabo de dos maneras:

- **Teórica**, cuando se realiza una predicción de fallos sobre un modelo analítico del sistema.
- **Experimental**, cuando, sobre un modelo de simulación o un prototipo del sistema, se lleva a cabo una predicción o eliminación de fallos.

Cuando se valida un sistema tolerante a fallos, la cobertura de los mecanismos de procesamiento de los errores y de tratamiento de los fallos tiene una influencia primordial. Su evaluación se puede hacer mediante modelado o mediante test. A este tipo de test se le denomina **inyección de fallos**, que se describe con detalle en los apartados 2.7 y 2.8, y que se utiliza en esta tesis para estudiar el comportamiento de los sistemas digitales en presencia de fallos intermitentes.

2.6 Tolerancia a fallos y validación experimental

La validación de los Sistemas Tolerantes a Fallos (STF) precisa de una parte experimental, dada la complejidad en el comportamiento de los sistemas informáticos tolerantes a fallos, debida principalmente a dos motivos [Arlat90, Avizienis04, PGil06]:

- La especialización y novedad de los componentes y las aplicaciones informáticas, tanto en el *hardware* como en el *software*.
  - En cuanto al *hardware*, debido al rápido avance de la tecnología, la utilización de componentes de una determinada "generación" tiene una validez temporal reducida, que hace que las experiencias obtenidas en cuanto a los tipos de fallos y sus consecuencias (patología de los fallos), sirvan de poco en diseños posteriores. Otros aspectos que hay que considerar son la cada vez mayor complejidad de los componentes, y la creación de componentes específicos para aplicaciones empotradas.

  - En el *software* se observa una situación parecida, no solamente en cuanto a los lenguajes de programación usados, sino también a las metodologías de programación e, incluso, a la influencia de los compiladores.
  - Por otra parte, los STF suelen ser utilizados en aplicaciones específicas, con un pequeño número de unidades construidas, lo que dificulta aún más el disponer de datos experimentales acerca de su comportamiento.

- Las incertidumbres relativas a la patología de los fallos: a causa de la complejidad de los sistemas informáticos, existen muchos interrogantes respecto al comportamiento de un sistema en presencia de fallos, sobre todo en el aspecto de cómo cuantificar la influencia de éstos en la confiabilidad.

Como consecuencia de dichos factores, los modelos de STF han evolucionado desde los llamados macroscópicos [Bouricius69], en alusión a un nivel de detalle que sólo tiene en cuenta los procesos de ocurrencia de fallos y reparaciones en los componentes del sistema, hasta los que consideran el comportamiento de los sistemas de tratamiento de fallos, que se denominan microscópicos [Dugan89].

Los modelos macroscópicos se pueden resolver utilizando datos estadísticos de los fabricantes de los circuitos del sistema, teniendo el gran inconveniente de la inexactitud de sus resultados, dado el tratamiento demasiado superficial del proceso de ocurrencia de los fallos. Además, su aplicación para el cálculo de la confiabilidad del *software* del sistema es muy compleja [Kanoun89].

Los modelos microscópicos están basados en la utilización de procesos estocásticos (procesos markovianos y/o semimarkovianos, redes de Petri estocásticas, etc.), e introducen técnicas de resolución analítica y/o de simulación. Se pueden aplicar al conjunto *hardware/software* del STF, consiguiendo resultados más exactos de la confiabilidad. Al tener en cuenta el comportamiento de los sistemas de tratamiento de errores, precisan de datos experimentales para calcular los coeficientes de cobertura y los tiempos de latencia en la detección y recuperación de los errores, parámetros de una importancia fundamental en el cálculo de la confiabilidad de los STF. Esto explica la necesidad de los métodos experimentales, tanto para el cálculo de la confiabilidad (predicción de fallos) como para un mejor conocimiento de la patología de los fallos, que permitirá optimizar los mecanismos de tolerancia a fallos introducidos en el sistema informático (eliminación de fallos).

2.7 Validación experimental e inyección de fallos

La validación experimental puede llevarse a cabo de dos formas diferentes [Arlat90]:

- Mediante experiencias no controladas, observando el comportamiento en fase operativa de uno o varios ejemplares de un sistema informático en presencia de fallos. De este modo se pueden recoger datos sobre coeficientes de cobertura, número de averías y coste temporal de las operaciones de mantenimiento.

- Mediante experiencias controladas, analizando el comportamiento del sistema en presencia de fallos introducidos deliberadamente.

El primer método tiene la ventaja de que es más realista, pues los fallos observados y sus consecuencias son los que ocurren en el funcionamiento real del sistema. Sin embargo presenta una serie de inconvenientes que lo hacen impracticable en la mayoría de los casos:

- La bajísima probabilidad de ocurrencia de los sucesos bajo observación, sobre todo si se trata de un STF. Esto hace que el número de fallos sea muy bajo para un tiempo aceptable, o que el tiempo de observación requerido sea demasiado largo si se desea realizar una estadística con un margen de confianza adecuado.

- El número reducido de STF, por ser sistemas para aplicaciones especiales. Esto hace todavía más difícil la observación en experimentos no controlados.

- La disparidad de las soluciones adoptadas para aumentar la confiabilidad en los STF, que complica la clasificación de estos sistemas para su estudio en presencia de fallos.

Estos inconvenientes hacen que el segundo método, denominado inyección de fallos, sea más adecuado para la validación de STF. La inyección de fallos se define de la siguiente forma [Arlat90]:

> "La inyección de fallos es la técnica de validación de la Confiabilidad de Sistemas Tolerantes a Fallos consistente en la realización de experimentos controlados donde la observación del comportamiento del sistema ante los fallos es inducida explícitamente por la introducción (inyección) voluntaria de fallos en el sistema."

Numerosos estándares, como el IEC 61508 (*Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*, utilizado en la industria [IEC10]) o el ISO 26262 (*Road vehicles - Functional safety*, enfocado a la inocuidad en sistemas informáticos para automoción [ISO11]) consideran la inyección de fallos como una herramienta necesaria en el proceso de certificación de la confiabilidad (especialmente de la inocuidad) de los sistemas informáticos.

La inyección de fallos posibilita la validación de los STF en los siguientes aspectos:

- En el estudio del comportamiento del sistema en presencia de fallos, permitiendo:
  - Confirmar la estructura y calibrar los parámetros (cobertura, tiempos de latencia) de los modelos microscópicos del sistema.
  - Desarrollar, en vistas de los resultados, otros modelos microscópicos más acordes con el comportamiento real del sistema.
- En la validación parcial de los mecanismos de tolerancia a fallos introducidos en el sistema. Se pueden llegar a resultados del tipo: el X% de los errores del tipo Y son detectados y/o recuperados para una carga del tipo Z.

En la figura 2.9 [Arlat90, PGil92, DBench01] se puede observar el proceso de validación de un sistema tolerante a fallos, tanto desde el punto de vista teórico como experimental, mediante inyección de fallos. Como se aprecia en la figura, para efectuar una validación teórica se seguirá el camino (1) del organigrama (I). A partir de unas especificaciones del sistema a desarrollar se construye, en primer lugar, un modelo teórico, normalmente basado en cadenas de Markov o Redes de Petri. A continuación se tiene que caracterizar el modelo, añadiéndole las coberturas y tiempos de latencia. Estos datos se pueden obtener de manera teórica (por hipótesis o comparación con otros modelos) o experimental. Una vez caracterizado el modelo, se procede a su resolución. Los datos que se obtengan de esta resolución se pueden utilizar para desarrollar versiones posteriores del sistema.

Los pasos que hay que seguir para llevar a cabo una validación experimental dependen de si la validación se realiza mediante predicción o eliminación de fallos.

Para realizar la validación mediante predicción de fallos hay que utilizar los organigramas (I) y (II). En este caso, se debe seguir el camino (2) del organigrama (I) para la realización del modelo teórico, si bien éste puede diferir del realizado para una predicción de fallos teórica. Otra diferencia entre ambas maneras de hacer la predicción de fallos es la caracterización del modelo, ya que ahora depende del organigrama (II): hay que realizar un modelo experimental del sistema (que puede ser un prototipo o un modelo de simulación), y a partir de las especificaciones del modelo teórico (flecha (a)) se plantea la inyección; con los resultados obtenidos (coberturas y tiempos de latencia) se finaliza la caracterización experimental del modelo teórico (flecha (b)), tras lo cual se puede resolver, y calcular las medidas de la confiabilidad del sistema, que al igual que en el caso teórico se pueden utilizar para corregir el sistema (realimentación (c)).

Si la validación se efectúa mediante eliminación de fallos, sólo hay que seguir el organigrama (II). En este caso, el planteamiento de la inyección sólo depende del modelo experimental del sistema. Con los valores de las coberturas y tiempos de latencia obtenidos, se pueden tomar las medidas oportunas sobre el sistema a través de la realimentación (d).

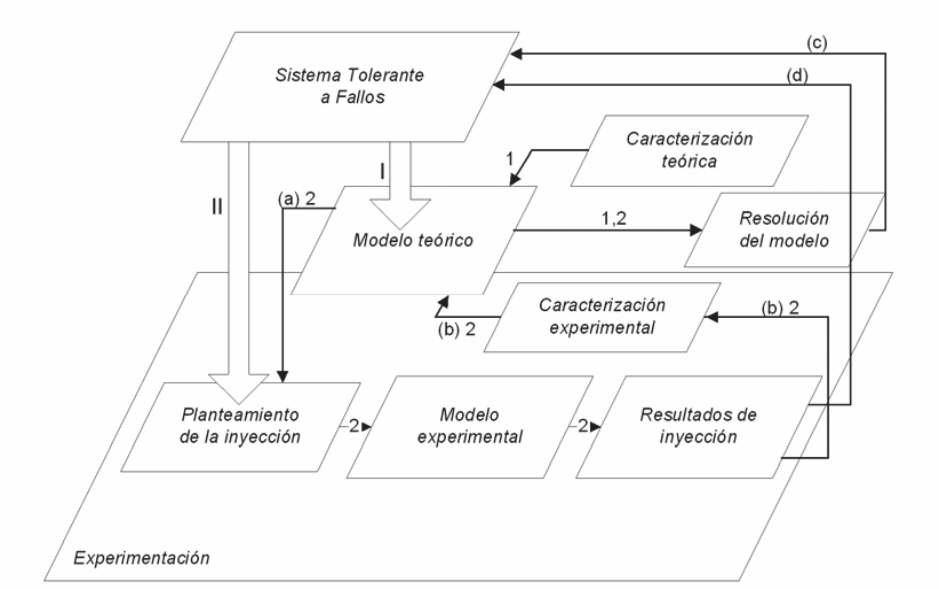

Figura 2.9: Diagrama de bloques del proceso de validación teórica y experimental mediante inyección de fallos.

2.8 Técnicas de inyección de fallos

Vista la utilidad de la inyección de fallos como herramienta de validación de la confiabilidad, y dado que en esta tesis se utiliza para estudiar el comportamiento de los sistemas informáticos en presencia de fallos intermitentes, a continuación se introducen las técnicas de inyección de fallos con mayor aceptación dentro de la comunidad científica. La clasificación más aceptada de las técnicas de inyección de fallos [Hsueh97, Yu01, Benso03] las divide en tres grandes grupos, como se puede ver en la figura 2.10:

- La inyección de fallos física, también llamada implementada mediante *hardware*, o HWIFI (del inglés *HardWare Implemented Fault Injection*) inyecta fallos en el sistema real o en un prototipo del mismo utilizando mecanismos físicos.
- La inyección de fallos *software*, también llamada SWIFI (del inglés *SoftWare Implemented Fault Injection*) utiliza mecanismos *software* para inyectar los fallos. Los fallos pueden inyectarse previamente a la ejecución del programa o aplicarse sobre el sistema en tiempo de ejecución.
- La inyección de fallos basada en modelos, también denominada SBFI (del inglés *Simulation-Based Fault Injection*), se aplica sobre un modelo del sistema, realizado generalmente en un lenguaje de descripción del *hardware* (HDL, del inglés *Hardware Description Language*). Este modelo permite simular su comportamiento mediante alguna herramienta *software*, o emularlo a partir de un prototipo *hardware* implementado sobre una FPGA.

Las técnicas existentes de inyección de fallos se pueden clasificar en una primera aproximación según la fase de diseño del sistema al que se aplican. Así, en las primeras etapas de diseño de un sistema se puede utilizar la inyección de fallos basada en simulación o emulación sobre modelos del sistema, mientras que en etapas más avanzadas se puede hacer inyección de fallos sobre un prototipo del sistema. Dentro de la inyección sobre prototipos se pueden considerar la inyección física de fallos y la inyección de fallos por *software*. En los siguientes apartados se analizan cada una de estas técnicas.

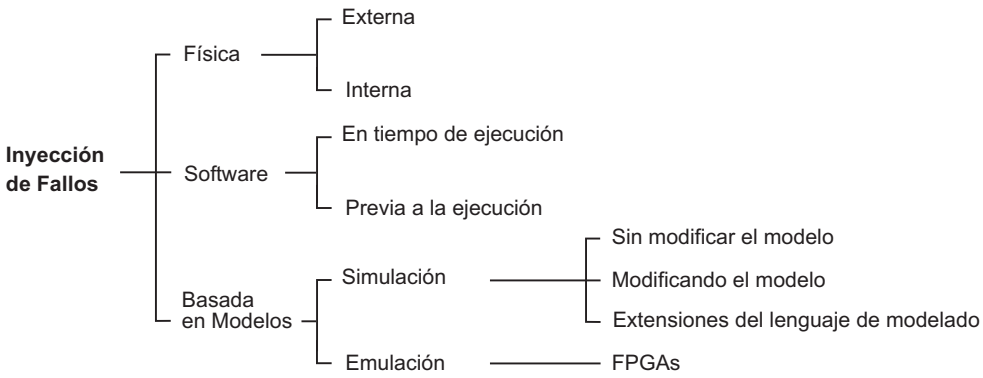

Figura 2.10: Clasificación de las técnicas de inyección de fallos.

2.8.1 Inyección física de fallos

Las técnicas de inyección física de fallos realizan una inyección real del fallo en los terminales de los circuitos, o emulan sus consecuencias (errores) por medio de perturbaciones externas o internas. Hay dos grupos de técnicas de inyección física de fallos: internas y externas. En las técnicas de inyección externa, los fallos se inyectan fuera de los componentes a validar (por ejemplo, la inyección a nivel de pin en los circuitos integrados [Arlat90, Madeira94, PGil97, Birner09, Friesenbichler10], o la inyección por interferencias electromagnéticas [Damm88, Karlsson95]). En las técnicas de inyección internas los fallos se inyectan dentro de dichos componentes (por ejemplo, mediante radiación de iones pesados [Karlsson95], radiación láser [Sampson98, Monnet06, Courbon15], o utilizando mecanismos especiales integrados en el *hardware* del sistema [Folkesson98, Aidemark01, Santos03]).

2.8.2 Inyección de fallos por *software*

El objetivo de la inyección de fallos por *software* es reproducir, a nivel lógico, los errores que se producen tras fallos en el *hardware* [Iyer95], o reproducir errores que ocurren en un sistema debidos a fallos de diseño del *software* (o fallos lógicos) [Madeira00, Duraes03].

La principal característica de este tipo de inyección es que es capaz de inyectar fallos en cualquier unidad funcional accesible mediante el *software* (memoria, registros, periféricos, etc.). Por otro lado, tiene la limitación de algunos registros internos a los que no se tiene acceso [DBench02, Iyer95, DGil04]. A nivel lógico, el fallo puede suponer la corrupción de la información almacenada (al inyectar sobre registros o memoria), la corrupción de las comunicaciones (al inyectar sobre los mensajes) y la corrupción de las unidades funcionales (al inyectar sobre los registros de datos y/o control de las mismas). Además, desde el punto de vista temporal se pueden inyectar fallos de tipo permanente, transitorio e intermitente [Rashid10].

Son numerosas las herramientas basadas en inyección de fallos por *software*. Algunos ejemplos son FIAT [Segall88], EFA [Echte92], DOCTOR [Han95], FERRARI [Kanawati95], FINE [Kao93]/DEFINE [Kao94], Xception [Carreira95, Carreira98], MAFALDA [Rodríguez99], SOFI [Campelo99a, Campelo99b], FlexFi [Benso99], BALLISTA [Kropp98], INERTE [Yuste03], Exhaustif [DaSilva07], simFI [Winter13] o BLFI [Islam14].

2.8.3 Inyección de fallos basada en modelos

El objetivo de la inyección de fallos basada en modelos a través de la simulación o la emulación es comprobar, en una etapa temprana del proceso de diseño, si el comportamiento en presencia de fallos de un sistema en desarrollo coincide con la especificación. El primer requisito es disponer de un modelo del sistema bajo prueba, el cual puede estar desarrollado en diferentes niveles de abstracción. En el caso de la simulación, el modelo es simulado en otro sistema. En el caso de la emulación, como instrumento de simulación del sistema se utiliza un soporte físico.

A la hora de realizar el modelo de un sistema, es frecuente utilizar lenguajes de descripción de *hardware* (HDL, del inglés *Hardware Description Language*), como Verilog [IEEE95] y VHDL (del inglés *Very high speed integrated circuit Hardware Description Language*) [IEEE93]. Mención especial merece, por su actualidad y uso extendido, el estándar SystemC [IEEE05]. Más que un lenguaje de descripción de *hardware*, SystemC es un lenguaje de descripción de sistemas a nivel de comportamiento. Es una biblioteca de código C++ abierto que es usada para desarrollar modelos ejecutables de sistemas *hardware-software* a diferentes niveles de abstracción. Se está convirtiendo en el estándar *de facto* en el codiseño industrial [Bolchini08], donde tanto los componentes *hardware* como los *software* pueden ser descritos utilizando un lenguaje común. Facilita la simulación y la inyección de fallos en sistemas basados en *hardware* y *software*, a diferentes niveles de abstracción [Misera07, Lee09].

La simulación presenta la capacidad de modelar sistemas complejos con un alto grado de fiabilidad, observabilidad y controlabilidad de todos los componentes modelados. La figura 2.10 muestra dos grupos de técnicas de inyección de fallos basadas en modelos. Por un lado, la simulación de esquemas eléctricos y de modelos en lenguajes de algún tipo (principalmente VHDL y Verilog, pero también SystemC, C, C++, ADA, etc.) son técnicas basadas en una simulación por *software* de un modelo mientras que, por otro lado, la inyección de fallos mediante emulación utiliza como instrumento de simulación del sistema un soporte físico, en concreto circuitos lógicos programables (PLD, del inglés *Programmable Logic Devices*) del tipo FPGA (*Field Programmable Gate Array*).

2.8.3.1 Inyección de fallos basada en simulación

En este grupo de técnicas, la inyección de fallos que se realiza para el análisis de la Confiabilidad depende de los diferentes niveles de abstracción definidos. Estos niveles varían según diferentes autores [Walker85, Siewiorek92, Jenn94a, Pradhan96]. Esta tesis utiliza como referencia la división de niveles definida en [DBench02]. En ella se identifican ocho niveles: físico (o de dispositivo), lógico (que se corresponde con las puertas lógicas), de transferencia de registros (RT, del inglés *Register Transfer*), algorítmico, *kernel*, *middleware*, aplicación y operación.

La complejidad de las simulaciones (tanto en espacio como en tiempo) depende de los niveles de abstracción utilizados. Cuanto mayor sea el nivel, se tendrá menor complejidad en las simulaciones a costa de perder detalles. En todo caso, existen problemas que afectan a la inyección de fallos en todos los niveles de abstracción [Pradhan96]. En primer lugar, se debe evitar la "explosión" del tiempo de simulación. Esta explosión puede ocurrir cuando se simulan muchos detalles o cuando se necesita una simulación larga para obtener resultados estadísticos significativos, ya que la probabilidad de los fallos es extremadamente pequeña. Respecto a los fallos, hay que tener en cuenta cuál es el modelo de fallos apropiado para el nivel de abstracción elegido. Para un modelo y un tipo de fallo dados, se debe estudiar cuidadosamente el lugar de la inyección del mismo, ya que puede suceder que el fallo afecte a partes no sensibilizadas por la carga de trabajo, o que los fallos inyectados en un módulo determinado presenten un impacto similar. Por último, el impacto de los fallos en la confiabilidad del sistema también depende de los programas de prueba o carga de trabajo (en inglés *workload*). Por esta razón, el análisis del sistema debe realizarse mientras éste ejecuta cargas representativas, que pueden ser aplicaciones reales, *benchmarks* selectivos o programas sintéticos.

Ejemplos de herramientas de simulación son NEST [Dupuy90], DEPEND [Goswami92, Goswami97], REACT [Clark92], POLIS [Lajolo00], BOND [Baldini03], ISAAC [Akerlund06] o MODIFI [Svenningsson10].

2.8.3.2 Técnicas de inyección de fallos basadas en VHDL

Dado que esta técnica es utilizada comúnmente, en esta sección se introducen algunos conceptos básicos. De hecho, esta es la técnica que se ha utilizado en esta tesis para el estudio de los efectos de los fallos intermitentes, cuyos resultados se exponen en el capítulo 3. Las razones del extendido uso de VHDL se pueden resumir en:

- Es un lenguaje ampliamente utilizado en el diseño digital actual.
- Permite describir el sistema a distintos niveles de abstracción [Aylor90, Dewey92] (puerta, RT, chip, algorítmico, sistema, etc.) gracias a la posibilidad de incluir descripciones estructurales y comportamentales en un único elemento sintáctico.
- Ofrece buenas prestaciones en el modelado de sistemas digitales a alto nivel.
- Tiene una buena capacidad para soportar actividades de prueba [Miczo90].
- Algunos elementos de su semántica facilitan la inyección de fallos.
- Permite una total controlabilidad y observabilidad.

En la figura 2.11 se puede observar un resumen de las técnicas de inyección de fallos sobre modelos en VHDL propuestos en la bibliografía consultada. Se distinguen dos grupos de técnicas, en función de si implican o no la modificación del código fuente del modelo [Benso03].

El primer grupo de técnicas se basa en la utilización de las órdenes del simulador (en inglés *simulator commands*) para modificar en tiempo de simulación el valor y la temporización de las señales y variables del modelo [Ohlsson92, Jenn94b, Baraza00, DGil12a, Gracia14a], sin tener que modificar el código VHDL del mismo.

El segundo grupo se apoya en la modificación del código VHDL, que se puede llevar a cabo de tres maneras: (i) mediante la inserción de componentes perturbadores (del inglés *saboteurs*) en arquitecturas estructurales [Amendola96, Boué98, Folkesson98, DGil03, Baraza08, Gracia13]; (ii) creando mutaciones (en inglés *mutants*) de

componentes ya existentes [Ghosh91, Armstrong92, Gracia01, DGil03, Baraza08]; (iii) existe otro conjunto de técnicas que se basan en la ampliación de los tipos y en la modificación de las funciones del lenguaje VHDL [DeLong96, Sieh97], a las que se denomina *otras técnicas*.

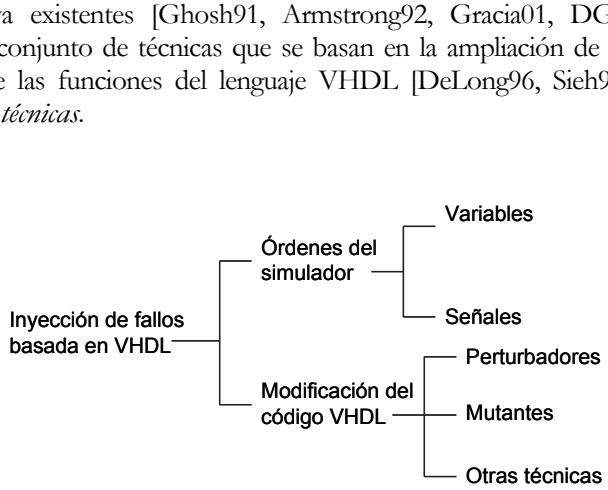

Figura 2.11: Técnicas de inyección de fallos sobre modelos en VHDL [DGil99].

2.8.3.3 Inyección de fallos basada en emulación

Aunque la inyección de fallos basada en emulación con FPGA es conceptualmente muy diferente de las técnicas basadas en simulación, se ha decidido incluirla en este apartado porque para llegar a realizar la inyección de los fallos, se parte de un modelo del sistema descrito en un lenguaje de descripción de *hardware*, siendo esta técnica una especie de evolución de la inyección de fallos mediante simulación basada en este tipo de lenguajes.

Esta técnica también se denomina *emulación de fallos* [Cheng95] y *emulación lógica* [Hwang98], y no hay que confundirla con la inyección de fallos mediante *hardware* (HWIFI) ni con la inyección de fallos mediante *software* (SWIFI). Está basada en el uso de FPGA para realizar prototipos preliminares del sistema en desarrollo, utilizándose habitualmente lenguajes de descripción de *hardware* (principalmente VHDL [IEEE93] o Verilog [IEEE95]) para la especificación del modelo que se sintetizará en la FPGA, así como herramientas de síntesis para poder mapearla y programarla. Con estos prototipos se persigue que puedan representar parcialmente el comportamiento del sistema, sin pretender que cumplan toda su funcionalidad, ni mucho menos que tengan el comportamiento temporal real esperado en el diseño final [Leveugle01].

Por estas razones, la utilidad principal de esta técnica es la localización temprana de fallos de diseño, que reduzcan en gran medida las combinaciones de prueba en etapas posteriores del diseño, bien mediante inyección de fallos o aplicando vectores de prueba, ganándose velocidad a la hora de realizar los experimentos pertinentes. Ejemplos de utilización de esta técnica pueden encontrarse en [Civera01, Alderighi03, deAndrés08].

2.8.3.4 Uso combinado de simulación y emulación

Con el objetivo de aprovechar las ventajas de ambas técnicas (principalmente la velocidad de la emulación y la controlabilidad y la observabilidad de la simulación), al tiempo que minimizar sus inconvenientes, algunos autores proponen el uso combinado de ambas técnicas [Ejlali03, Saiz06, Asadi07, Jeitler09]. Estos métodos proponen la validación de sistemas modelados con un HDL, utilizando emulación en aquellas partes del modelo que sea posible, al ser la técnica más rápida; y usar simulación donde la emulación no sea posible, o sea más conveniente. En general, la parte sintetizable del modelo se podrá emular, mientras que las partes no sintetizables, o aquellas sometidas a rediseño intensivo serán simuladas.

2.9 Códigos correctores de errores

Los códigos detectores y correctores de errores se llevan utilizando desde mediados del siglo XX para mitigar los efectos de distintos tipos de fallos que pueden provocar errores en la información transmitida o almacenada. Su uso es frecuente en almacenamiento de información (CDs y DVDs, memorias DRAM, etc.) y en todo tipo de comunicaciones (telefonía, satélites, etc.). Hoy en día es cada vez más habitual su uso en la estructura interna de los procesadores (registros, memorias cache, buses, etc.).

En este apartado se introducen algunos conceptos sobre teoría de la información, centrándose en el uso de la redundancia con el objetivo de codificar la información para detectar y/o corregir errores.

2.9.1 La transmisión de la información

La transmisión de la información consiste en enviar, a través de un canal, bloques de información que son cadenas de elementos de un alfabeto o conjunto de símbolos. Esta transmisión puede ser tanto espacial (enviar información entre dos puntos distantes en el espacio) como temporal (retener o almacenar la información en un momento determinado para recuperarla posteriormente). Los canales pueden ser muy diversos, y algunos de ellos pueden introducir ruido capaz de alterar la integridad de la información transmitida [Neubauer07].

La codificación de la información transmitida consiste en convertir (codificar) el mensaje original (el bloque de información a transferir) en un mensaje distinto, que será traducido (decodificado) por el destinatario. Aunque se puede codificar la información con distintos objetivos (ocultarla de observadores no autorizados, por ejemplo), en este apartado se consideran los códigos correctores de errores (ECCs, del inglés *Error Correction Codes*), ideados para mejorar la confiabilidad de la transmisión de información a través de canales con ruido.

2.9.2 Códigos correctores de errores: tipos

Hay dos tipos de códigos correctores de errores: los códigos de bloque y los códigos convolucionales [Lin04]. En los códigos de bloque, la codificación de un mensaje (también denominado palabra) de k símbolos depende exclusivamente del propio mensaje. Por tanto, la codificación de un mensaje es siempre la misma, el codificador no tiene memoria y se puede implementar mediante un circuito combinacional.

La codificación y la decodificación de los códigos convolucionales depende del mensaje a enviar/recibir y, además, de los m mensajes anteriores. Se dice que el codificador tiene memoria de orden m. Por esta causa, se debe implementar utilizando circuitos secuenciales.

Los códigos de bloque, a su vez, pueden clasificarse como códigos lineales y no lineales. Un código es lineal si la combinación lineal[1] de dos palabras cualquiera del código es también una palabra del código. Como se verá posteriormente con más detalle,

[1] Dados dos vectores **u** y **v**, y dos escalares a y b, el vector $a\mathbf{u}+b\mathbf{v}$ se dice que es una combinación lineal de **u** y **v**.

esto implica que los códigos lineales de bloque se pueden describir utilizando una matriz generadora, lo que simplifica el proceso de codificación.

Una cuestión importante es el alfabeto (o alfabetos, ya que pueden ser distintos) de entrada y de salida. Los códigos más frecuentes en computación son los códigos binarios, es decir, los códigos que tienen, tanto de entrada como de salida, el alfabeto formado por los símbolos 0 y 1.

Esta tesis se centra en la transmisión de información que se produce en un procesador. Por tanto, el alfabeto (tanto el del mensaje original como el del codificado) es el binario; la información se transmite a través de buses y líneas de interconexión, o bien se almacena en registros y memorias; además, como se detallará en el capítulo 3, tanto los buses como los elementos de almacenamiento son susceptibles a la ocurrencia de fallos que pueden provocar errores en la información transmitida. En el siguiente apartado se introducen con más detalle los códigos de bloque lineales binarios. Por sus características, la mayoría de los códigos prácticos pertenecen a este grupo [Fujiwara06]. Aunque más adelante se presenta algún ejemplo de código no lineal, su utilización dentro de los procesadores no es habitual. Igualmente, los códigos convolucionales requieren de elementos de almacenamiento, lo que ralentiza el proceso de codificación y decodificación (degradando las prestaciones) y aumenta el riesgo de nuevos fallos.

2.9.3 Códigos de bloque lineales binarios

Un código de bloque lineal binario $\mathbb{B}(n, k)$ codifica una palabra de k bits, denominada palabra de datos, en una palabra de n bits, denominada palabra codificada. La palabra de datos $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ es un vector de k bits que representa la información original. Un código $\mathbb{B}(n, k)$ que codifique esta información puede definirse por su matriz generadora $\mathbf{G}$, construida por un conjunto de k vectores de n bits linealmente independientes[2]. Esta matriz define el proceso de codificación, añadiendo la redundancia requerida para la corrección de los errores inducidos por el canal de transmisión. La **redundancia** es la relación de los bits de código añadidos $(n - k)$ respecto a los bits de datos originales (k). La palabra codificada $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ es un vector de n bits obtenido por la regla de codificación:

$$\mathbf{b} = \mathbf{u} \cdot \mathbf{G}.$$

[2] Si un vector es combinación lineal de otros, entonces todos los vectores son linealmente dependientes. Por tanto, varios vectores son linealmente independientes si no existe ninguna combinación lineal entre ellos que genere un vector del conjunto.

Una vez transmitida por el canal la palabra codificada, la palabra recibida $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ es un vector de n bits. El vector de error $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ representa el error inducido por el canal. Si hay un error en el bit i-ésimo, $e_i = 1$; en caso contrario, $e_i = 0$. Por tanto, se puede aplicar la siguiente expresión:

$$\mathbf{r} = \mathbf{b} \oplus \mathbf{e}.$$

Aunque la generación e implementación de códigos puede realizarse eficientemente, el problema de la decodificación puede ser difícil de resolver para algunos códigos [Neubauer07]. Por tanto, la complejidad de la decodificación es una cuestión fundamental. Aunque hay distintos métodos de decodificación, el utilizado más frecuentemente en los códigos de bloque lineales es la decodificación mediante síndrome, fácilmente implementable y que permite una rápida decodificación.

La matriz de paridad $\mathbf{H}$ contiene la información necesaria para realizar la decodificación mediante síndrome. Esta matriz se obtiene a partir de la matriz generadora, y también puede definir el código por sí misma. $\mathbf{G}$ y $\mathbf{H}$ son ortogonales, es decir, $\mathbf{H} \cdot \mathbf{G}^T = \mathbf{0}_{n-k,k}$. Así pues, considerando que el sistema de ecuaciones de paridad se obtiene por la expresión $\mathbf{H} \cdot \mathbf{r}^T = \mathbf{0} \leftrightarrow \mathbf{r} \in \mathbb{B}(n, k)$, el síndrome se define como $\mathbf{s}^T = \mathbf{H} \cdot \mathbf{r}^T$, y depende exclusivamente del vector de error $\mathbf{e}$:

$$\mathbf{s}^T = \mathbf{H} \cdot \mathbf{r}^T = \mathbf{H} \cdot (\mathbf{b} \oplus \mathbf{e})^T = \mathbf{H} \cdot \mathbf{b}^T \oplus \mathbf{H} \cdot \mathbf{e}^T = \mathbf{H} \cdot \mathbf{e}^T$$

Debe haber un síndrome diferente para cada vector de error que se desee corregir. Si el síndrome es un vector cero, se asume que el vector de error también es cero. Por tanto, se da por correcta la palabra recibida. Todas las columnas de $\mathbf{H}$ tienen que ser distintas de cero, para asegurar que un error de un bit no produce un síndrome cero. Hay que considerar que errores de varios bits también podrían producir un síndrome cero, si no son tenidos en cuenta. Por ello, a la hora de diseñar un código es importante que el conjunto de errores a corregir coincida con los que razonablemente se puedan esperar durante su utilización, para minimizar el riesgo de avería.

La decodificación mediante síndrome se lleva a cabo utilizando una tabla de búsqueda que relaciona cada síndrome con el vector de error estimado $\hat{\mathbf{e}}$, que puede contener uno o varios bits erróneos. Haciendo la OR exclusiva entre el vector de error estimado y el vector recibido, se obtiene la palabra decodificada $\hat{\mathbf{b}} = \mathbf{r} \oplus \hat{\mathbf{e}}$. Si el síndrome es distinto de cero, pero dicho síndrome no tiene entrada en la tabla de búsqueda, el error o errores se detectan pero no pueden corregirse.

El esquema del proceso de codificación, transmisión y decodificación mediante síndrome puede observarse en la figura 2.12.

A partir de **G** o de **H** se pueden implementar fácilmente las operaciones de codificación y decodificación. A continuación se presenta un ejemplo de implementación con un código corrector de errores simples, el código Hamming.

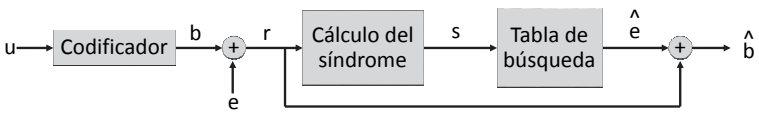

Figura 2.12: Proceso de codificación, transmisión y decodificación.

2.9.3.1 Ejemplo: códigos de Hamming

Los códigos SEC (del inglés *Single Error Correction codes*) son códigos capaces de corregir errores de un bit. Para corregir errores de un bit es necesario que un código tenga distancia de Hamming igual o superior a tres. La **distancia de Hamming** entre dos palabras es el número de bits que son distintos, posición a posición (por ejemplo, la distancia entre 0**01**1 y 0**11**0 es dos, con los bits distintos marcados en negrita). La distancia de Hamming de un código es la mínima distancia entre todas las palabras pertenecientes al código. Si esta distancia es igual o superior a tres, una palabra con un error en un bit estará más próxima a una palabra que pertenece al código que a las demás. De esta forma se podrá determinar la palabra correcta y corregir el error producido.

Los códigos de Hamming [Hamming50] son los códigos SEC que tienen la menor redundancia. Se basan en la idea de las paridades solapadas. Cada bit de código se genera calculando la paridad de un subconjunto de bits de datos (distinto para cada bit de código) de forma que todos los bits de datos quedan cubiertos por varios bits de código. En el proceso de decodificación se obtiene el síndrome calculando todas las paridades. El síndrome obtenido indicará si hay error o no, y en qué bit.

Como ejemplo del proceso de generación, diseño de codificadores y decodificadores, y funcionamiento, a continuación se presenta el código Hamming (7, 4). El código original, tal como se presentó en [Hamming50], se obtiene siguiendo una serie de pasos. En primer lugar, se numeran los bits de la palabra codificada del 1 al 7. Ese número se representa en binario, en este caso con 3 bits (del 001 al 111). Los bits cuya

posición representada en binario tengan un solo 1 corresponden a bits de código, y los demás son los bits de datos. Cada bit de código se obtiene calculando la paridad del resto de bits cuya posición representada en binario tiene a 1 el mismo bit que el bit de código. Por ejemplo, el bit de código en la posición 00**1** se obtiene calculando la paridad de los bits de datos situados en las posiciones 01**1**, 10**1** y 11**1**, o el bit de código en la posición 0**1**0 se obtiene con los bits 0**1**1, 1**1**0 y 1**1**1.

Expresado en notación matricial, se representan en columnas cada uno de los valores, de forma que en cada fila quedan los bits del mismo peso. Por tanto, se obtiene esta matriz, que es la matriz de paridad **H**.

$$\mathbf{H} = \begin{bmatrix} 1010101 \\ 0110011 \\ 0001111 \end{bmatrix}$$

A partir de **H** se podría obtener **G** para el proceso de codificación, tal como se ha explicado anteriormente. No obstante, no sería necesario, ya que únicamente con **H** se pueden obtener las fórmulas necesarias para la codificación. En la tabla siguiente puede observarse cómo se ubican los bits de datos en la palabra de código, considerando una palabra de datos **u** de 4 bits, y cómo se obtienen los bits de código a partir de los bits de datos. Como puede observarse, a cada fila le corresponde un bit de código, que se genera obteniendo la paridad de todos los bits de datos que tienen un 1 en dicha fila.

b_0	b_1	b_2	b_3	b_4	b_5	b_6	***Fórmulas***
		u_0		u_1	u_2	u_3	
1	0	1	0	1	0	1	$b_0 = u_0 \oplus u_1 \oplus u_3$
0	1	1	0	0	1	1	$b_1 = u_0 \oplus u_2 \oplus u_3$
0	0	0	1	1	1	1	$b_3 = u_1 \oplus u_2 \oplus u_3$

El proceso de codificación consiste únicamente en calcular los tres bits de código, lo que puede hacerse de forma rápida y sencilla con varias puertas XOR, por ejemplo. La palabra **b** de 7 bits es enviada a través de un canal susceptible de producir errores. El receptor recibe la palabra **r** de 7 bits. Para comprobar si **r** = **b**, hay que decodificarla. El proceso de decodificación comienza calculando la paridad de cada fila para obtener el síndrome. Las fórmulas se obtienen como se muestra en la tabla siguiente.

r_0	r_1	r_2	r_3	r_4	r_5	r_6	*Fórmulas*
		u_0		u_1	u_2	u_3	
1	0	1	0	1	0	1	$s_0 = r_0 \oplus r_2 \oplus r_4 \oplus r_6$
0	1	1	0	0	1	1	$s_1 = r_1 \oplus r_2 \oplus r_5 \oplus r_6$
0	0	0	1	1	1	1	$s_2 = r_3 \oplus r_4 \oplus r_5 \oplus r_6$

A continuación se asocia cada síndrome con un error distinto. En el caso de los códigos de Hamming, si el valor del síndrome coincide con alguna de las columnas de la matriz, indica que el error se ha producido en el bit correspondiente a esa columna. Obviamente, si el síndrome es cero se entiende que la palabra es correcta.

Veamos un ejemplo sencillo de codificación y decodificación. Supongamos la palabra de datos **u** = (0110). Utilizando la tabla y las fórmulas para la codificación, calculamos la palabra codificada.

b_0	b_1	b_2	b_3	b_4	b_5	b_6	*Fórmulas*
		0		**1**	**1**	**0**	
1	0	1	0	1	0	1	$b_0 = \mathbf{0} \oplus \mathbf{1} \oplus \mathbf{0} = \mathbf{1}$
0	1	1	0	0	1	1	$b_1 = \mathbf{0} \oplus \mathbf{1} \oplus \mathbf{0} = \mathbf{1}$
0	0	0	1	1	1	1	$b_3 = \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{0} = \mathbf{0}$

Por tanto, **b** = (**11**0**0**110), con los bits de código resaltados. Supongamos ahora que esa palabra se envía y en el canal se produce un error en un bit (es indiferente que sea de datos o de código), de forma que la palabra recibida sea **r** = (11001**0**0), donde se ha resaltado el bit erróneo. Si ahora se decodifica esta palabra, se obtiene un síndrome distinto de cero, como puede observarse en la tabla.

1	**1**	**0**	**0**	**1**	**0**	**0**	*Fórmulas*
		u_0		u_1	u_2	u_3	
1	0	1	0	1	**0**	1	$s_0 = \mathbf{1} \oplus \mathbf{0} \oplus \mathbf{1} \oplus \mathbf{0} = \mathbf{0}$
0	1	1	0	0	**1**	1	$s_1 = \mathbf{1} \oplus \mathbf{0} \oplus \mathbf{0} \oplus \mathbf{0} = \mathbf{1}$
0	0	0	1	1	**1**	1	$s_2 = \mathbf{0} \oplus \mathbf{1} \oplus \mathbf{0} \oplus \mathbf{0} = \mathbf{1}$

La columna cuyo valor coincida con el síndrome indica la posición del bit erróneo. Corrigiendo ese bit y extrayendo los bits de datos, obtenemos la palabra decodificada (0110), que coincide con la palabra original.

2.10 Resumen y conclusiones

En este capítulo se ha definido la confiabilidad, que es la propiedad de un sistema informático que permite depositar una confianza justificada en el servicio que proporciona. La confiabilidad se debe ver desde el punto de vista de sus atributos, amenazas y medios.

Los atributos de la confiabilidad permiten expresar las propiedades que se esperan de un sistema así como valorar la calidad del servicio entregado. Los atributos de la confiabilidad son disponibilidad, fiabilidad, inocuidad, confidencialidad, integridad y mantenibilidad.

Las amenazas son circunstancias no deseadas que provocan la pérdida de la confiabilidad. Las amenazas de la confiabilidad son fallos, errores y averías. La aparición de fallos en el sistema provoca errores que, a su vez, pueden desencadenar averías, que son comportamientos anómalos que hacen incumplir la función del sistema.

Los medios para conseguir la confiabilidad son los métodos y técnicas que capacitan al sistema para entregar un servicio en el que se pueda confiar, y que permiten al usuario tener confianza en esa capacidad. Los medios son la prevención de fallos, la tolerancia a fallos, la eliminación de fallos y la predicción de fallos. La prevención de fallos se corresponde con las técnicas generales de diseño de sistemas. La tolerancia a fallos consiste en la utilización de técnicas que permitan al sistema cumplir con su función a pesar de la existencia de fallos. La eliminación de fallos intenta, mediante las etapas de verificación, diagnóstico y corrección, reducir la existencia de fallos en el sistema. Por último, la predicción de fallos intenta estimar el número de fallos en un sistema y su gravedad.

La eliminación de fallos está muy ligada a la predicción. Al conjunto de las dos se le denomina validación, que puede ser teórica o experimental en función del modelo de sistema en el que se realiza. La validación teórica se aplica sobre modelos analíticos, y la experimental sobre prototipos o modelos experimentales. La validación experimental permite calcular los valores de parámetros como las latencias y los coeficientes de cobertura de detección y recuperación de errores de una manera más sencilla que con los métodos analíticos. Una de las formas más habituales es la inyección de fallos, introduciendo deliberadamente fallos en un modelo experimental o prototipo del sistema.

En este capítulo también se ha realizado un breve repaso a las técnicas de inyección de fallos. Se ha visto que se pueden aplicar tanto sobre modelos (entre las que se incluyen las de simulación y las de emulación) como sobre prototipos (que pueden estar basadas en *hardware* o en *software*).

En particular, se ha hecho especial énfasis en las técnicas de inyección de fallos mediante simulación de modelos en VHDL, utilizadas para la realización de experimentos y obtención de resultados en el capítulo 3 de la presente tesis.

Finalmente, se ha hecho una breve introducción a los conceptos básicos sobre los códigos correctores de errores. Se ha presentado el problema de la transmisión de información sobre un canal susceptible de provocar errores en la información recibida, y cómo se puede paliar esta situación añadiendo redundancia a la información. Estos conceptos serán especialmente útiles para desarrollar los códigos propuestos en esta tesis como parte de una solución para mitigar los efectos de los fallos intermitentes, y que se presentan en el capítulo 4 de esta tesis.

3

Efectos de los fallos intermitentes

3.1 Introducción

En este capítulo se hace un exhaustivo estudio de las causas y efectos de los fallos intermitentes en los sistemas digitales. En primer lugar se analizan los mecanismos físicos que pueden provocar fallos intermitentes en las tecnologías submicrométricas utilizadas hoy en día, con el objetivo de determinar modelos de fallo que sean representativos de lo que sucede en la realidad, y que se puedan incorporar en modelos de simulación para inyectar fallos.

Una vez establecidos los modelos, éstos se han utilizado para realizar experimentos de inyección de fallos con el fin de estudiar los efectos de los fallos intermitentes en sistemas no tolerantes a fallos. En este caso, se ha utilizado la técnica de inyección de fallos basada en simulación de modelos en VHDL [IEEE93]. Mediante la herramienta VFIT [Baraza02, DGil03, Baraza03] se han inyectado los modelos de fallos intermitentes en los modelos en VHDL de dos computadores: el microcontrolador 8051 [MC8051-01] y el Plasma, un microprocesador segmentado con arquitectura RISC [Plasma01]. El análisis de los efectos observados en los experimentos ha permitido obtener interesantes conclusiones.

El siguiente paso ha sido estudiar las técnicas de tolerancia a fallos más habituales, para ver si son adecuadas para los fallos intermitentes. Nuevamente, se ha usado VFIT para inyectar fallos intermitentes; en este caso, en un microprocesador tolerante a fallos basado en el procesador académico MARK2 [Armstrong89]. El sistema es dúplex, con procesador de repuesto en frío, detección de paridad y temporizador de guardia [DGil99], una arquitectura típica de sistemas tolerantes a fallos no críticos [Siewiorek98].

Finalmente, se presentan las conclusiones obtenidas tras todo este estudio. La principal es que las técnicas de tolerancia a fallos existentes no son todo lo eficaces que deberían ante la aparición de fallos intermitentes, debido a las características específicas que los diferencian de los fallos permanentes y transitorios. Por ello, se ha abierto una línea de investigación con el fin de desarrollar nuevas técnicas de tolerancia a fallos diseñadas para tolerar adecuadamente los fallos intermitentes, que se presentará en el siguiente capítulo.

3.2 Modelos de fallos intermitentes

A la hora de utilizar modelos de fallos, una de las cuestiones fundamentales es su representatividad, es decir, la equivalencia del efecto del modelo de fallo con el del fallo real, en el nivel de abstracción utilizado. En este apartado, en primer lugar, se presentan los distintos niveles de abstracción, para a continuación centrar la atención en los niveles *hardware* (puertas lógicas y transferencia entre registros, principalmente).

La representatividad a nivel de *hardware* de los modelos de fallos permanentes y transitorios ha sido profusamente documentada, y los modelos definidos están bien establecidos por la comunidad científica [Amerasekera97, DGil99, DBench02, Baraza03]. Sin embargo, para los fallos intermitentes esto no ha sido así, y no han sido muchos los trabajos previos a esta tesis realizados con este objetivo en tecnologías submicrométricas. Seguramente, esto es debido a la consideración que se ha hecho de los fallos intermitentes como preludio de fallos permanentes [DBench02, Smolens07], ya que tradicionalmente se han tratado como el inicio de un proceso de desgaste, que puede manifestarse de forma esporádica e intermitente, sin presentar ninguna cadencia concreta de aparición, hasta que el daño se vuelve irreversible. Por este motivo, los modelos de fallo que se proponían eran los mismos que para los fallos permanentes.

Está previsto que los fallos intermitentes, junto con los transitorios, tengan un gran impacto en las nuevas tecnologías submicrométricas. En estas tecnologías pueden aparecen nuevos mecanismos físicos de fallos intermitentes que no desembocan en fallos permanentes. Hay distintas causas que pueden originar estos fallos [Constantinescu02,

Borkar03, Barros04, Kranitis06, McPherson06]. Entre ellas, la reducción de los tamaños de los transistores, que implican una mayor complejidad de los procesos de fabricación, que provoca residuos o variaciones en el proceso; el incremento de la frecuencia de funcionamiento; la necesidad de ajustar el consumo de energía y mecanismos de desgaste.

Los errores inducidos por los fallos intermitentes se manifiestan de forma similar a los transitorios. La principal característica de los fallos intermitentes está en que éstos se activan repetidamente en el mismo sitio, y habitualmente lo hacen en ráfagas [Constantinescu06]. Además, cambiar el componente afectado elimina el fallo intermitente, lo que no sucede con los transitorios, que no pueden ser reparados por ser debidos a causas externas. Por otra parte, los fallos intermitentes se pueden activar o desactivar por cambios en la temperatura, la tensión de alimentación o la frecuencia de trabajo (cambios conocidos como variaciones PVT, del inglés *Process, Voltage and Temperature*) [Constantinescu07].

Además de estudiar los mecanismos físicos, se han utilizado estudios previos, realizados por distintos autores, en los que la observación de sistemas reales permite la obtención de trazas o *logs* de errores. Ello permite deducir modelos de fallo representativos de los errores reales observados para los niveles de abstracción bajo estudio.

Por tanto, una vez presentados los niveles de abstracción, se hace una recopilación de las causas y los mecanismos físicos encontrados en la literatura científica, que sirven como base para proponer una serie de modelos de fallo específicos para fallos intermitentes a nivel de puerta lógica y de transferencia de registros.

3.2.1 Niveles de abstracción

El primer paso a la hora de establecer unos modelos de fallos adecuados es la definición de los niveles de abstracción en los que se puede dividir un sistema informático. Los modelos de fallos que se van a utilizar en este trabajo están basados en la división de niveles definida en [DBench02], mostrados en la figura 3.1. Existen otras clasificaciones, como las presentadas en [Walker85, Siewiorek92, Jenn94a, Pradhan96]. En estos casos, todos los niveles están relacionados e incluso se solapan.

Como es sabido, los fallos que se producen en un nivel pueden propagarse hacia niveles superiores como errores o averías. No es propósito de esta tesis la definición de los distintos niveles ni su justificación. Puede encontrarse más información sobre este tema en [DBench02, Baraza03].

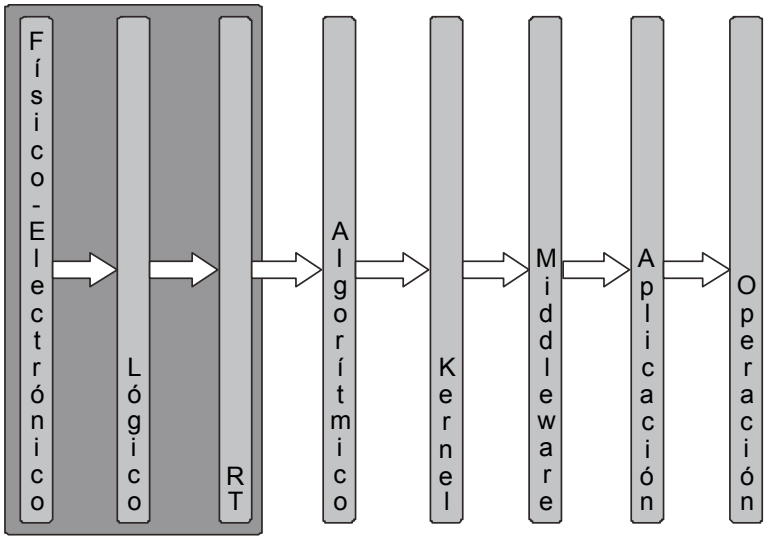

Figura 3.1: Niveles utilizados en el modelado de los fallos *hardware*.

A tenor de lo observado en la figura 3.1 se puede decir que los fallos *hardware* abarcan los tres primeros niveles (físico-electrónico, lógico (puertas lógicas) y RT (*register transfer*, transferencia de registros)). Los modelos de fallos propuestos en esta tesis están orientados a los niveles lógico y RT.

3.2.2 Mecanismos de fallos y su modelado

Para poder establecer modelos de fallo representativos de los fallos intermitentes, se puede actuar en dos sentidos: por una parte, estudiar los mecanismos y las causas que los provocan; y, por otro, utilizar los modelos de fallos permanentes, cambiando su comportamiento temporal, de forma que la duración del fallo no sea indefinida. Para ver si este segundo planteamiento es válido, hay que apoyarse en el primero.

Por ello, en este apartado se estudiarán una serie de mecanismos físicos que pueden causar fallos intermitentes, y que han sido documentados por diferentes autores. En algunos trabajos se han monitorizado sistemas reales, observando los fallos producidos y las averías provocadas. Tras su estudio se determinaban las fuentes de errores y sus manifestaciones más frecuentes. Esta información permite extrapolar sus

efectos a los modelos de fallo adecuados a cada manifestación de los errores, de acuerdo al nivel de abstracción que se quiera considerar, en este caso los niveles lógico y de transferencia de registros (RT).

Para facilitar la consulta por ambos caminos, en primer lugar se presentan en detalle los mecanismos físicos de los fallos intermitentes, indicando para cada uno de ellos los modelos de fallo que se proponen, y posteriormente se hace una recopilación de los modelos de fallo, indicando el mecanismo o los mecanismos en que se fundamentan.

3.2.2.1 Mecanismos físicos

RESIDUOS DE FABRICACIÓN

En [Constantinescu08] se ha documentado la observación de ráfagas de errores de un bit (SBE, del inglés *Single Bit Errors*) en memorias. Tras un análisis de la avería, se constató que residuos de polímero provocaban contactos intermitentes. Este tipo de error, que se puede producir en celdas de almacenamiento (registros, memoria, etc.), causa que el valor de la celda se fije intermitentemente a un valor lógico determinado.

Aunque los errores observados afectaban a un solo bit dentro de una palabra de memoria, es fácil deducir que, con la reducción del tamaño de los transistores, los residuos pueden afectar a más de un bit.

El origen del fallo se basa en un *hardware* inestable, no en una influencia ambiental del exterior. Por tanto, no se trata de un *bit-flip* que aparece repetidamente en el mismo sitio. Más bien se puede considerar como un *stuck-at*, pero con comportamiento temporal intermitente. Así pues, el modelo de fallo que se puede deducir de este mecanismo se ha denominado ***intermittent stuck-at*** (pegado-a intermitente).

SOLDADURAS DEFECTUOSAS

También se han observado ráfagas de SBE en buses de datos [Constantinescu05]. El análisis de las averías reveló que la causa de los errores eran contactos intermitentes en las soldaduras. Es fácil deducir que el mismo problema puede aparecer también en los buses de direcciones y de control. También es fácil deducir, como en el mecanismo anterior, que la reducción de tamaños provoque que las soldaduras defectuosas puedan afectar a más de un bit.

Este tipo de error se puede producir en líneas de interconexión (principalmente buses), y puede causar que la línea conmute entre los dos valores lógicos durante un

tiempo de forma intermitente. El modelo de fallo seleccionado para este caso es el ***intermittent pulse*** (pulso intermitente). También puede ocasionar que se cortocircuite con una línea vecina. En este caso, el modelo es el ***intermittent short*** (cortocircuito intermitente). Finalmente, también puede provocar que la línea se quede desconectada, sin ningún valor lógico. El modelo de fallo deducido para este caso es el ***intermittent open*** (línea abierta intermitente).

DESLAMINACIÓN

La humedad absorbida por un circuito integrado puede expandirse en posteriores procesos a alta temperatura y ocasionar varios fallos. Entre ellos, son característicos la deslaminación entre el chip y el soporte, y entre el soporte y el encapsulado (rompiendo el chip). Sus efectos son la degradación de los cables de conexión, la rotura del encapsulado, el desplazamiento de la metalización y la corrosión. Todo ello, a su vez, puede ocasionar fallos eléctricos debidos al incremento de las corrientes de fuga o circuitos abiertos.

En concreto, la deslaminación del material de la capa de barrera (*barrier layer material delamination*) [McPherson06] produce diferencias de grosor en los conductores, lo que incrementa la resistencia del material y, como resultado, puede llevar al incumplimiento de los requisitos temporales de forma intermitente, dependiendo de las condiciones del entorno (variaciones PVT). Incluso pueden aparecer cortocircuitos o circuitos abiertos intermitentes en una o varias líneas metálicas. Por tanto, los modelos de fallo propuestos son ***intermittent delay*** (aumento intermitente del retardo), ***intermittent short*** e ***intermittent open***.

ELECTROMIGRACIÓN

La electromigración consiste en el arrastre y transporte de los átomos metálicos del conductor por el constante flujo de alta intensidad de electrones que lo atraviesa. A medida que las conexiones metálicas se hacen más delgadas y aumenta la densidad de corriente, se incrementa la importancia de sus efectos [Hawkins00]. Al igual que ocurre con la deslaminación, se produce un incremento de la resistencia del material en las interconexiones, pudiendo llegar a causar circuitos abiertos en las líneas, al generarse huecos (*voids*) [Constantinescu07]. La electromigración también puede provocar cortocircuitos entre conexiones en la misma capa o en capas adyacentes, pudiendo afectar a una o varias señales en un instante de tiempo concreto. Dada la sensibilidad a las variaciones PVT de las tecnologías submicrométricas, estas situaciones pueden no suceder de forma permanente, sino aparecer de forma intermitente dependiendo de las condiciones del entorno. Los modelos de fallo propuestos son ***intermittent delay***, ***intermittent short*** e ***intermittent open***.

DIAFONÍA

La diafonía (en inglés *crosstalk*) aparece cuando hay un acople capacitivo entre líneas adyacentes. Las variaciones de proceso, voltaje o temperatura (variaciones PVT) tienden a amplificar sus efectos.

La diafonía puede producir tres efectos distintos [Moore00]. El primero de ellos tiene lugar cuando una de las líneas se mantiene a valor constante y una línea adyacente conmuta. Si es en la dirección adecuada, y el efecto capacitivo es lo suficientemente grande, se puede provocar un pico o pulso en la línea que debería permanecer a valor constante.

La diafonía también puede causar retardos cuando señales adyacentes conmutan en direcciones contrarias, al producirse un efecto capacitivo entre ellas, conocido como efecto Miller [Sylvester99].

Finalmente, también pueden ocurrir aceleraciones: si dos señales adyacentes conmutan en la misma dirección, el tiempo de conmutación puede ser inferior al esperado.

Los modelos de fallo asignados a estos mecanismos son ***intermittent pulse***, ***intermittent delay*** e ***intermittent speed-up*** (aceleración intermitente) en buses y líneas de interconexión.

DAÑOS EN LA CAPA DE ÓXIDO

Otra causa de fallos intermitentes se debe a daños en la capa de óxido y procesos de desgaste. A esta clase pertenece el mecanismo denominado *soft breakdown*. En este caso, la corriente de fuga del óxido de la puerta fluctúa a lo largo del tiempo, sin inducir daño térmico [Stathis01]. Esto produce fluctuaciones erráticas del voltaje de salida que pueden llevar a valores lógicos indeterminados. Se espera que la sensibilidad a este parámetro aumente conforme se incrementa la escala de integración. La consecuencia es que pueden aparecer tensiones no asociadas a ningún nivel lógico. El modelo de fallo deducido es ***intermittent indetermination***.

Por otro lado, el aumento de la corriente de fuga puede disminuir la velocidad de conmutación de las puertas [Smolens07]. En efecto, la variación de la corriente de fuga hace que se modifiquen las corrientes de carga y descarga (de la carga conectada a la salida del transistor); éstas a su vez afectan a los tiempos de conmutación del transistor. El modelo de fallo deducido en este caso es ***intermittent delay***.

***NEGATIVE BIAS TEMPERATURE INSTABILITY* (NBTI)**

NBTI es un efecto que aparece en los transistores metal-óxido-semiconductor de efecto de campo (MOSFET) de canal p. Este efecto no es nuevo, y ya se producía por envejecimiento y procesos de desgaste, pero el agresivo incremento en la escala de integración ha aumentado los campos eléctricos en las puertas, ha elevado la temperatura de trabajo de los chips y ha modificado el proceso de fabricación (uso de canales p en superficie en lugar de enterrados, y adición de nitrógeno al SiO_2 como dieléctrico de puerta) [Schroder07]. NBTI produce un incremento de la tensión umbral (*threshold voltage*, V_{TH}) y una reducción de la movilidad de los portadores en un transistor, provocados por defectos en la red cristalina (*traps*) generados en la unión Si-SiO_2 ante determinadas situaciones. Aparentemente, los enlaces Si-H, generados tras el proceso de pasivación de hidrógeno para eliminar átomos sueltos de silicio en la unión Si-SiO_2, pueden romperse a elevadas temperaturas, produciendo dichos *traps*. Este efecto suele desaparecer rápidamente cuando desciende la temperatura, aunque se han observado degradaciones permanentes [McPherson06].

El uso extendido de transistores con óxido ultradelgado en las nuevas tecnologías submicrométricas incrementa sustancialmente el campo de óxido vertical, produciendo degradaciones más severas que en tecnologías más antiguas. Además, el incremento de la tensión umbral (V_{TH}) en los transistores MOSFET de canal p puede generar retardos en la circuitería de lógica combinacional y decrementar los márgenes de ruido estático en elementos de almacenamiento [Park09].

Como se ha dicho anteriormente, el efecto aparece y desaparece en función de la evolución temporal de la temperatura, por lo que puede ser causa de fallos intermitentes, tanto en la lógica combinacional como en los elementos de almacenamiento. En el primer caso se pueden generar retardos, por lo que el modelo de fallo deducido es ***intermittent delay***. En el segundo, un menor margen de ruido puede provocar que un nivel lógico sea interpretado erróneamente, por lo que los modelos de fallo deducidos para esta situación son ***intermittent bit-flip*** e ***intermittent indetermination***.

INYECCIÓN DE PORTADORES CALIENTES

La inyección de portadores calientes (en inglés *hot carrier injection*) provoca la degradación de las prestaciones del transistor. Esta degradación de algunos parámetros del transistor, como la tensión umbral, la transconductancia y la corriente del drenador [Hawkins00], se suele atribuir a la presencia de cargas fijas en la capa de óxido a causa de la captura de electrones y huecos en dicha capa, producidos por procesos de dispersión y de ionización por impacto [Rodder95]. Este efecto es conocido desde hace tiempo y puede provocar fallos por envejecimiento, tanto intermitentes como permanentes. Hay factores que han ayudado a reducir los fallos causados por este mecanismo, pero su

efectividad se está viendo contrarrestada por el agresivo incremento en la escala de integración [McPherson06].

El efecto en la circuitería de lógica combinacional es la generación de retardos, de forma similar al mecanismo NBTI, pero en transistores MOSFET de canal n. Por tanto, el modelo de fallo deducido para esta situación es ***intermittent delay***.

ROTURA DE DIELÉCTRICO CON BAJA CONSTANTE DIELÉCTRICA

El incremento en la escala de integración también genera problemas de confiabilidad en las interconexiones intra-chip, especialmente tras la introducción de nuevos materiales, como el cobre o los dieléctricos con baja constante dieléctrica (*low-k dielectrics*), cuya constante dieléctrica (κ) es menor que la del dióxido de silicio. Estos materiales incrementan las prestaciones de los cables, pero degradan la estabilidad térmica y mecánica. Tanto la resistencia mecánica como la resistencia a la rotura eléctrica son mucho menores que las de los dieléctricos basados en silicio [McPherson06], lo que incrementa la probabilidad de fallo.

La rotura de este dieléctrico provoca fluctuaciones, variables en el tiempo, en las corrientes de fuga y en la capacitancia de los cables, lo que a su vez puede provocar retrasos en la respuesta de los circuitos de lógica combinacional. Por tanto, el modelo de fallo deducido para este mecanismo de fallo es ***intermittent delay*** en buses y líneas de interconexión.

Estas roturas también pueden provocar cortocircuitos intermitentes, que eventualmente pueden llevar a una rotura permanente. El modelo de fallo deducido en este caso es ***intermittent short***, también en buses y líneas de interconexión.

VARIACIONES EN EL PROCESO DE FABRICACIÓN

Al reducirse el tamaño de los dispositivos por debajo de la longitud de onda de la luz utilizada en el proceso de fotolitografía durante la fabricación de los chips, el diseño dibujado originalmente puede no reproducirse exactamente en el chip fabricado [McPherson06]. La extensión de las desviaciones aleatorias es, proporcionalmente, cada vez mayor, y sus efectos más severos. Por ejemplo, las desviaciones en el tamaño de las puertas de los transistores y las fluctuaciones en el perfil de dopado provocan variaciones de la tensión umbral de los transistores y de la velocidad de conmutación [Stanisavljevic11]. Los dispositivos más lentos pueden llevar a violaciones de la temporización de los circuitos y, por tanto, a provocar averías. Este tipo de fallos se manifiestan como fallos intermitentes, ya que el circuito funcionará correctamente durante la mayor parte del tiempo.

El modelo de fallo deducido para este mecanismo de fallo es ***intermittent delay***, y se puede producir en transistores MOSFET, presentes tanto en memorias como en circuitos de lógica combinacional y secuencial.

RESUMEN

En este apartado se han recopilado un conjunto de efectos físicos estudiados por diferentes autores [Moore00, Stathis01, Constantinescu02, Borkar03, Constantinescu05, McPherson06, Constantinescu08, Stanisavljevic11], que pueden provocar fallos intermitentes. La tabla 3.1 los resume, indicando los lugares del sistema afectados y los modelos de fallo deducidos. No obstante, esta relación de causas y modelos de fallo no es una lista cerrada. Bien al contrario, la reducción del tamaño de los dispositivos en las tecnologías actuales y futuras pueden causar la aparición de otros mecanismos físicos capaces de provocar fallos intermitentes.

3.2.2.2 Modelos de fallo

En este apartado se recopilan todos los modelos de fallos intermitentes deducidos a partir de los mecanismos físicos descritos en el apartado anterior. Todo ello aparece resumido en la tabla 3.1.

INTERMITTENT STUCK-AT

Este modelo de fallo simula la situación en la que uno o varios bits de un elemento de almacenamiento (registros, memoria, etc.) se quedan, de forma intermitente, fijados a un valor lógico concreto. Se ha deducido tras apreciar, en informes de errores de sistemas reales, la aparición de ráfagas de fallos intermitentes en celdas de almacenamiento, que provocan errores que son detectados, registrados en *logs* y corregidos [Constantinescu08]. Tras el estudio de los elementos afectados, se observó que la causa física fue la existencia de residuos de polímero que provocaban contactos intermitentes.

En cuanto a las duraciones de las ráfagas, la duración de cada activación y desactivación, la separación entre ráfagas y el número de activaciones del fallo dentro de una ráfaga, se ha documentado poco. Posteriormente se ampliará esta cuestión.

Tabla 3.1. Algunos mecanismos de fallos intermitentes y sus modelos [DGil12a]

Causas	Lugares	Mecanismos de fallo	Tipos de fallo	Modelos de fallo
Residuos en celdas	Memoria y registros	Contactos intermitentes	Defecto de fabricación	*Intermittent stuck-at*
Soldaduras defectuosas	Buses	Contactos intermitentes	Defecto de fabricación	*Intermittent pulse* *Intermittent short* *Intermittent open*
Electromigración Deslaminación	Buses Conexiones de E/S	Variación de la resistencia del metal Huecos	Desgaste Temporización	*Intermittent delay* *Intermittent short* *Intermittent open*
Diafonía	Buses Conexiones de E/S	Interferencia electromagnética	Temporización Ruido interno	*Intermittent delay* *Intermittent speed-up* *Intermittent pulse*
Daños en la capa de óxido (*soft breakdown*)	Transistores NMOS en celdas SRAM	Fluctuación de la corriente de fuga	Desgaste Temporización	*Intermittent delay* *Intermittent indetermination*
Negative bias temperature instability	Transistores PMOS en lógica combinacional	Incremento de la tensión umbral del transistor V_{TH} Reducción de la movilidad de los portadores	Desgaste Temporización	*Intermittent delay*
Negative bias temperature instability	Transistores PMOS en celdas SRAM	Disminución del margen de ruido estático	Desgaste	*Intermittent bit-flip* *Intermittent indetermination*
Inyección de portadores calientes	Transistores NMOS en lógica combinacional	Incremento de la tensión umbral del transistor V_{TH}	Desgaste Temporización	*Intermittent delay*
Rotura de la capa del dieléctrico *low-k*	Buses Conexiones de E/S	Fluctuación de la corriente de fuga Variaciones de temperatura Capacitancia degradada	Desgaste Temporización	*Intermittent delay* *Intermittent short*
Desviaciones en el tamaño de la puerta y en el perfil de dopado	Transistores MOSFET en memoria y lógica combinacional	Desviaciones en la tensión umbral del transistor V_{TH} Desviaciones en la velocidad de operación	Variaciones de fabricación	*Intermittent delay*

INTERMITTENT BIT-FLIP

Este modelo de fallo es similar al *bit-flip* utilizado para modelar fallos transitorios en elementos de almacenamiento (memoria y registros) cuando se produce una conmutación en el valor lógico almacenado. Como se ha visto anteriormente, hay mecanismos de fallo que afectan a los transistores con los que están fabricados los elementos de almacenamiento, como el NBTI, que aumentan la tensión umbral de los transistores y reducen los márgenes de tolerancia al ruido estático bajo determinadas condiciones de trabajo [McPherson06]. En esta situación se pueden producir conmutaciones en los valores almacenados y, dado que la evolución temporal de las condiciones de trabajo pueden hacer que esto suceda o no, su comportamiento es intermitente.

INTERMITTENT PULSE

Los pulsos intermitentes simulan la aparición de picos (pulsos) en líneas de interconexión, principalmente los buses del sistema, que modifican de forma intermitente el valor lógico que se transfiere por esa línea. Se deducen tras observar ráfagas de SBE en los buses de datos mediante informes de errores [Constantinescu05]. El análisis de las averías reveló que la causa de los errores eran contactos intermitentes en soldaduras defectuosas. Aunque dichos informes de errores solo registran los errores producidos en los buses de datos, es fácil deducir que el mismo problema puede también aparecer en los buses de direcciones y de control.

Asimismo, se pueden producir pulsos intermitentes en líneas de interconexión provocados por los efectos capacitivos causados por diafonía (*crosstalk*).

INTERMITTENT SHORT

Este modelo de fallo simula la aparición de cortocircuitos intermitentes entre dos líneas de interconexión adyacentes. Tiene diversos orígenes. El primero de ellos es la aparición de ráfagas de SBE en buses de datos [Constantinescu05], provocados por contactos intermitentes en las soldaduras. También se ha deducido tras el estudio de los efectos de la deslaminación del material de la capa de barrera (*barrier layer material delamination*) [McPherson06], que produce diferencias de grosor en los conductores que pueden llegar a causar cortocircuitos.

Igualmente, estudios sobre los efectos de la electromigración revelan que, con el aumento de la escala de integración, pueden llegar a provocar cortocircuitos entre conexiones contiguas (en la misma capa) o situadas en capas adyacentes [McPherson06].

Finalmente, la introducción de nuevos materiales, como los dieléctricos con baja constante dieléctrica (*low-k dielectrics*), ha generado problemas de confiabilidad en las interconexiones intra-chip. La rotura de este dieléctrico puede provocar cortocircuitos intermitentes en buses y líneas de interconexión.

INTERMITTENT OPEN

Este modelo de fallo simula la apertura, desconexión o rotura de una línea de interconexión (circuito abierto), provocando que la electrónica trabaje de forma distinta a la esperada. En este caso no se trataría de una rotura permanente, sino que el efecto se activará y desactivará dependiendo de las condiciones de funcionamiento.

Al igual que en el modelo de fallo anterior, hay distintos mecanismos físicos asociados a este modelo de fallo. Soldaduras defectuosas observadas tras la aparición de ráfagas de SBE en buses de datos [Constantinescu05] pueden provocar esas desconexiones. La deslaminación [McPherson06] produce diferencias de grosor en los conductores y puede llegar a ocasionar circuitos abiertos intermitentes en líneas metálicas. También, a medida que las conexiones metálicas se hacen más delgadas y aumenta la densidad de corriente, se incrementa el problema de la electromigración. Su efecto es el incremento de la resistencia del material, pudiendo llegar a provocar circuitos abiertos en las líneas, al aparecer huecos (*voids*) [Constantinescu07].

INTERMITTENT DELAY

El retardo intermitente modela el incumplimiento de los requisitos temporales en líneas de interconexión, al ralentizarse la conmutación entre niveles lógicos y la transferencia de dichos niveles. Generalmente, esto es debido a un incremento de la resistencia de los conductores. Es decir, el valor lógico que se envía por una línea de interconexión tarda más tiempo del esperado en llegar, lo que en caso de conmutación entre dos valores lógicos distintos puede suponer un funcionamiento inadecuado en los circuitos secuenciales. También modela retardos en el tiempo de respuesta en los circuitos combinacionales, e incluso de los elementos de almacenamiento, generalmente debidos a variaciones en los parámetros de los transistores con los que están hechos dichos circuitos.

Entre los mecanismos que pueden provocar el incremento en la resistencia del material usado en las líneas de interconexión se encuentra la deslaminación del material de la capa de barrera (*barrier layer material delamination*) [McPherson06], que causa diferencias de grosor en los conductores. A medida que las conexiones metálicas se hacen más delgadas y aumenta la densidad de corriente, se incrementa el problema de la electromigración, es decir, el arrastre y transporte de los átomos metálicos del conductor

por el constante flujo de alta intensidad de electrones que lo atraviesa. El efecto es una variación en el grosor del conductor, que puede provocar una mayor resistencia.

Otro mecanismo que puede provocar retardos es la diafonía o *crosstalk*, pero en este caso se producen por efectos capacitivos entre líneas metálicas adyacentes que conmutan en dirección contraria (efecto Miller).

Mecanismos físicos relacionados con el deterioro de la capa de óxido, como el denominado *soft breakdown*, pueden disminuir la velocidad de conmutación de las puertas [Smolens07]. La variación de la corriente de fuga en la salida del transistor afectado hace que se modifiquen las corrientes de carga y descarga (de la carga conectada a la salida del transistor), y éstas a su vez afectan a los tiempos de conmutación del transistor.

Un efecto similar tiene el mecanismo NBTI en los transistores MOSFET de canal p. Este mecanismo aumenta la tensión umbral y reduce la movilidad de los portadores [Schroder07], disminuyendo la velocidad de conmutación cuando se producen determinadas condiciones. Igualmente, la inyección de portadores calientes en transistores MOSFET de canal n produce un incremento de las cargas fijas por procesos de dispersión e ionización por impacto [Rodder95]. En ambos casos, los mecanismos son conocidos desde hace bastante tiempo, pero el agresivo incremento en la escala de integración ha hecho que sus efectos sean cada vez más severos [McPherson06].

Otro mecanismo que provoca retardos intermitentes en las interconexiones intra-chip es la rotura del dieléctrico, especialmente a causa de la introducción de nuevos materiales, como el cobre o los dieléctricos con baja constante dieléctrica (*low-k dielectrics*). Se provocan fluctuaciones, variables en el tiempo, en las corrientes de fuga y en la capacitancia de los cables.

Finalmente, otra causa de retardos, tanto en memorias como en circuitos de lógica combinacional, son las variaciones en el proceso de fabricación, cuyos efectos son cada vez más severos. Por ejemplo, las desviaciones en el tamaño de las puertas de los transistores y las fluctuaciones en el perfil de dopado provocan desviaciones de la tensión umbral de los transistores y en la velocidad de conmutación [Stanisavljevic11]. El comportamiento es intermitente, ya que el circuito funcionará correctamente durante la mayor parte del tiempo.

INTERMITTENT SPEED-UP

Al contrario que en el caso anterior, este modelo de fallo simula la conmutación de una señal a mayor velocidad de la esperada, debido a los efectos capacitivos que aparecen por diafonía (*crosstalk*). Estos efectos aparecen cuando dos señales adyacentes conmutan en la misma dirección.

INTERMITTENT INDETERMINATION

Aparece una indeterminación en una línea de interconexión cuando el valor de tensión en dicha línea no puede asociarse a ningún valor lógico. Uno de los mecanismos físicos que lo causa es el denominado *soft breakdown*, un tipo de defecto o desgaste del óxido que provoca que la corriente de fuga del óxido de la puerta fluctúe a lo largo del tiempo, sin inducir daño térmico [Stathis01]. Esto produce fluctuaciones erráticas del voltaje de salida, pudiendo aparecer tensiones no asociadas a ningún nivel lógico.

El mismo efecto puede producirse ante otros mecanismos de fallo, como el NBTI, donde la reducción de los márgenes de tolerancia al ruido estático puede provocar un voltaje de salida no asociado a ningún nivel lógico.

3.2.2.3 Parametrización de los modelos de fallo

Una vez establecidos diferentes modelos de fallo aplicables a los fallos intermitentes, es preciso determinar cada uno de los parámetros que permiten definir el comportamiento del modelo de fallo, según sea conveniente en cada situación.

MULTIPLICIDAD ESPACIAL

En general, los mecanismos físicos que provocan los fallos intermitentes pueden afectar a más de un bit, bien sea en circuitos de lógica combinacional, en celdas de almacenamiento o en líneas de interconexión. A la hora de configurar los fallos, se puede especificar si el fallo afectará a un solo bit (fallo simple) o a varios al mismo tiempo (fallo múltiple en el espacio). Respecto a los fallos múltiples, los bits afectados pueden ser adyacentes o no. El primer caso simularía bits contiguos de la misma palabra de memoria o líneas vecinas de un bus, mientras que el segundo simularía, por ejemplo, líneas de interconexión que puedan estar en distintas capas del integrado.

CONSIDERACIONES TEMPORALES

Como se ha comentado anteriormente, los fallos intermitentes se manifiestan habitualmente en ráfagas, es decir, cuando se produce la circunstancia física que ocasiona el fallo, éste no se produce una sola vez. En vez de ello, se activa y desactiva repetidas veces hasta que deja de producirse la circunstancia que lo originó, o bien hasta que el fallo se convierte en permanente. Dado que la primera situación está cobrando mayor importancia en las tecnologías submicrométricas, esta tesis pretende estudiar sus consecuencias y evaluar las técnicas de mitigación existentes.

Una primera configuración temporal de los modelos de fallos debería tener en cuenta la separación entre ráfagas y su duración, además del número de ráfagas por simular. Todos estos parámetros se pueden especificar asignando valores fijos, o bien generándolos aleatoriamente según distintas distribuciones de probabilidad.

Simular la aparición de varias ráfagas requerirá una carga de trabajo excesivamente larga, lo que supera el ámbito de esta tesis, por lo que no se va a profundizar en este aspecto. Más importantes, y por ello se incluyen en un apartado específico, son los parámetros que modelan una ráfaga.

MODELADO DE UNA RÁFAGA

En una ráfaga de un fallo intermitente real hay una serie de activaciones y desactivaciones del fallo. Esto es lo que se pretende mostrar en la figura 3.2, donde un nivel alto representa la activación del fallo, mientras que un nivel bajo indica que el fallo no está activado. Tras la última activación, se producirá un tiempo generalmente más largo, que será la separación entre ráfagas.

Como se puede observar, para modelar este comportamiento son precisos tres parámetros: el número de activaciones del fallo dentro de la ráfaga (longitud de la ráfaga o L_{Burst}), el tiempo de actividad (t_A, el tiempo durante el cual está activo el fallo) y el tiempo de inactividad (t_I, el tiempo durante el cual el fallo no está activo). Cada activación o desactivación del fallo puede tener distinta duración.

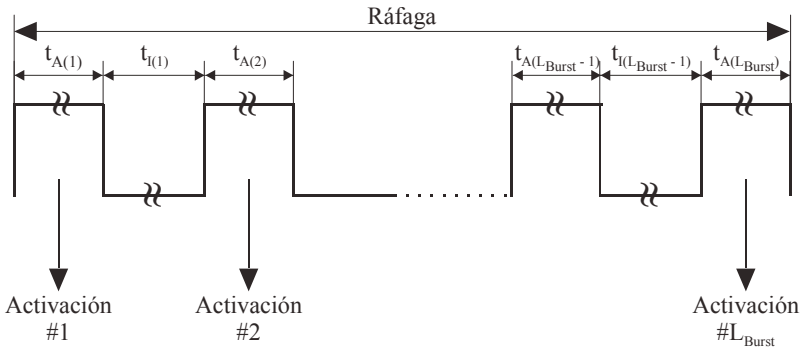

Figura 3.2: Parámetros de una ráfaga [Saiz09].

A la hora de configurar los distintos parámetros de una ráfaga, la longitud de la ráfaga se puede fijar o generar aleatoriamente según alguna distribución de probabilidad.

Lo mismo sucede con los tiempos de actividad e inactividad de cada activación del fallo. La selección del tipo de distribución de probabilidad, o de sus parámetros, no son asuntos triviales. En [Wells08] se hacen comentarios interesantes sobre los parámetros temporales, pero es muy poco lo que hay en la literatura científica al respecto. A continuación se profundiza en la cuestión de las distribuciones de probabilidad.

FUNCIONES DE PROBABILIDAD

Evidentemente, parece poco realista utilizar un valor constante para el número de activaciones, y mucho menos que todas las activaciones tengan los mismos tiempos de actividad e inactividad. Por tanto, parece necesario recurrir a valores aleatorios de cada uno de los parámetros que hay que modelar. Lo importante a la hora de obtener simulaciones bien formuladas es que los valores aleatorios sigan una distribución de probabilidad concreta, y se pueda especificar qué distribución utilizar, y con qué parámetros, para cada una de las variables del modelo de fallo. Cada distribución de probabilidad tendrá unos parámetros distintos. Por ejemplo, si se utiliza una distribución Uniforme, los parámetros serán los valores mínimo y máximo. Si se usa una distribución Gaussiana (o Normal), los parámetros serán la media y la desviación típica.

Una primera aproximación para modelar una ráfaga de un fallo intermitente puede ser usar una distribución Uniforme, tanto para la longitud de la ráfaga como para los tiempos de actividad e inactividad. Es decir, seleccionar aleatoriamente el número de activaciones dentro de un rango que se indique, seleccionar cada uno de los tiempos de actividad aleatoriamente dentro de un rango, y lo mismo para los tiempos de inactividad. Los rangos pueden ser distintos, ya que se trata de variables diferentes. En este caso, todos los valores dentro del rango tienen la misma probabilidad (distribución uniforme). Esta aproximación es válida ya que, por lo general, las condiciones que producen la activación y desactivación de un fallo intermitente no varían excesivamente dentro de la duración de una ráfaga.

Se podría estudiar el uso de otro tipo de distribuciones: Gaussiana (con un valor medio más frecuente y cierta desviación típica), Weibull (ampliamente utilizada para modelar procesos de desgaste o envejecimiento [Siewiorek92]; esta distribución es muy flexible, pues modificando sus parámetros se pueden obtener distintas formas de distribución de probabilidad), etc. No obstante, la función de probabilidad que determine las características de los fallos intermitentes en un sistema real es dependiente de la tecnología de fabricación. Dado que no hay informes o estudios sobre casos reales que determinen una función de probabilidad concreta, para el alcance y objetivos de esta tesis se ha considerado adecuado y suficiente el uso de la distribución Uniforme, dejando el estudio de otras distribuciones de probabilidad como parte del trabajo futuro.

3.3 Efectos de los fallos intermitentes

Aunque los primeros estudios documentados sobre los efectos de los fallos intermitentes usando inyección de fallos basada en simulación datan de hace casi 50 años [Hardie67, Ball69], ha sido en los últimos años cuando han empezado a proliferar trabajos en esta línea, trabajando en distintos niveles de abstracción [Constantinescu05, Gracia08, Constantinescu08, Rashid10, Pan10, DGil12a, Gracia14a].

En este apartado se presenta el análisis realizado por el autor en colaboración con su grupo de investigación. Sobre este análisis se han publicado numerosos artículos [Gracia08, Saiz08, DGil08, Saiz09, Gracia10a, Gracia10b, DGil12a, DGil12b, Gracia13, Gracia14a]. Lo fundamental de estos trabajos se presenta en las secciones 3.3.4 a 3.3.6.

Para estos estudios se ha utilizado VFIT como herramienta de simulación, y que se describe en la sección 3.3.1. Se han utilizado los modelos de fallos definidos anteriormente para inyectar fallos en los modelos en VHDL del microcontrolador 8051 [MC8051-01] y del microprocesador segmentado Plasma [Plasma01], implementados en los niveles de abstracción lógico y de transferencia de registros. El primer procesador tiene una arquitectura CISC (*Complex Instruction Set Computer*, computador con juego de instrucciones complejo), mientras que el segundo tiene una arquitectura RISC (*Reduced Instruction Set Computer*, computador con juego de instrucciones reducido). Ambos han sido ampliamente utilizados en el ámbito científico y se pueden considerar representativos de ambas arquitecturas.

Hay que hacer notar que, aunque los diseños originales de ambos procesadores no estaban pensados para ser fabricados con tecnología submicrométrica (el 8051 porque es anterior a esta tecnología, mientras que el Plasma fue ideado para ser sintetizado en una FPGA), la inyección de fallos se realizaría de forma similar. De hecho, la metodología utilizada se puede generalizar a cualquier sistema más complejo.

3.3.1 La herramienta de inyección de fallos VFIT

Con el fin de automatizar los experimentos de inyección, se ha utilizado la herramienta de simulación **VFIT** *(**V**HDL-Based **F**ault **I**njection **T**ool)* [Baraza02, DGil03]. Esta herramienta automatiza todo el proceso de inyección, desde la configuración del simulador VHDL hasta el análisis y obtención de los resultados. Ha sido diseñada e implementada en torno a un simulador comercial de VHDL. Con VFIT se pueden inyectar una gran cantidad de modelos de fallo, con distintos comportamientos

espaciales (fallos simples o múltiples en distintos puntos del sistema) y temporales (fallos transitorios, permanentes e intermitentes, con su parametrización).

Otro aspecto remarcable de esta herramienta es la posibilidad de analizar automáticamente los resultados obtenidos en los distintos experimentos de inyección. VFIT puede realizar dos tipos de análisis:

- Cuando se estudia el *síndrome de error*, se analizan los efectos de los fallos en el sistema. Se calculan las latencias medias de propagación y el porcentaje de averías producidas. También se obtiene una clasificación de los fallos y errores, así como su incidencia relativa, calculándose las latencias medias de propagación para cada tipo de fallo especificado. Este tipo de análisis es interesante a la hora de determinar las áreas del sistema más sensibles a los fallos, y así elegir los mecanismos de detección y recuperación más apropiados con el fin de mejorar la Confiabilidad del sistema.

- Cuando se *valida un Sistema Tolerante a Fallos*, se realiza un estudio detallado del funcionamiento de los mecanismos de detección y recuperación de fallos del sistema. A partir de este estudio se calculan diferentes parámetros de la Confiabilidad, como latencias y coberturas, generales y para cada mecanismo. Normalmente, un sistema se valida después de habérsele realizado un análisis del síndrome de error y mejorado los mecanismos de detección y recuperación de fallos.

VFIT puede ser ejecutada en una plataforma IBM-PC (o compatible), funcionando bajo el entorno *Windows*™. Ha sido diseñada utilizando como base el simulador comercial *ModelSim* (de Model Technology [Model01]). VFIT es fácilmente transportable y utilizable, además de permitir experimentos de inyección de fallos en sistemas modelados en VHDL en cualquier nivel de abstracción, aunque se ha utilizado principalmente en modelos implementados en los niveles de puerta, registro y chip. VFIT permite utilizar diferentes técnicas de inyección en VHDL: *órdenes del simulador*, *perturbadores* y *mutantes*.

No es propósito de este trabajo explicar en detalle las características de VFIT. Se puede encontrar información más exhaustiva sobre esta herramienta en [Baraza03].

3.3.2 Sistemas bajo prueba y cargas de trabajo

Como se ha comentado anteriormente, se han inyectado fallos en los modelos en VHDL del microcontrolador 8051 [MC8051-01], como ejemplo de la arquitectura CISC, y del microprocesador segmentado Plasma [Plasma01], ejemplo de la arquitectura RISC. Ambos están implementados en los niveles de abstracción lógico y de transferencia de registros. Este capítulo describe brevemente sus características y las cargas de trabajo utilizadas en las simulaciones.

3.3.2.1 Microcontrolador 8051

El 8051 es un microcontrolador muy popular. Pensado para su uso en sistemas empotrados, su núcleo aparece en muchos microcontroladores de diferentes fabricantes. Desde su aparición en el mercado se ha utilizado ampliamente tanto en aplicaciones comerciales como científicas.

El hecho de que se disponga de un modelo estructural en VHDL ha favorecido su utilización como sistema bajo observación en simulaciones usando inyección de fallos [MC8051-01].

En la figura 3.3 se puede observar el esquema de bloques del modelo del 8051 utilizado en los experimentos, así como los lugares donde se han inyectado fallos. La figura 3.4 muestra el diagrama de bloques y la jerarquía de diseño del modelo en VHDL. Los módulos que contienen lógica combinacional se han diseñado usando la descripción comportamental a nivel de transferencia de registros, mientras que los módulos compuestos de sub-módulos interconectados utilizan una descripción estructural. El tipo de descripción se muestra en cada módulo: Struc (estructural) o RTL (comportamental a nivel de transferencia de registros).

3.3.2.2 Microprocesador Plasma

El Plasma es un microprocesador que presenta una arquitectura MIPS™ de 32 bits, con una unidad de ejecución de instrucciones segmentada en cuatro etapas. Es, por tanto, un microprocesador con arquitectura RISC, utilizado principalmente en aplicaciones científicas [Plasma01]. Se diseñó como un modelo en VHDL para ser sintetizado en una FPGA y, como ejemplo de aplicación, actualmente está ejecutando un

servidor web [Plasmaw01]. Tiene controlador de interrupciones, UART, controlador de memoria y controlador de Ethernet. Ejecuta todas las instrucciones MIPS I™ de modo usuario, excepto las de carga y almacenamiento no alineados.

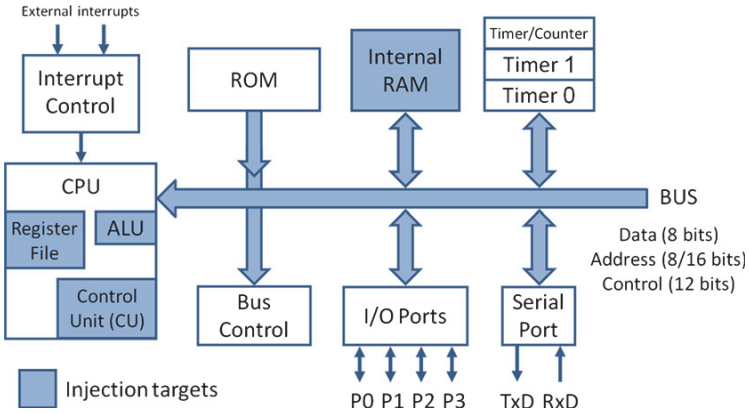

Figura 3.3: Esquema de bloques del modelo del procesador 8051 [DGil08].

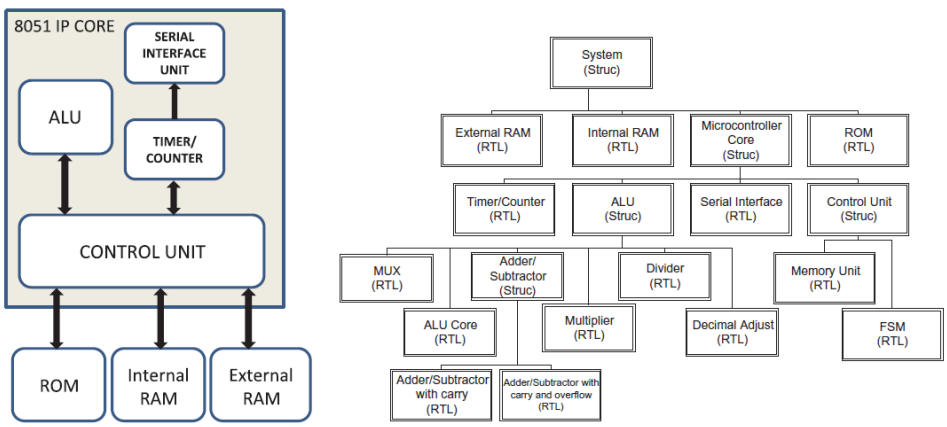

Figura 3.4: Modelo en VHDL del microcontrolador 8051. (a) Diagrama de bloques. (b) Jerarquía de diseño.

Este procesador se está ejecutando a 25 MHz sintetizado en una FPGA de Xilinx©, y también se ha verificado en una FPGA de Altera©. El núcleo del Plasma Versión 3 tiene un puerto serie bidireccional, controlador de interrupciones y temporizador *hardware*. La versión 3.5 añade un controlador de memoria DDR SDRAM, e interfaz para Ethernet y Flash. La figura 3.5 muestra un esquema de bloques del modelo del Plasma utilizado en las simulaciones y los puntos donde se han inyectado fallos. Por último, la figura 3.6 detalla los elementos y las conexiones de la CPU.

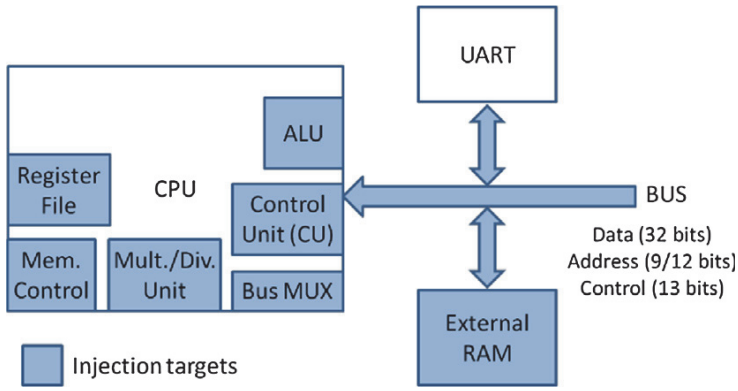

Figura 3.5: Esquema de bloques del modelo del procesador Plasma [DGil12a].

3.3.2.3 Cargas de trabajo utilizadas

Los modelos en VHDL de los procesadores descritos anteriormente permiten simular la ejecución de distintos programas, denominados cargas de trabajo. Cada carga de trabajo tiene unas características particulares en lo referente a los recursos del sistema que utiliza y a su duración. Aunque cada procesador utiliza sus propias cargas de trabajo, y en cada experimento se han utilizado unas cargas concretas, este apartado presenta su descripción y características brevemente. En general, se han utilizado tres cargas distintas:

- Serie aritmética: consiste en sumar los ***n*** primeros números naturales, donde el valor de ***n*** se selecciona antes de iniciar un experimento de inyección. Es la carga cuya ejecución ocupa menos tiempo, por lo que es interesante para reducir los tiempos de simulación. Utiliza muy poca memoria y casi todas las operaciones realizadas sobre ella son de lectura. Por el contrario, hace un uso intensivo de la unidad aritmético-lógica.

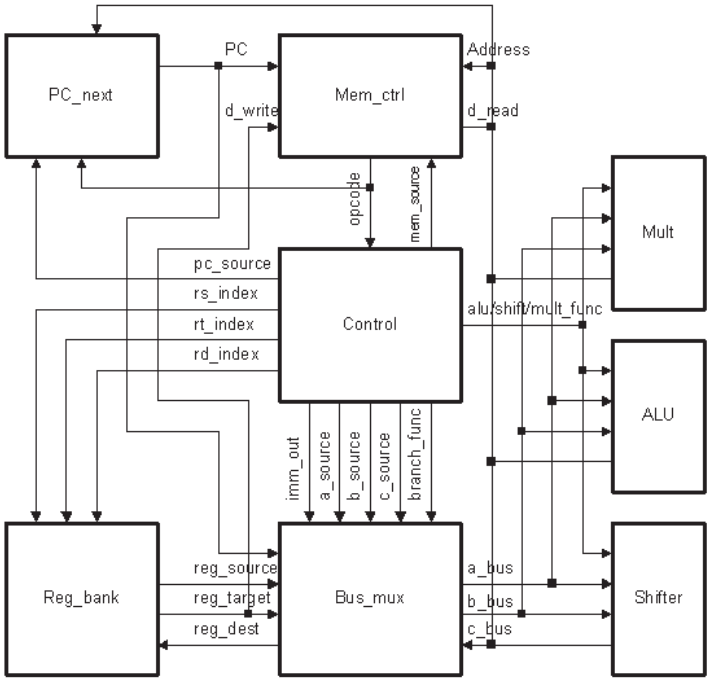

Figura 3.6: Diagrama de bloques y conexiones de la CPU del procesador Plasma [Plasmaw01].

- *Bubblesort*: consiste en ordenar un vector de 10 números con el algoritmo de la burbuja. Esta carga tiene un tiempo de ejecución superior a la anterior. También utiliza muy poca memoria, aunque en este caso se combinan operaciones de lectura y escritura, al realizar la ordenación sobre el mismo vector en memoria.

- Matrix: multiplica dos matrices de tamaño 4×4. Es, de las tres cargas de trabajo, la de mayor tiempo de ejecución, lo que la hace especialmente interesante para el estudio de los fallos intermitentes, donde se deben activar y desactivar repetidamente los fallos. También es la que mayor cantidad de memoria utiliza, combinando operaciones de lectura y escritura.

La obtención de resultados utilizando diferentes cargas de trabajo, con sus distintas características, permitirá obtener conclusiones suficientes para poder generalizar dichos resultados.

3.3.2.4 Puntos de inyección de fallos

Los lugares de inyección de fallos seleccionados para los experimentos son aquellos que pueden verse afectados por los mecanismos de fallos intermitentes tratados en la sección 3.2.2, es decir, elementos de almacenamiento (registros y memorias), líneas de interconexión (buses del sistema, por ejemplo) y circuitos de lógica combinacional (por ejemplo la unidad aritmético-lógica). En concreto, para cada procesador bajo estudio se han seleccionado unos puntos de inyección específicos, que pueden verse resaltados en las figuras 3.3 y 3.5, y que se resumen en esta lista:

- Líneas de interconexión: buses del sistema, que comunican la CPU con la memoria y otros elementos de entrada/salida.
- Elementos de almacenamiento: banco de registros y memoria RAM (interna en el caso del 8051, externa en el caso del Plasma).
- Circuitos de lógica combinacional: unidad aritmético-lógica y unidad de control, en ambos procesadores. Para el Plasma, además, se han inyectado fallos en otros circuitos, como el controlador de memoria, el multiplexor del bus y un circuito aritmético específico para multiplicaciones y divisiones.

3.3.3 Definiciones y terminología

A continuación aparecen una serie de definiciones y abreviaturas que se utilizarán a lo largo del capítulo para expresar distintos parámetros y resultados obtenidos:

Simulación de fallos	Simulación del modelo del sistema bajo estudio en presencia de fallos
Experimento de inyección	Conjunto de simulaciones de fallos configuradas con los mismos parámetros de inyección
$\mathbf{t_{inj}}$	Instante de la inyección (momento de la primera activación del fallo)
$\mathbf{\Delta t_{inj}}$	Duración del fallo (utilizado sólo para fallos transitorios)
$\mathbf{t_A}$	Tiempo de actividad de un fallo intermitente (véase la figura 3.2)
$\mathbf{t_I}$	Tiempo de inactividad de un fallo intermitente (separación entre activaciones; véase la figura 3.2)
$\mathbf{L_{Burst}}$	Longitud de la ráfaga (número de activaciones de un fallo intermitente; véase la figura 3.2)

$\mathbf{N_{Injected}}$	Número de inyecciones realizadas
$\mathbf{N_{Propagated}}$	Número de errores propagados. Se define como el número de fallos inyectados que se propagan a los registros del microprocesador y a la memoria
$\mathbf{N_{Failures}}$	Número de averías. Se define como el número de errores propagados que provocan averías. Una avería se produce cuando el resultado es erróneo al final del tiempo de simulación
$\mathbf{N_{Latent}}$	Número de errores latentes. Un error latente es un error propagado que no provoca avería Se cumple que $N_{Propagated} = N_{Failures} + N_{Latent}$
$\mathbf{N_{Non\_effective}}$	Número de fallos no efectivos. Es el número de fallos inyectados que no han producido ni avería ni error latente Se cumple que $N_{Injected} = N_{Propagated} + N_{Non\_effective}$
$\mathbf{P_{Failures}}$	Porcentaje de averías Se define como $P_{Failures} = (N_{Failures} / N_{Injected}) * 100$
$\mathbf{P_{Latent}}$	Porcentaje de errores latentes Se define como $P_{Latent} = (N_{Latent} / N_{Injected}) * 100$
$\mathbf{P_{Non\_effective}}$	Porcentaje de errores no efectivos Se define como $P_{Non\_effective} = (N_{Non\_effective} / N_{Injected}) * 100$

Una avería se detecta comparando la traza de la ejecución correcta (esto es, en ausencia de fallos, denominada *golden-run* en la bibliografía en inglés) con una traza generada tras la inyección de fallos. Se comparan los contenidos de los registros y/o posiciones de memoria en los que se almacenan los resultados de la carga de trabajo bajo estudio. Si no se ha producido una avería, pero al comparar las trazas hay diferencias en posiciones de memoria o registros distintos a los que almacenan el resultado, se considera que se ha producido un error latente.

3.3.4 Influencia de los parámetros

3.3.4.1 Consideraciones generales

En este apartado se analiza la influencia de algunos de los parámetros de los experimentos de simulación en los efectos de los fallos intermitentes. Se incluyen tanto los parámetros de los modelos de fallos, descritos en el apartado 3.2.2.3, como otros parámetros específicos de los experimentos de simulación.

Uno de los factores que merece especial atención, por ser el rasgo más característico de los fallos intermitentes, es el modelado de una ráfaga. En el apartado 3.2.2.3 se describían los parámetros tiempo de actividad (t_A), tiempo de inactividad (t_I) y longitud de la ráfaga (L_{Burst}). En el apartado 3.3.4.2 se analiza en detalle la influencia de cada uno de estos parámetros.

Por supuesto, no sólo los parámetros de la ráfaga influyen en los efectos causados por los fallos intermitentes. Los resultados observados en las simulaciones dependerán de muchos otros factores. Algunos de ellos son características prefijadas a la hora de realizar las simulaciones, mientras que otros son parámetros que se pueden variar durante las simulaciones.

Por ejemplo, la primera decisión que hay que tomar es el **sistema** que se quiere estudiar. Los resultados obtenidos van a depender no sólo del sistema bajo estudio, sino también del **modelo** del sistema. En estos primeros experimentos se ha utilizado un solo sistema, el 8051, y concretamente una versión del modelo en VHDL desarrollado por Oregano Systems [MC8051-01]. En la sección 3.3.6 se hace un estudio comparativo con los resultados obtenidos para el sistema Plasma, descrito anteriormente. A la hora de generalizar las conclusiones obtenidas, es importante realizar los estudios con modelos fidedignos de distintos sistemas. A mayor exactitud del modelo, más fiables serán los resultados obtenidos.

Los resultados también dependen de la **carga de trabajo** utilizada. En función de su duración, del uso de los recursos del sistema, de las operaciones que se realizan, etc. se pueden obtener distintos resultados. Para generalizar conclusiones se han realizado experimentos con varias cargas de trabajo [Gracia10b], cuyos resultados se incluyen en la sección 3.3.4.3. La principal conclusión a la que se llega es que, aunque en los resultados se observan discrepancias asociables a las características y consumo de recursos del sistema propias de cada carga, las tendencias son similares.

Más específico de la forma en que se realizan las simulaciones, los resultados también dependen de la herramienta de simulación y de la **técnica de inyección** de fallos utilizada. En el caso de esta tesis la herramienta utilizada ha sido VFIT, como se ha comentado anteriormente. Para estos primeros análisis, la técnica de inyección de fallos utilizada ha sido *órdenes del simulador*. Esta técnica permite realizar las inyecciones sin modificar el modelo VHDL original, y es la más sencilla de implementar, aunque permite simular menos modelos de fallo. Para utilizar más modelos de fallo se ha recurrido al uso de *perturbadores* [Gracia10b].

No obstante lo comentado en el párrafo anterior, la influencia de la herramienta de simulación debe ser mínima si está bien diseñada. La precisión de los resultados depende más de la fidelidad y nivel de detalle del modelo utilizado que de la propia herramienta de simulación. Para confirmar la bondad de los resultados obtenidos en

nuestros estudios, en la sección 3.3.7 se incluye una comparación con los resultados obtenidos en otros análisis de distintos autores.

Otro de los parámetros que hay que determinar a la hora de planificar las simulaciones tiene que ver con los números aleatorios (pseudo-aleatorios, en realidad) y la **distribución de probabilidad** utilizada para cada parámetro de la simulación. VFIT utiliza números pseudo-aleatorios donde se puede parametrizar la semilla, de forma que los experimentos sean reproducibles. Para cada parámetro se puede utilizar una función de probabilidad distinta, y la herramienta permite seleccionar entre varias de ellas: Uniforme, Gaussiana, Weibull y Exponencial. El problema surge por la ausencia de valores que se puedan utilizar para los parámetros asociados (ver sección 3.2.2.3).

A continuación, en el apartado 3.3.4.2 se va a estudiar la influencia de los parámetros de una ráfaga de un fallo intermitente, mientras que en el apartado 3.3.4.3 se van a analizar otros parámetros que pueden tener especial incidencia o que pueden ayudar a obtener algunas conclusiones. Para ello se han realizado distintos experimentos de inyección de fallos. La mayoría de los datos presentados en los apartados mencionados se han obtenido a partir de experimentos cuyos principales parámetros se presentan a continuación (donde han sido necesarios experimentos adicionales, se indica expresamente):

Sistema bajo estudio: 8051.

Carga de trabajo: *Bubblesort*; esta carga permite tener un tiempo de ejecución suficiente, sin requerir excesivo tiempo de simulación.

Puntos de inyección (implica experimentos distintos):

- Elementos de almacenamiento (banco de registros y memoria RAM).
- Buses del sistema.
- Circuitería combinacional (ALU y unidad de control).

Técnica de inyección: *Órdenes del simulador*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelos de fallo:

- Para elementos de almacenamiento: *intermittent stuck-at (0, 1)*.
- Para buses: *intermittent pulse*.
- Para lógica combinacional: *intermittent {pulse, open, stuck-at, indetermination}*.
- Al no disponer de especificaciones temporales en el modelo en VHDL del procesador, no se ha podido utilizar el modelo *intermittent delay* con la técnica de inyección usada (*órdenes del simulador*). Esta situación es habitual en modelos de procesadores, ya que los retardos son introducidos en la fase de implementación. Para poder inyectar el modelo *intermittent delay* es necesario utilizar la técnica de inyección basada en *perturbadores*, empleada posteriormente en otros estudios.

Tiempo de actividad (t_A): se ha generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. En estos experimentos, la frecuencia de reloj del procesador es 10 MHz, así que T=100 ns. Por simplicidad, y dada la ausencia de parámetros reales, se ha usado la distribución Uniforme, como se justifica en la sección 3.2.2.3. Cada rango supone un experimento distinto.

Tiempo de inactividad (t_I): se ha generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Longitud de la ráfaga (L_{Burst}): generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: $P_{Non\_effective}$, $P_{Failures}$ y P_{Latent}.

3.3.4.2 Influencia de los parámetros de la ráfaga

En el apartado 3.2.2.3 se hablaba del modelado de una ráfaga de un fallo intermitente, y se describían los parámetros tiempo de actividad (t_A), tiempo de inactividad (t_I) y longitud de la ráfaga (L_{Burst}). A continuación se estudia la influencia de cada uno de estos parámetros utilizando los resultados de cada uno de los experimentos descritos.

TIEMPO DE ACTIVIDAD

Las figuras 3.7 y 3.8 presentan los resultados de las simulaciones realizadas variando el tiempo de actividad, utilizando el rango intermedio de tiempo de inactividad ([0.1T, 1.0T]). La figura 3.7 muestra el porcentaje de averías. Se incluyen datos de las inyecciones en lógica combinacional (*Comb. Logic*), en elementos de almacenamiento (*Storage*) y en buses (*Buses*), y distinguiendo la inyección de fallos simples y múltiples (*Fault multiplicity*). En cada gráfica se pueden observar tres valores para cada una de las combinaciones lugar de inyección/multiplicidad, uno por cada uno de los intervalos del tiempo de actividad.

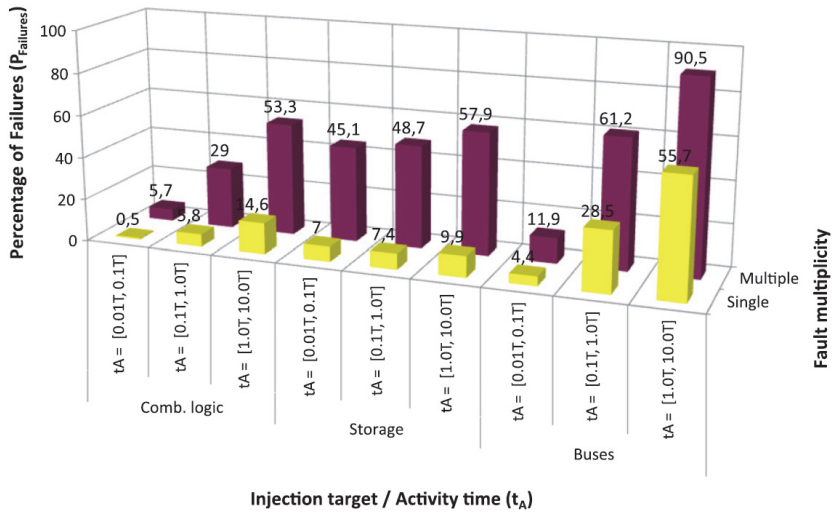

Figura 3.7: Influencia del tiempo de actividad y de la multiplicidad de los fallos en el porcentaje de averías. Microcontrolador 8051 [DGil12b].

Como era previsible, conforme se va aumentando el tiempo de actividad, se observa un mayor porcentaje de averías. Este comportamiento se repite tanto en fallos simples como en múltiples, y para todos los lugares de inyección. En los buses y en la lógica combinacional el incremento es aproximadamente logarítmico (nótese que la escala del tiempo de actividad es logarítmica). En los elementos de almacenamiento se observa un incremento más moderado, debido principalmente a dos causas: (i) los fallos en los registros críticos provocan averías independientemente del tiempo de actividad, y (ii) los fallos en memoria principalmente causan errores latentes, ya que los fallos se inyectan

aleatoriamente y la capacidad total de la memoria es mucho mayor que la parte utilizada por la carga de trabajo.

Relacionado con lo anterior se puede observar que, para los tiempos de actividad más pequeños ([0.01T, 0.1T]), los fallos en elementos de almacenamiento son los que provocan mayores porcentajes de averías, mientras que para duraciones mayores los fallos intermitentes en buses son los más dañinos. También se puede constatar que los porcentajes de avería son mayores para fallos múltiples que para fallos simples, como cabía esperar. Estas observaciones se ampliarán en la sección 3.3.4.3.

La figura 3.8 muestra el porcentaje de errores latentes. Como se puede comprobar, éstos no muestran una tendencia clara respecto del tiempo de actividad. Aumenta en la lógica combinacional y en buses, excepto para el mayor rango de actividad al inyectar fallos múltiples, mientras que es casi constante o ligeramente descendente en elementos de almacenamiento, donde se producen los mayores porcentajes.

Estas discrepancias son debidas, principalmente, al hecho de que los fallos en memoria y registros se propagan instantáneamente (si el fallo cambia el valor del bit afectado, automáticamente se produce un error), mientras que los fallos en los buses o en la lógica combinacional pueden ser enmascarados (por mecanismos de enmascaramiento lógico o temporal intrínsecos de la lógica combinacional [Baraza03]). En estos casos, el incremento del tiempo de actividad reduce la efectividad de los mecanismos de enmascaramiento. Para el mayor rango de tiempos de actividad en buses y en lógica combinacional, el porcentaje de errores latentes provocados por fallos múltiples decrece. Esto probablemente es debido a que fallos, que con un tiempo de actividad menor habrían causado errores latentes, al aumentar el tiempo de actividad y reducirse la efectividad de los mecanismos de enmascaramiento, han acabado siendo averías (reduciendo el porcentaje de errores latentes).

Como se ha comentado anteriormente, también se observa que los errores latentes son mucho más frecuentes en los elementos de almacenamiento que en el resto de lugares. Esto se debe, por una parte, a la ausencia de enmascaramiento en la propagación en este tipo de circuitería; por otra, a la existencia de un elevado porcentaje de celdas de memoria que no son utilizadas por la carga de trabajo seleccionada, pero en las que sí pueden producirse fallos. En la lógica combinacional y en los buses, los fallos intermitentes provocan porcentajes de errores latentes bajos, inferiores incluso a los porcentajes de averías para la mayoría de los casos.

Para finalizar el análisis, es interesante comentar que los porcentajes de errores latentes muestran menores diferencias entre fallos simples y múltiples que los porcentajes de averías. En los elementos de almacenamiento, los fallos simples provocan incluso más errores latentes que los fallos múltiples, ya que la mayoría de los fallos múltiples provocan averías.

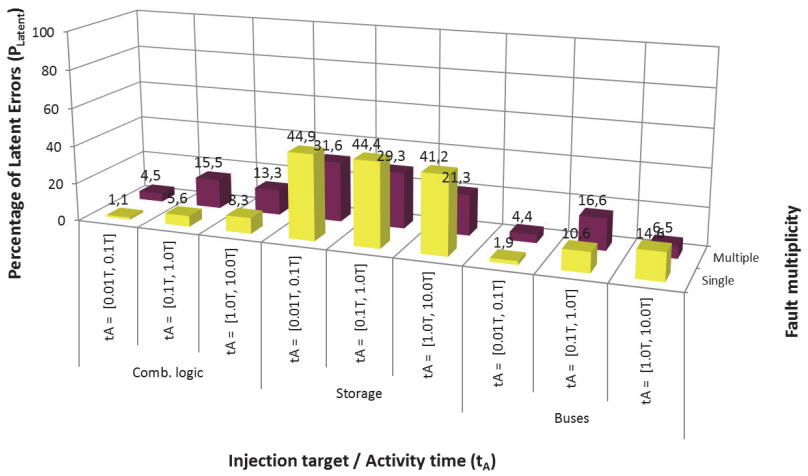

Figura 3.8: Influencia del tiempo de actividad y de la multiplicidad de los fallos en el porcentaje de errores latentes. Microcontrolador 8051 [DGil12b].

TIEMPO DE INACTIVIDAD

Los resultados de variar el tiempo de inactividad (separación entre activaciones) se muestra en la figura 3.9, donde se muestra el porcentaje de averías. En este caso, se ha utilizado el rango intermedio de tiempo de actividad ([0.1T, 1.0T]). Como en el caso anterior, se incluyen datos de las inyecciones en lógica combinacional, en elementos de almacenamiento y en buses, y distinguiendo la inyección de fallos simples y múltiples. En la gráfica se pueden observar tres valores para cada una de las combinaciones lugar de inyección/multiplicidad, para cada uno de los intervalos del tiempo de inactividad.

En cuanto a los resultados, no se aprecian diferencias significativas al variar el tiempo de inactividad, salvo para los elementos de almacenamiento. La separación entre activaciones del fallo dentro de una ráfaga solo provoca variaciones apreciables en el porcentaje de averías ocurridas en los elementos secuenciales del sistema, aunque su comportamiento no es lineal. Este comportamiento se podría explicar, ya que el rango intermedio de tiempo de inactividad ([0.1T, 1.0T]) probablemente cause que la activación del fallo coincida más frecuentemente con el flanco de activación de los elementos de almacenamiento. También se comprueba, en un análisis más detallado, que la separación entre activaciones no ha afectado al número de activaciones en la ráfaga, ya que la carga de trabajo es suficientemente larga como para dar cabida a todas las activaciones, independientemente de su separación.

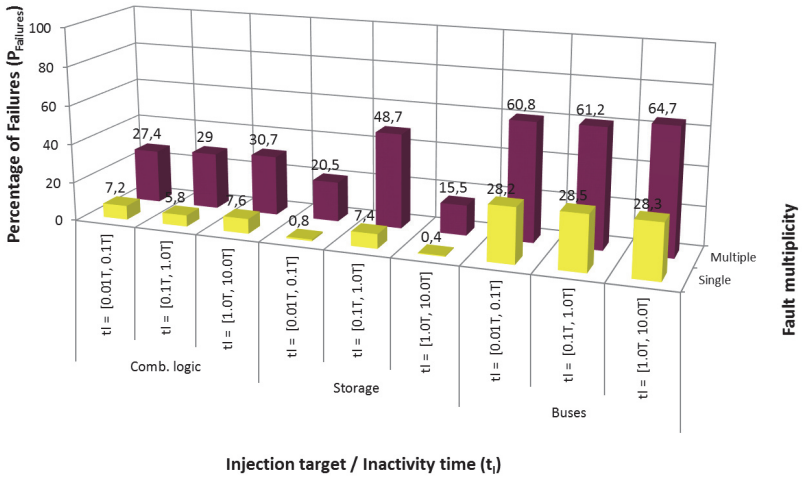

Figura 3.9: Influencia del tiempo de inactividad y de la multiplicidad de los fallos en el porcentaje de averías. Microcontrolador 8051.

No obstante, en sistemas tolerantes a fallos, el tiempo de inactividad puede afectar a los mecanismos de recuperación. Si éstos no son lo suficientemente rápidos, los mecanismos de recuperación pueden ser interrumpidos por la siguiente activación del fallo, provocando una avería en el sistema. Además, en un caso general, el tiempo de inactividad puede influir en el número de activaciones del fallo que afectan al sistema, afectando al porcentaje de averías y errores latentes. A continuación se estudia cómo afecta la longitud de la ráfaga a estos porcentajes.

LONGITUD DE LA RÁFAGA

Para este estudio se han hecho nuevos experimentos fijando el número de activaciones del fallo dentro de la ráfaga a valores entre 1 y 10. La figura 3.10 muestra los porcentajes de averías obtenidos tras inyectar fallos múltiples en la lógica combinacional, los elementos de almacenamiento y los buses del 8051. En este caso, se ha utilizado el rango [0.1T, 1.0T] tanto para el tiempo de actividad como de inactividad.

Se puede observar que, en buses, el porcentaje de averías aumenta asintóticamente hasta un 75% aproximadamente, estabilizándose a partir de una longitud de ráfaga de 9. Para la lógica combinacional la tendencia es similar, creciendo asintóticamente hasta valores alrededor del 35%, estabilizándose a partir de ráfagas de 7 activaciones.

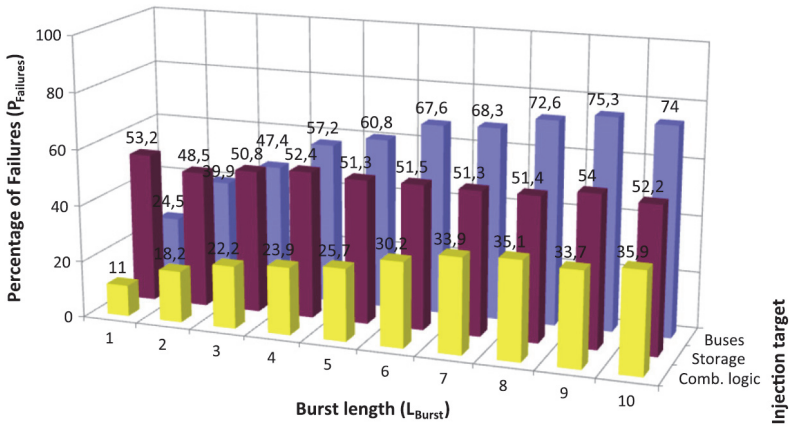

Figura 3.10: Influencia de la longitud de la ráfaga en el porcentaje de averías (fallos múltiples). Microcontrolador 8051 [DGil12b].

Sin embargo, los elementos de almacenamiento muestran un porcentaje de averías casi constante (con valores entre el 48.5% y el 54%), independientemente de la longitud de la ráfaga. Esto es debido a dos factores:

- Los fallos que afectan a registros críticos suelen provocar averías en las primeras activaciones (es decir, incluso con ráfagas cortas).
- Los fallos que afectan a celdas de memoria no accedidas solo causan errores latentes, pero no averías, incluso con ráfagas largas.

3.3.4.3 Influencia de otros parámetros

MULTIPLICIDAD ESPACIAL

Como se ha podido observar en los datos mostrados anteriormente en la figura 3.7, los porcentajes de avería son mayores para fallos múltiples que para fallos simples. Este es el resultado esperado, ya que los fallos múltiples afectan simultáneamente a varios puntos físicos del sistema. Comparando la ratio $P_{Failures\ (multiple)}$ / $P_{Failures\ (single)}$ se observan distintos valores y tendencias para cada uno de los lugares de inyección y rangos de tiempo de actividad:

	Tiempo de activación (t_A)		
	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.0T, 10.0T]
Lógica combinacional	11.4	5.0	3.7
Almacenamiento	6.4	6.6	5.8
Buses	2.7	2.1	1.6

Se puede deducir que:

- La lógica combinacional tiene el mayor valor, obtenido para los tiempos de actividad más pequeños. Sin embargo, al aumentar el tiempo de actividad, el valor de esta proporción desciende rápidamente.
- En los elementos de almacenamiento, por el contrario, se obtienen valores casi constantes y sin una tendencia definida.
- Para los buses se aprecia una tendencia descendente conforme aumenta el tiempo de actividad, de forma similar a la lógica combinacional, pero con valores más bajos, y menor pendiente.

Teniendo en cuenta que, a mayor valor de esta ratio, más afecta la multiplicidad de los fallos, se pueden obtener varias conclusiones. En general, se obtienen mayores ratios para los menores rangos de tiempo de actividad, lo que indica que, en una proporción elevada, fallos con tiempo de actividad pequeño no provocan avería si son simples, pero sí al ser fallos múltiples.

Los buses son los elementos del sistema en los que menos se aprecia la diferencia entre fallos simples y múltiples. Esto es debido a que, si el bus está en uso, incluso un fallo simple puede causar un elevado porcentaje de averías, reduciendo la importancia de la multiplicidad del fallo.

Los elementos de almacenamiento son, en general, los más afectados por la multiplicidad y, además, su efecto es independiente del tiempo de actividad del fallo intermitente. Sin embargo, el tiempo de actividad juega un papel fundamental en la influencia de la multiplicidad en la lógica combinacional. Parece lógico pensar que fallos simples con un tiempo de actividad pequeño pueden, en buena medida, no afectar a la circuitería de la lógica combinacional, mientras que si son múltiples sí pueden afectar en mucha mayor medida. Por el contrario, al aumentar el tiempo de actividad es más fácil que un fallo simple pueda acabar en avería, reduciendo la importancia de la multiplicidad del fallo.

En cuanto a los porcentajes de errores latentes, que se han mostrado anteriormente en la figura 3.8, no se observa una tendencia uniforme entre los distintos lugares de inyección, ni en cuanto al tiempo de actividad del fallo. Sin embargo, si se considera la suma del porcentaje de errores latentes y el porcentaje de averías, es decir, el

porcentaje de errores propagados, se observan las mismas tendencias que para los porcentajes de averías, y para las ratios entre fallos simples y múltiples. La única excepción es que, al considerar los errores propagados, se reduce enormemente el efecto de la multiplicidad en los elementos de almacenamiento. Se puede observar que un elevado porcentaje de fallos simples se manifiestan como errores latentes, mientras que los fallos múltiples se manifiestan mayoritariamente como averías.

LUGAR DE LA INYECCIÓN

A partir de los resultados mostrados anteriormente se puede observar que los fallos intermitentes son bastante más dañinos en los buses que para el resto de lugares de inyección, excepto para los tiempos de actividad más pequeños. Esto es debido a que los buses son un cuello de botella en las fases de búsqueda y ejecución de instrucción del procesador. Los tiempos de actividad pequeños facilitan que el fallo pueda darse en períodos de no uso del bus, de ahí su menor influencia.

Los fallos intermitentes en registros provocan un elevado porcentaje de averías, ya que se usan con mucha frecuencia al ejecutar las instrucciones del procesador. Sin embargo, los fallos intermitentes en memoria se manifiestan principalmente como errores latentes, al no utilizar para la carga de trabajo ejecutada un elevado porcentaje de la memoria.

La lógica combinacional es menos sensible a los fallos intermitentes, aunque su impacto es mayor para valores altos del tiempo de actividad, y para ráfagas largas.

FRECUENCIA DE RELOJ DEL SISTEMA

Aunque la tendencia a aumentar la frecuencia de funcionamiento de los sistemas VLSI se ha moderado, debido a los problemas generados por el consumo eléctrico y el sobrecalentamiento, es frecuente encontrar sistemas preparados para trabajar a distintas frecuencias, dependiendo de la carga de trabajo o de otros factores [Zhan14]. Para comprobar cómo afecta esta variabilidad a los efectos de los fallos intermitentes se han realizado nuevos experimentos de inyección de fallos. Como ejemplo, se incluyen los resultados obtenidos al inyectar fallos intermitentes simples y múltiples en los buses del 8051 modificando la frecuencia de funcionamiento, aunque se han observado tendencias similares en los demás lugares de inyección. En este caso se ha utilizado el rango [10ns, 100ns] tanto para el tiempo de actividad como para el tiempo de inactividad, siendo 100ns la duración de un ciclo considerando la frecuencia original F. La figura 3.11 muestra los porcentajes de averías y de errores latentes. $P_{Failures}$ crece asintóticamente, tanto para fallos simples (donde tiende al 60%) como para fallos múltiples (donde se estabiliza alrededor del 90%).

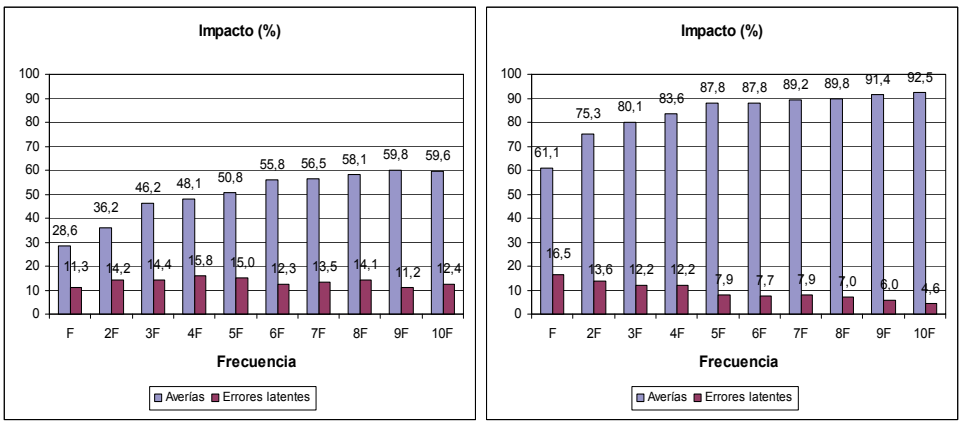

Figura 3.11: Influencia de la frecuencia de reloj en los porcentajes de averías y de errores latentes en los buses del 8051. (a) Fallos simples. (b) Fallos múltiples.

La justificación de la gran influencia de la frecuencia es que a mayores frecuencias la probabilidad de capturar un error en flancos activos de los componentes síncronos aumenta.

Por otra parte, el porcentaje de errores latentes no refleja una dependencia determinada. Mientras para fallos múltiples parece descender asintóticamente, para fallos simples no hay una tendencia clara, manteniéndose en un rango entre el 11% y el 16%.

CARGA DE TRABAJO

Para comprobar cómo afecta la utilización de distintas cargas de trabajo en los efectos de los fallos intermitentes, se han realizado nuevos experimentos de simulación con los siguientes parámetros:

Sistema bajo estudio: 8051.

Cargas de trabajo (implica experimentos distintos):

- Serie aritmética.
- *Bubblesort.*
- Matrix.

Punto de inyección: buses del sistema.

Técnica de inyección: *Perturbadores*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelos de fallo (implica experimentos distintos):

- *Intermittent pulse*.
- *Intermittent short*.
- *Intermittent open*.
- *Intermittent delay*, con un valor de retardo generado mediante una distribución Uniforme en el rango [0.1T, 1.5T].

Tiempo de actividad (t_A): generado mediante una distribución Uniforme en el rango [0.1T, 1.0T].

Tiempo de inactividad (t_I): generado mediante una distribución Uniforme en el rango [0.1T, 1.0T].

Longitud de la ráfaga (L_{Burst}): generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: $P_{Non\_effective}$, $P_{Failures}$ y P_{Latent}.

La figura 3.12 muestra los porcentajes de averías obtenidos. En ella se puede observar que los valores son bastante similares y siguen las mismas tendencias, a excepción de los porcentajes de averías obtenidos con el modelo de fallo *intermittent delay* que son sensiblemente inferiores a los obtenidos con el resto de modelos. Este comportamiento se observa para todas las cargas de trabajo.

La figura 3.13 muestra los porcentajes de errores latentes. Se puede apreciar que, para fallos simples, el modelo de fallo *intermittent delay* presenta menores valores que el resto de modelos. También se puede observar que el modelo de fallo *intermittent short* obtiene mayores porcentajes de errores latentes para la carga de trabajo *bubblesort* que para el resto.

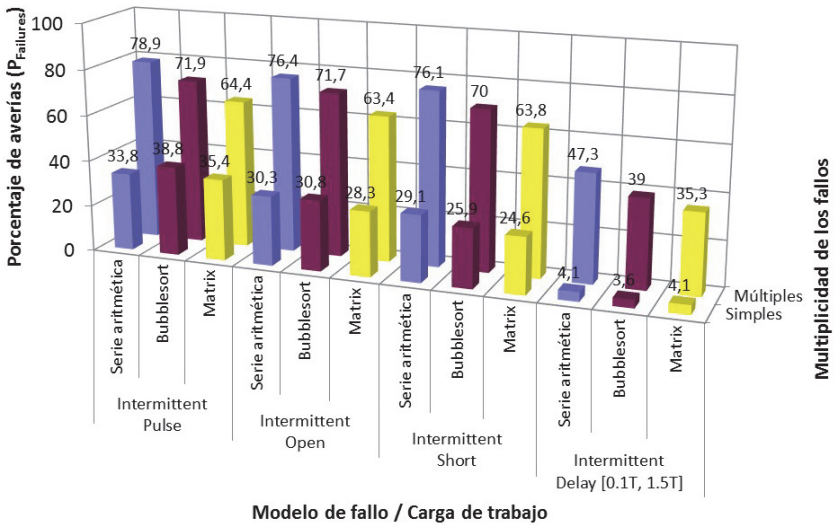

Figura 3.12: Influencia de la carga de trabajo en los porcentajes de averías en los buses del 8051.

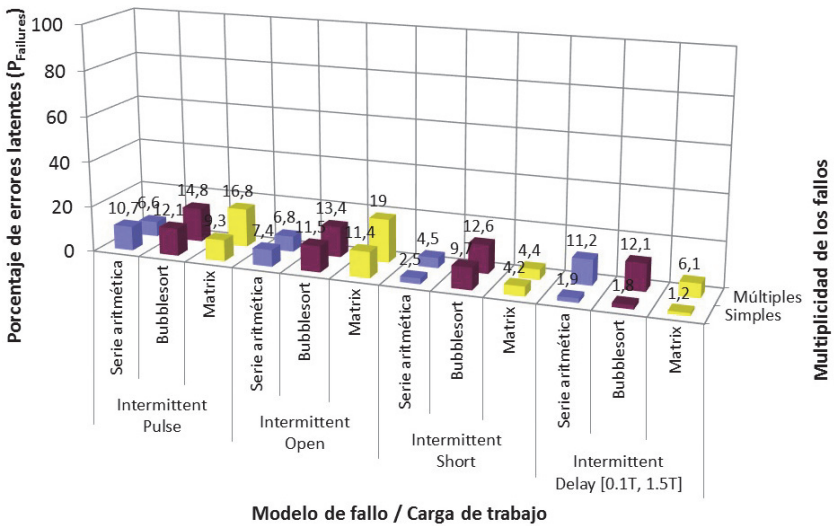

Figura 3.13: Influencia de la carga de trabajo en los porcentajes de errores latentes en los buses del 8051.

3.3.5 Comparación con fallos transitorios y permanentes

En este apartado se describen los experimentos de inyección realizados con el objetivo de comparar los efectos de los fallos clasificados respecto a su persistencia temporal: permanentes, intermitentes y transitorios. Éstos son los principales parámetros de los experimentos realizados:

Sistema bajo estudio: 8051.

Carga de trabajo: *Bubblesort*.

Puntos de inyección (implica experimentos distintos):

- Elementos de almacenamiento (banco de registros y memoria RAM).
- Buses del sistema.
- Circuitería combinacional (ALU y unidad de control).

Técnica de inyección: *Órdenes del simulador*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelos de fallo (implica experimentos distintos por cada tipo de fallo):

- Para fallos transitorios: *bit-flip* en elementos de almacenamiento, *pulse* en buses y *pulse* e *indetermination* en lógica combinacional.
- Para fallos permanentes: *stuck-at (0, 1)*, *open* e *indetermination*.
- Para fallos intermitentes: *intermittent stuck-at (0, 1)* para elementos de almacenamiento, *intermittent pulse* en buses, e *intermittent {pulse, open, stuck-at, indetermination}* para lógica combinacional.

Duración del fallo (Δt_{inj}, en fallos transitorios**)**: generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Tiempo de actividad (t_A, en fallos intermitentes**)**: generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Tiempo de inactividad (t_I, en fallos intermitentes**)**: generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Longitud de la ráfaga (L_{Burst}, en fallos intermitentes**)**: generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: $P_{Failures}$ y P_{Latent}.

La figura 3.14 representa los porcentajes de averías obtenidos en las inyecciones de fallos simples. En la figura se han utilizado tres rangos distintos del tiempo de actividad para los fallos intermitentes, y el rango intermedio [0.1T, 1.0T] para el tiempo de inactividad. Se han utilizado los mismos tres rangos para la duración de los fallos transitorios, salvo para el fallo *bit-flip*. Hay que tener en cuenta que este fallo no perdura en el tiempo (se produce en un instante determinado), por lo que no tiene sentido variar la duración de los fallos transitorios en elementos de almacenamiento.

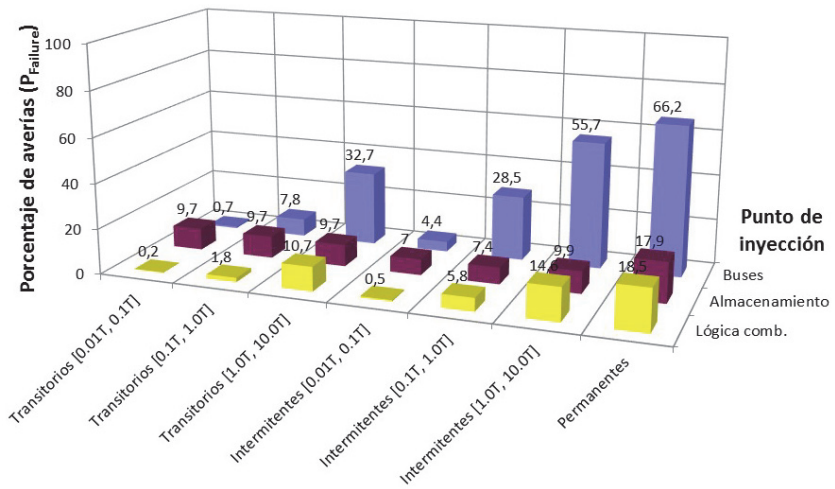

Figura 3.14: Comparación de los efectos de fallos simples transitorios, intermitentes y permanentes en los diferentes puntos de inyección del 8051.

En los resultados se puede constatar que los fallos intermitentes provocan, en buses y lógica combinacional, mayores porcentajes de averías que los fallos transitorios para cada rango de tiempo de actividad. Este comportamiento era previsible, ya que los fallos intermitentes se manifiestan de forma similar a una serie de fallos transitorios, aunque el origen y mecanismos físicos sean distintos. Sin embargo, en los resultados en elementos de almacenamiento se observa un resultado inesperado: los fallos intermitentes tienen menor impacto que los transitorios, incluso para los mayores tiempos de actividad. Tras estudiar con más detalle los modelos de fallo inyectados y la carga de trabajo, se ha podido establecer la causa de este comportamiento anómalo: las características de la carga de trabajo hacen que se produzca una baja tasa de operaciones de sobrescritura en las celdas de memoria afectadas por los fallos transitorios (*bit-flips*). Esto causa que la duración *de facto* de estos fallos sea infinita; es decir, quedan almacenados permanentemente, por lo que el efecto resultante es como si el fallo fuese permanente.

Obviamente, los fallos que mayor porcentaje de averías provocan son los permanentes, dada su duración infinita.

La figura 3.15 muestra los porcentajes de averías obtenidos al inyectar fallos múltiples. Puede constatarse que en los fallos múltiples se repiten las tendencias comentadas anteriormente aunque con mayores valores absolutos, como cabía esperar.

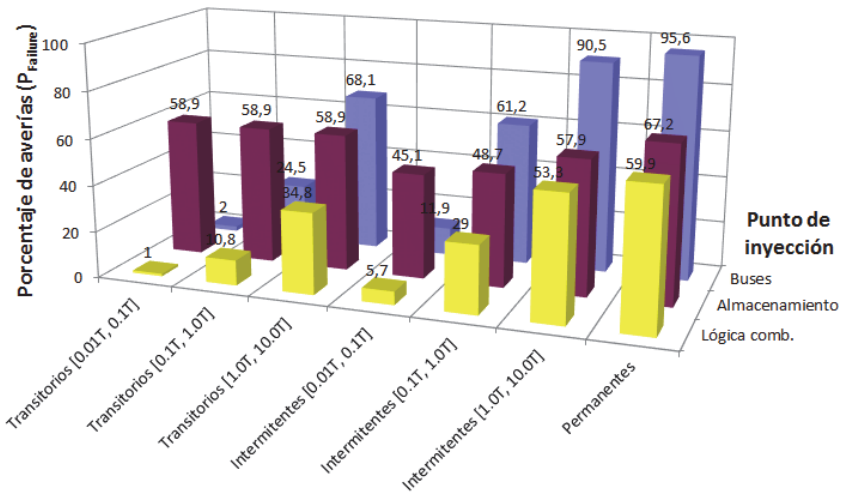

Figura 3.15: Comparación de los efectos de fallos múltiples transitorios, intermitentes y permanentes en los diferentes puntos de inyección del 8051.

3.3.6 Influencia del sistema seleccionado

Hasta ahora, todos los resultados mostrados hacen referencia al modelo del microcontrolador 8051. Sin embargo, ¿son generalizables los resultados a cualquier sistema? Para verificar que las tendencias descritas anteriormente pueden ser aplicables a cualquier computador, se han repetido los experimentos utilizando el modelo del Plasma, procesador RISC segmentado descrito en el apartado 3.3.2.2.

Por ejemplo, dado que el parámetro que más afecta a los porcentajes de averías y de errores latentes es el tiempo de actividad, se ha repetido el estudio de su influencia. Los experimentos de inyección han tenido los mismos parámetros que en el estudio equivalente realizado sobre el 8051. La figura 3.16 muestra los porcentajes de averías, y la figura 3.17 los porcentajes de errores latentes. Éstas son análogas a las figuras 3.7 y 3.8, pero con los datos obtenidos para el microprocesador Plasma.

En la figura 3.16 se puede observar que en los buses y la lógica combinacional el porcentaje de averías se incrementa enormemente al aumentar el tiempo de actividad. Ese incremento es mucho más moderado en los elementos de almacenamiento. También se puede observar que los fallos múltiples tienen mayor influencia que los fallos simples. Esto reproduce las tendencias observadas para el microcontrolador 8051.

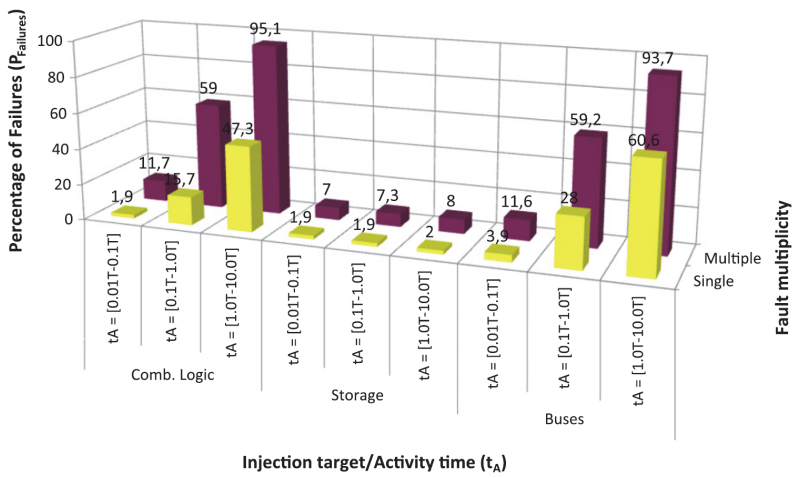

Figura 3.16: Influencia del tiempo de actividad y de la multiplicidad en el porcentaje de averías. Microprocesador Plasma [DGil12b].

La influencia de los fallos intermitentes en la lógica combinacional es muy alto, próximo al producido en los buses, especialmente al inyectar fallos múltiples. Además, su impacto es mucho mayor que en los elementos de almacenamiento. Esas serían las principales diferencias entre el 8051 y el Plasma. Por una parte, aunque las tendencias son similares, los valores absolutos obtenidos al inyectar fallos intermitentes en los elementos de almacenamiento son menores en el Plasma. Esto es debido a que la memoria del Plasma es mayor que la del 8051 (2 Kbytes el Plasma, por 128 bytes el 8051), lo que hace que la porción de memoria utilizada para la carga de trabajo sea menor en el Plasma. Por otra parte, el Plasma parece ser más sensible a los fallos intermitentes en la lógica combinacional que el 8051. Este resultado puede ser debido a las diferentes estructuras de los procesadores: el 8051 tiene una arquitectura CISC y su ruta de datos está basada principalmente en buses, mientras que el Plasma tiene una arquitectura RISC con segmentación, y su ruta de datos está basada en multiplexores y conexiones directas (no buses).

En la figura 3.17 se puede observar que, al igual que ocurría en el 8051, los errores latentes son mucho más frecuentes en los elementos de almacenamiento que en otras zonas del procesador.

En elementos de almacenamiento, el tiempo de actividad parece no tener ninguna influencia significativa, tanto si se consideran solo los errores latentes como si se observan los errores propagados (es decir, tratando conjuntamente averías y errores latentes).

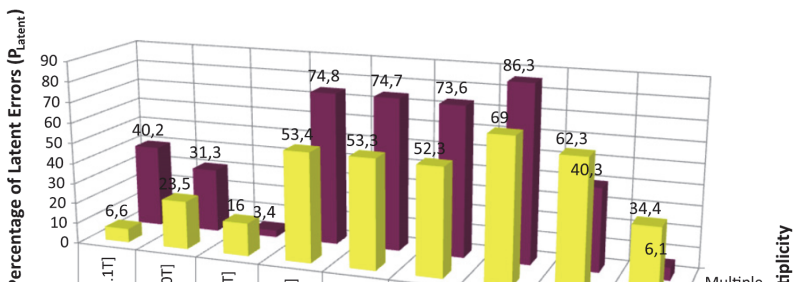

Figura 3.17: Influencia del tiempo de actividad y de la multiplicidad en el porcentaje de errores latentes. Microprocesador Plasma [DGil12b].

En los buses y en la lógica combinacional, en general, el porcentaje de errores latentes decrece al aumentar el tiempo de actividad. Esto es debido a que muchos fallos que con un tiempo de actividad menor se manifestarían como errores latentes, al aumentar el tiempo de actividad acaban convirtiéndose en averías. Al tratar averías y errores latentes conjuntamente, se puede observar que en todos los casos los porcentajes de errores propagados son mayores conforme aumenta el tiempo de actividad.

En general, se pueden observar más errores latentes en el Plasma que en el 8051. Si se consideran junto con las averías, también los porcentajes de errores propagados en el Plasma son apreciablemente superiores a los obtenidos en el 8051.

Por otro lado, también se han repetido los experimentos de inyección realizados con el objetivo de comparar los efectos de los fallos clasificados respecto a su persistencia: permanentes, intermitentes y transitorios. Los experimentos se han realizado de nuevo con los mismos parámetros. Los principales resultados se pueden observar en las figuras 3.18 y 3.19, que son equivalentes a las figuras 3.14 y 3.15, pero con los datos obtenidos para el microprocesador Plasma.

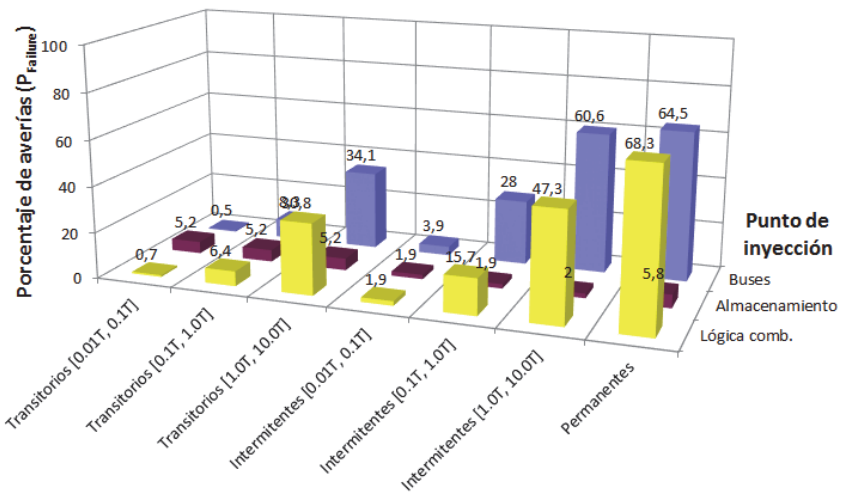

Figura 3.18: Comparación de los efectos de los fallos simples transitorios, intermitentes y permanentes en los diferentes puntos de inyección del Plasma.

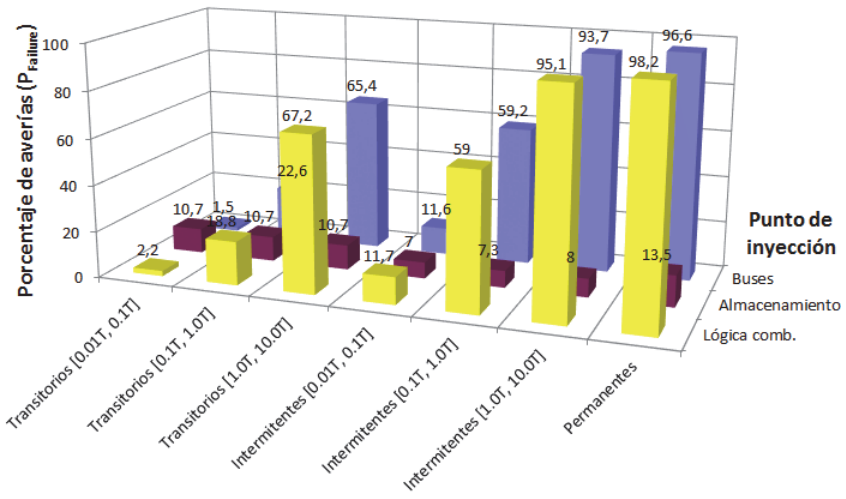

Figura 3.19: Comparación de los efectos de los fallos múltiples transitorios, intermitentes y permanentes en los diferentes puntos de inyección del Plasma.

Los resultados muestran las mismas tendencias ya observadas para el 8051: los fallos intermitentes provocan mayores porcentajes de averías que los fallos transitorios, para cada rango de tiempo de actividad, en buses y lógica combinacional, pero no en los elementos de almacenamiento. Los fallos permanentes son los que mayor porcentaje de averías provocan. Las razones son las mismas que se han expuesto anteriormente para el microcontrolador 8051 en la sección 3.3.5. Las principales diferencias que se pueden apreciar en los datos obtenidos para el Plasma respecto a los del 8051 son los mayores porcentajes de averías para todos los tipos de fallos en la lógica combinacional. Esta tendencia ya se ha apreciado, y justificado, anteriormente en esta misma sección.

La tabla 3.2 muestra un resumen de los análisis realizados y las principales conclusiones de la comparación de los sistemas bajo estudio.

Tabla 3.2. Comparación de los efectos de los fallos intermitentes en los procesadores Plasma y 8051 [Gracia14a]

Análisis	Comparación Plasma vs 8051
Influencia del tiempo de actividad	Tendencias similares (porcentaje de averías): • Buses y lógica combinacional: incremento aproximadamente logarítmico • Almacenamiento: influencia despreciable
Influencia del tiempo de inactividad	Tendencia similar: despreciable
Influencia de la longitud de la ráfaga	Tendencias similares (porcentaje de averías): • Buses y lógica combinacional: ascenso asintótico • Almacenamiento: influencia despreciable
Influencia del lugar de inyección	Porcentaje de averías: • Plasma: buses > lógica combinacional > almacenamiento • 8051: buses > almacenamiento > lógica combinacional
Comparación con fallos transitorios y permanentes	Tendencias similares: • Buses y lógica combinacional: permanentes > intermitentes > transitorios • Almacenamiento: permanentes > transitorios(*) > intermitentes (*) Por la baja tasa de celdas sobrescritas
Impacto de los fallos en los buses	Porcentaje de averías similar para ambos procesadores Porcentaje de errores latentes similar para ambos procesadores
Impacto de los fallos en la lógica combinacional	Porcentaje de averías mayor en el Plasma Porcentaje de errores latentes mayor en el Plasma
Impacto de los fallos en elementos de almacenamiento	Porcentaje de averías mayor en el 8051 Porcentaje de errores latentes mayor en el Plasma

3.3.7 Comparación con los resultados de otros estudios

La importancia creciente de los efectos de los fallos intermitentes ha focalizado la atención de la comunidad científica durante estos últimos años, por lo que diversos autores han realizado diferentes estudios, desde distintos puntos de vista y niveles de abstracción. A continuación se indican algunos de ellos, comparando sus conclusiones con los resultados mostrados en esta tesis.

Cristian Constantinescu es uno de los autores que lleva más tiempo estudiando las causas y mecanismos de los fallos intermitentes, y sus efectos, tanto a nivel de datos reales a partir de registros de errores como a nivel de simulación e inyección de fallos. Por ejemplo, en [Constantinescu08] se hace un estudio monitorizando 257 servidores de dos fabricantes distintos, acumulando 310,7 años de trabajo en total. Analizando los registros de errores, se comprueba que el 6,2% de los errores producidos en los subsistemas de memoria son debidos a fallos intermitentes. También se registraron fallos intermitentes en los buses del sistema. Esto demuestra la importancia de los fallos intermitentes, conclusión a la que se llega también con esta tesis.

Un trabajo que merece especial atención es [Schroeder09], donde se realiza un análisis sobre registros de datos de errores en la RAM dinámica de servidores de *Google* durante dos años y medio. La conclusión más importante de este estudio es que los errores en memoria son principalmente causados por el *hardware*, más que por causas ambientales. Además, esos errores en el *hardware* no son en su totalidad causados por fallos permanentes sino que aparecen, con bastante frecuencia, fallos que se activan y se desactivan repetitivamente en el mismo sitio. Este trabajo coincide, pues, en la importancia de los fallos intermitentes. Sin embargo, hay una discrepancia con lo presentado en esta tesis. Según este estudio, la tasa de errores en memoria es órdenes de magnitud superior a lo publicado previamente por otros autores cuando se consideran condiciones de laboratorio. Efectivamente, en esta tesis se han presentado porcentajes de averías muy bajos cuando se inyectaban fallos en memoria. No obstante, la explicación es lógica: nuestros experimentos consideran una carga de trabajo que utiliza una pequeña porción de la memoria, por lo que los fallos se manifestaban, en su mayoría, no como averías sino como errores latentes. Es lógico pensar que en un sistema real, con una carga de trabajo importante, la mayor parte de la memoria esté ocupada, por lo que se producirían más averías que errores latentes.

En [Pan10] se define una nueva métrica, denominada IVF (*Intermittent Vulnerability Factor*), para estudiar el efecto de los fallos intermitentes en los bloques internos de los microprocesadores. Se realizan experimentos de inyección de fallos sobre

un modelo del Alpha 21264, un microprocesador RISC de *Digital Equipment Corporation* (DEC). Los autores llegan a conclusiones similares a algunas de las presentadas en esta tesis: mayores tiempos de actividad o ráfagas con más activaciones causan más averías, fallos en los registros especiales tienen un gran impacto y, finalmente, los fallos intermitentes provocan más averías en el sistema que los fallos transitorios.

En [Nightingale11], investigadores de *Microsoft* analizan los datos del *Windows Error Reporting system*, que permite dar soporte al diagnóstico de fallos en el *software* ocurridos en todas las máquinas con sistema operativo *Microsoft Windows*™. Aunque el estudio tiene algunas limitaciones, la principal ventaja es la inmensa cantidad de datos disponible (755.539 registros). Una de las conclusiones de este estudio es que buena parte de las averías inducidas en el *hardware* son recurrentes. Por ejemplo, se estima que el 39% de las máquinas que han sufrido una avería en la CPU han tenido como causa un fallo intermitente. También se obtienen porcentajes importantes para las averías en memoria y discos duros.

Finalmente, en [Wei11] se comparan los efectos de los fallos intermitentes y transitorios en los programas de aplicación ejecutados en el modelo de un procesador RISC segmentado. Este estudio muestra que ambos tipos de fallo presentan diferencias sustanciales en el porcentaje de averías causadas en los programas. Además, se verifica la influencia importante del origen *hardware* del fallo intermitente (es decir, el lugar de inyección), y del tamaño del fallo (es decir, su duración total, incluyendo todas las activaciones del fallo). Estas conclusiones se pueden considerar similares a algunas de las obtenidas en esta tesis.

3.4 Técnicas de mitigación existentes

Una vez justificado que los fallos intermitentes pueden tener efectos muy perniciosos para el normal funcionamiento de un sistema informático, hay que comprobar si las técnicas de mitigación existentes son capaces de responder adecuadamente ante su aparición. Los mecanismos de tolerancia a fallos (FTM, del inglés *Fault Tolerant Mechanisms*) existentes actualmente están diseñados, en su mayoría, para tolerar fallos permanentes (a veces también llamados *hard faults*) o transitorios (cuando producen un error, éste es conocido habitualmente como *soft error*).

Como se comentó anteriormente, los fallos permanentes se producen por un daño irreversible en el *hardware*, con lo que éste no puede continuar funcionando sin un riesgo muy elevado de avería. En ausencia de mecanismos de tolerancia a fallos, la única solución es el reemplazo del *hardware* afectado.

Lo contrario sucede con los fallos transitorios, que son debidos a causas externas y que normalmente no vuelven a producirse en el mismo sitio. El reemplazo del *hardware* no puede solucionar los fallos transitorios, ya que el *hardware* sigue funcionando correctamente. Evidentemente, estos fallos son más difíciles de detectar. Dependiendo de diversos factores (duración del fallo, lugar y momento en el que ocurre, etc.) pueden o no causar error. A su vez, ese error puede causar una avería, o bien permanecer como un error latente, sin llegar a producirla.

Aunque hay mecanismos de tolerancia a fallos que son adecuados para ambos tipos de fallos, generalmente están diseñados pensando principalmente para uno u otro tipo. Un ejemplo de FTM pensado para fallos permanentes son los componentes de repuesto (*spare parts*), que permanecen inactivos en ausencia de fallo, y sustituyen al componente original cuando se detecta un fallo permanente. Un ejemplo de FTM pensado para fallos transitorios son los mecanismos de reintento de operaciones tras la detección de un fallo. Evidentemente, este reintento no funcionaría en caso de fallo permanente, pero puede solucionar los fallos transitorios. Finalmente, un ejemplo de FTM que puede ser válido tanto para fallos permanentes como transitorios puede ser un sistema *n* modular redundante (por ejemplo, un TMR o triple modular redundante), donde varios circuitos realizan la misma operación y un votador decide qué salida se ha dado en la mayoría de ellos, dando esa salida por buena.

Para el estudio del comportamiento de los mecanismos de tolerancia a fallos ante la ocurrencia de fallos intermitentes se han realizado experimentos de validación de un sistema tolerante a fallos, que incluye varios de estos mecanismos, y se ha analizado su respuesta ante la inyección de fallos intermitentes. Nuevamente, para los experimentos de inyección se ha utilizado la herramienta VFIT. En este caso, se ha utilizado el modelo VHDL de un sistema tolerante a fallos basado en el procesador académico MARK2 [Armstrong89]. El sistema es dúplex, con procesador de repuesto en frío, detección de paridad y temporizador de guardia [DGil99]. Se trata de un sistema tolerante a fallos sencillo, pero la estructura dual con repuesto en frío es habitual en sistemas no críticos tolerantes a fallos, como sistemas de alta disponibilidad y de vida prolongada, por lo que puede ser considerado suficientemente representativo para el ámbito de este trabajo.

3.4.1 Sistema bajo prueba: MARK2 tolerante a fallos

El sistema que se va a validar, cuyo esquema puede observarse en la figura 3.20, es un sistema dual con repuesto en frío, como se ha comentado anteriormente. Tanto el procesador principal como el de repuesto son una versión mejorada del procesador MARK2 [Armstrong89]. Se han añadido al sistema distintos mecanismos de tolerancia a fallos para mejorar su confiabilidad. Como mecanismos de detección se han incluido control de paridad en la memoria y control del flujo del programa mediante la utilización de un temporizador de guardia (*watchdog timer*). Como mecanismos de recuperación dispone de un ciclo adicional de reintento de instrucción cuando se detecta un error de paridad, puntos de control para recuperarse cuando detecta errores el temporizador de guardia, y arranque del procesador de repuesto en caso de errores permanentes. El número de errores detectados requerido para activar el procesador de repuesto puede ser configurado en el sistema [DGil99].

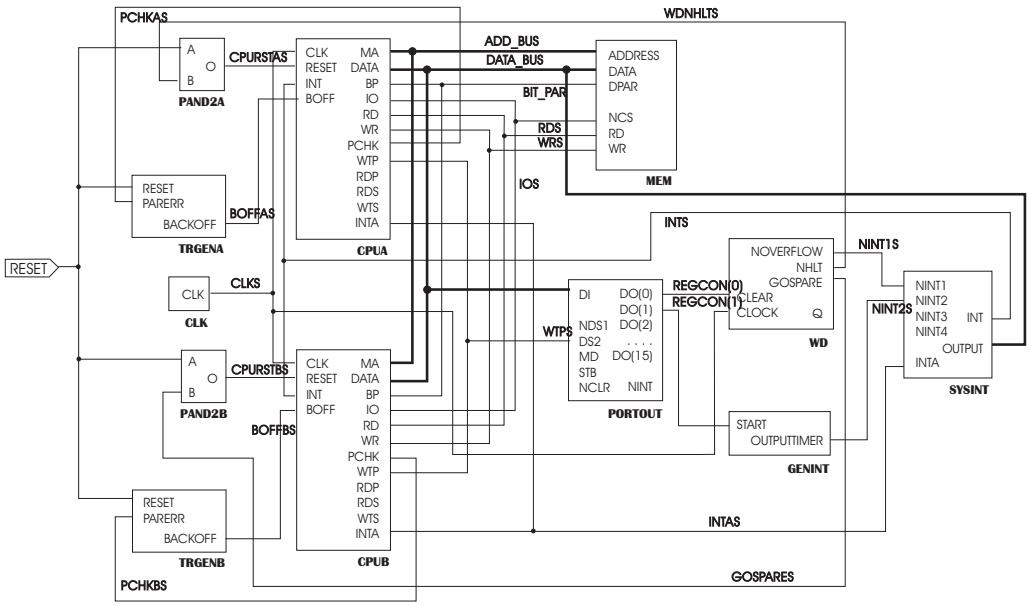

Figura 3.20: Diagrama de bloques del computador MARK2 tolerante a fallos [DGil99].

La arquitectura estructural del modelo se compone de los procesadores principal y de repuesto (CPUA y CPUB en el esquema), memoria RAM (MEM), puerto paralelo de salida (PORTOUT), controlador de interrupciones (SYSINT), generador de señal de reloj (CLK), procesador de guardia (WD), generador de pulsos (GENINT), dos generadores de ciclo de reintento (TRGENA, TRGENB) y dos puertas AND (PAND2A, PAND2B). Cada componente está modelado en VHDL mediante una arquitectura comportamental. Los buses de datos y direcciones son más anchos que en el MARK2 original, y se han añadido nuevos registros, instrucciones y modos de direccionamiento. Se ha añadido detección de paridad a los datos, que es comprobada en cada lectura de memoria. El bit de paridad se genera en los ciclos de escritura. Además, se han añadido varias señales de control para la comprobación de la paridad (PCHK) y la generación del ciclo de reintento (BOFF). A continuación se describen los mecanismos de detección y recuperación con un poco más de detalle.

Cuando el procesador detecta un error de paridad, el generador de ciclo de reintento activa su señal durante un tiempo fijo (que puede ser configurado). En esta situación, el procesador espera por si la causa del error termina. Cuando el tiempo expira, el procesador ejecuta de nuevo la última instrucción. En caso de que el error de paridad persista, la señal de reintento se activa permanentemente.

El procesador usa una interrupción periódica (NINT2) para tratar errores en el flujo del programa. Cada vez que se recibe la interrupción, la rutina de respuesta inicia el temporizador de guardia para evitar su desbordamiento, y el componente GENINT se activa para producir una nueva interrupción. Estas acciones se llevan a cabo a través del puerto paralelo de salida. Un error en el control de flujo del programa durante la rutina de tratamiento de la interrupción producirá desbordamiento en el procesador de guardia. Esto, a su vez, activará una señal de interrupción del procesador (NINT1) que hará que el sistema se recupere desde un punto de recuperación previamente almacenado en memoria estable. Hay un segundo banco de memoria con una copia de seguridad del punto de recuperación y una variable que indica cuál es el banco activo. De esta forma, la integridad de los datos está asegurada en todo momento.

En caso de dos desbordamientos sucesivos del temporizador de guardia, la señal NHLT se activa para detener permanentemente el procesador principal y arrancar el de repuesto. Éste recupera el estado desde el punto de recuperación almacenado en la memoria estable para continuar con las tareas del procesador principal.

Las cargas de trabajo utilizadas en este caso son dos de las descritas anteriormente en la sección 3.3.2.3: la serie aritmética y la ordenación mediante el algoritmo de la burbuja. Ambas son cargas típicas en este tipo de experimentos, y con una duración moderada para no hacer demasiado largas las simulaciones.

3.4.2 Definiciones y terminología

Además de las definiciones y abreviaturas presentadas en la sección 3.3.3, se van a introducir algunas nuevas relacionadas con los resultados que se espera obtener (porcentajes de detección y recuperación):

Act	Porcentaje de errores activados (ha cambiado alguna señal o variable del modelo, o el fallo se ha propagado a una señal externa). Se define como $Act = (N_{Propagated} / N_{Injected}) * 100$
No_Effect	Porcentaje de errores no efectivos (errores sobrescritos o que permanecen latentes en el sistema, pero no provocan avería). Se define como $No\_effect = P_{Non\_effective}$
Detect	Porcentaje de errores detectados
ND_Fail	Porcentaje de errores no detectados que han provocado una avería
D_PAR	Porcentaje de errores detectados por la paridad
D_WD	Porcentaje de errores detectados por el temporizador de guardia
Recov	Porcentaje de errores recuperados por los mecanismos de recuperación de errores
NR_Fail	Porcentaje de errores no recuperados que han provocado una avería
R_BOFF	Porcentaje de errores recuperados mediante ciclo de reintento
R_CP	Porcentaje de errores recuperados mediante puntos de control
R_SP	Porcentaje de errores recuperados mediante el procesador de repuesto

En la figura 3.21 se muestra el *grafo de predicados de los mecanismos tolerantes a fallos* [Arlat93]. Este diagrama refleja la patología de los fallos, es decir, la evolución de los fallos desde que éstos son inyectados hasta que los errores son detectados y, eventualmente, recuperados por los mecanismos tolerantes a fallos. A partir del grafo, se pueden calcular las latencias medias de propagación, detección y recuperación, y las coberturas de detección y recuperación. El análisis se puede realizar tanto a nivel global del sistema, como para cada mecanismo de detección o recuperación en particular.

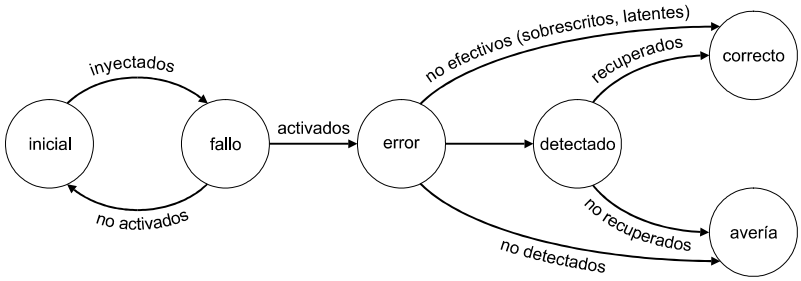

Figura 3.21: Grafo de predicados de los mecanismos tolerantes a fallos [Arlat93].

3.4.3 Respuesta ante fallos en buses

Como se ha visto en la sección 3.3, los buses son un elemento crítico del sistema, y son especialmente sensibles a los fallos intermitentes. En este apartado se presentan los resultados de distintos experimentos de inyección de fallos realizados con el objetivo de estudiar la respuesta de un sistema tolerante a fallos tras la aparición de fallos intermitentes en los buses. Se han obtenido los datos necesarios para establecer la patología de los fallos, lo que permite determinar si los sistemas de detección/recuperación son efectivos o no, y cuáles son más efectivos. En primer lugar se hace un estudio variando los parámetros temporales de la ráfaga, y a continuación se analizan los efectos al inyectar distintos modelos de fallo.

3.4.3.1 Influencia de los parámetros temporales de la ráfaga

Para llevar a cabo este estudio se han realizado un total de 36 experimentos de inyección de fallos. Éstos son los principales parámetros:

Sistema bajo estudio: MARK2 tolerante a fallos.

Cargas de trabajo (implica experimentos distintos): serie aritmética y *bubblesort*.

Punto de inyección: buses del sistema.

Técnica de inyección: *Órdenes del simulador*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelo de fallo: *intermittent pulse.*

Tiempo de actividad (t_A): generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Tiempo de inactividad (t_I): generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Longitud de la ráfaga (L_{Burst}): generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: todos los definidos en la sección 3.4.2.

Los resultados de estos experimentos se pueden observar en las tablas 3.3, 3.4, 3.5 y 3.6. Como se puede observar, los porcentajes de errores activados (columna *Act*) son superiores al 93% en todos los casos. Esto indica que los buses son un elemento crítico del sistema, y muy sensibles a fallos intermitentes, lo que coincide con la conclusión alcanzada en la sección 3.3.4. Los porcentajes son ligeramente más elevados para la carga de trabajo que ejecuta el algoritmo *bubblesort*, ya que ésta usa los buses más intensamente.

En cuanto a la detección de errores, se pueden observar porcentajes de errores detectados (columna *Detect*) entre el 47% y el 93% para fallos simples, y entre el 70% y el 99% para fallos múltiples. Los mayores porcentajes se presentan para la carga *bubblesort*, por el mayor tráfico generado, y van aumentando conforme aumenta el tiempo de actividad, como era previsible.

El porcentaje de averías provocadas por errores no detectados (columna *ND_Fail*) es muy bajo, inferior al 5% en el peor de los casos. Esto indica que casi todos los errores activados se detectan o no tienen efecto (columna *No_effect*).

La paridad es el mecanismo de detección de errores más efectivo para ambas cargas de trabajo (columna *D_PAR*). Para la carga de trabajo que ejecuta la serie aritmética, e inyectando fallos simples, los porcentajes son similares (entre el 81% y el 85%), mientras que para el resto de las combinaciones la variación es mayor (entre el 76% y el 85% para fallos simples en el *bubblesort*, y entre el 79% y el 99% para fallos múltiples). Además, en estos casos se aprecia una diferencia sensible entre los porcentajes

de errores detectados cuando el tiempo de actividad es menor. Es decir, las ráfagas de fallos intermitentes cuyo tiempo de actividad es menor se detectan con mayor dificultad por parte de la paridad. El temporizador de guardia (columna *D_WD*) detecta menos fallos que la paridad (valores entre el 15% y el 24% para fallos simples, y hasta el 21% para fallos múltiples).

No se aprecia una influencia significativa del tiempo de inactividad en los porcentajes de errores detectados.

Tabla 3.3. Efectos de los fallos intermitentes simples en los buses (serie aritmética)

t_A = [0.01T, 0.1T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	94.90	51.74	47.63	0.63	83.41	16.59	63.72	26.33	84.38	2.43	13.19
[0.1T, 1.0T]	93.60	53.31	45.94	0.75	81.63	18.37	59.30	31.16	80.00	3.53	16.47
[1.0T, 10.0T]	93.60	50.64	48.40	0.96	81.90	18.10	56.29	33.33	60.39	2.35	37.25
t_A = [0.1T, 1.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	93.70	37.46	59.55	2.99	83.15	16.85	59.50	29.93	80.72	4.82	14.46
[0.1T, 1.0T]	94.30	37.75	60.02	2.23	82.86	17.14	59.36	30.04	55.36	7.14	37.50
[1.0T, 10.0T]	93.90	37.81	58.04	4.15	84.77	15.23	59.63	32.11	38.15	6.15	55.69
t_A = [1.0T, 10.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	93.70	23.27	72.36	4.38	82.15	17.85	62.83	27.73	16.43	7.04	76.53
[0.1T, 1.0T]	94.30	23.44	71.69	4.88	81.21	18.79	64.64	28.25	15.10	9.84	75.06
[1.0T, 10.0T]	94.80	24.37	70.99	4.64	84.99	15.01	66.72	29.12	15.37	9.35	75.28

Tabla 3.4. Efectos de los fallos intermitentes simples en los buses (*bubblesort*)

t_A = [0.01T, 0.1T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	95.30	37.57	61.70	0.73	78.57	21.43	75.68	23.30	57.53	2.02	40.45
[0.1T, 1.0T]	95.30	39.56	59.50	0.94	76.72	23.28	71.78	26.98	55.28	1.72	43.00
[1.0T, 10.0T]	94.30	38.71	60.66	0.64	76.05	23.95	69.06	28.85	37.47	3.04	59.49
t_A = [0.1T, 1.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	94.60	18.92	79.39	1.69	83.22	16.78	75.63	23.70	55.46	1.58	42.96
[0.1T, 1.0T]	94.80	13.92	84.07	2.00	82.81	17.19	75.41	23.96	41.10	3.16	55.74
[1.0T, 10.0T]	94.40	13.77	84.64	1.59	81.73	18.27	73.84	25.03	23.39	2.37	74.24
t_A = [1.0T, 10.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	94.00	6.06	93.72	0.21	84.45	15.55	76.73	22.36	11.54	1.48	86.98
[0.1T, 1.0T]	93.80	5.22	94.03	0.75	84.58	15.42	77.21	21.88	9.69	1.47	88.84
[1.0T, 10.0T]	94.70	6.55	93.03	0.42	83.54	16.46	75.60	23.38	6.16	1.20	92.64

Tabla 3.5. Efectos de los fallos intermitentes múltiples en los buses (serie aritmética)

t_A = [0.01T, 0.1T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	98.90	25.38	74.22	0.40	88.83	11.17	80.11	14.71	52.55	26.70	20.75
[0.1T, 1.0T]	98.60	24.44	74.85	0.71	87.80	12.20	79.95	14.36	53.39	24.07	22.54
[1.0T, 10.0T]	99.00	28.48	70.81	0.71	88.30	11.70	79.89	14.12	34.82	19.11	46.07
t_A = [0.1T, 1.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	98.80	9.62	89.17	1.21	97.16	2.84	85.02	13.39	51.40	24.30	24.30
[0.1T, 1.0T]	98.60	9.63	89.45	0.91	97.17	2.83	86.62	11.79	33.38	15.71	50.92
[1.0T, 10.0T]	98.10	9.38	89.50	1.12	97.04	2.96	85.65	12.64	14.89	6.65	78.46
t_A = [1.0T, 10.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	98.90	5.76	93.73	0.51	97.63	2.37	89.32	9.71	7.73	4.35	87.92
[0.1T, 1.0T]	98.70	5.37	94.12	0.51	96.99	3.01	90.10	9.26	6.69	4.54	88.77
[1.0T, 10.0T]	98.60	6.39	92.80	0.81	96.07	3.93	87.32	12.35	4.26	4.26	91.49

Tabla 3.6. Efectos de los fallos intermitentes múltiples en los buses (*bubblesort*)

t_A = [0.01T, 0.1T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	99.10	15.74	84.06	0.20	84.51	15.49	90.76	8.04	28.31	2.38	69.31
[0.1T, 1.0T]	98.60	14.40	85.50	0.10	83.04	16.96	88.73	9.96	29.14	2.54	68.32
[1.0T, 10.0T]	99.00	13.74	86.06	0.20	79.46	20.54	85.92	11.97	15.85	2.60	81.56
t_A = [0.1T, 1.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	99.00	3.03	96.77	0.20	97.08	2.92	89.98	10.02	29.81	3.13	67.05
[0.1T, 1.0T]	99.20	3.73	95.97	0.30	97.16	2.84	88.66	11.13	19.67	2.49	77.84
[1.0T, 10.0T]	99.20	2.92	96.57	0.50	96.66	3.34	84.66	14.72	5.55	0.74	93.71
t_A = [1.0T, 10.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	99.00	1.82	98.08	0.10	98.25	1.75	87.54	12.26	3.41	0.35	96.24
[0.1T, 1.0T]	99.00	1.21	98.59	0.20	98.16	1.84	87.40	12.40	2.58	0.12	97.30
[1.0T, 10.0T]	98.90	1.11	98.69	0.20	97.34	2.66	87.09	12.70	1.06	0.47	98.47

En cuanto al tratamiento de los errores detectados, hay que puntualizar que la suma de los errores recuperados (columna *Recov*) y los errores no recuperados que provocan una avería (columna *NR_Fail*) puede ser inferior al 100%, ya que algunos errores se recuperan por la redundancia intrínseca del sistema, como se ha mencionado en el apartado 3.3.4.2. Los porcentajes de la columna *NR_Fail* son elevados para fallos simples y ambas cargas de trabajo (entre el 23% y el 34%). Sin embargo, los valores son sensiblemente inferiores para fallos múltiples (entre el 8% y el 15%). Aunque en un principio esto pueda resultar sorprendente, el análisis de los valores obtenidos por los distintos mecanismos de recuperación explica esta situación.

La recuperación mediante puntos de control presenta los menores porcentajes de errores recuperados (columna *R_CP*), principalmente para el algoritmo *bubblesort*. Esto es debido al bajo porcentaje de errores detectados por el temporizador de guardia. Entre el ciclo de reintento y el procesador de repuesto se recuperan la mayoría de los errores. El mecanismo de ciclo de reintento (columna *R_BOFF*) responde bien ante fallos con menor tiempo de actividad, mientras que para tiempos de actividad mayores el procesador de repuesto (columna *R_SP*) recupera la mayor parte de errores. Esto se hace todavía más patente para los casos en que se inyectan fallos múltiples, donde hasta el 98.47% de los errores recuperados lo son por el procesador de repuesto. En este caso, el sistema interpreta que el fallo es permanente.

Esta última conclusión justifica el resultado del párrafo anterior: al considerar fallos múltiples (o bien en fallos simples, al aumentar el tiempo de actividad) aumenta la probabilidad de que el fallo sea detectado y considerado un fallo permanente. Si bien el procesador de repuesto recupera gran parte de los errores, el problema surge en que este mecanismo es muy lento, lo que ralentiza todo el proceso de recuperación de fallos intermitentes. La conclusión es que, aunque los mecanismos de detección de errores funcionan bastante bien, los mecanismos de recuperación deberían ser mejorados.

3.4.3.2 Influencia de los modelos de fallo

Con el objeto de estudiar las posibles diferencias de comportamiento ante los distintos modelos de fallo, se han realizado nuevos experimentos de inyección de fallos. En este caso se ha utilizado la técnica de inyección basada en *perturbadores*, lo que permite experimentar con más modelos de fallos que con *órdenes del simulador*. Los principales parámetros de estos experimentos son los siguientes:

Sistema bajo estudio: MARK2 tolerante a fallos.

Cargas de trabajo (implica experimentos distintos): serie aritmética y *bubblesort*.

Punto de inyección: buses del sistema.

Técnica de inyección: *Perturbadores*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelos de fallo (cada modelo supone un experimento distinto): *intermittent pulse*, *intermittent short*, *intermittent open* e *intermittent delay*.

Valor del retardo (solo para el modelo *intermittent delay*): generado mediante una distribución Uniforme en el rango [0.1T, 1.5T].

Tiempo de actividad (t_A): generado mediante una distribución Uniforme en el rango [0.1T, 1.0T].

Tiempo de inactividad (t_I): generado mediante una distribución Uniforme en el rango [0.1T, 1.0T].

Longitud de la ráfaga (L_{Burst}): generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: todos los definidos en la sección 3.4.2.

Los resultados obtenidos se han recogido en las tablas 3.7 y 3.8. Se puede observar que el modelo *intermittent pulse* presenta el mayor porcentaje de errores activados (columna *Act*). El menor porcentaje corresponde al modelo *intermittent delay*. Esto es debido a fenómenos de enmascaramiento temporal en componentes síncronos, ya mencionados en la sección 3.3.4.2, y explicados con más detalle en [Baraza03].

El porcentaje de errores detectados (columna *Detect*) muestra el mismo comportamiento que los errores activados. Por el contrario, los porcentajes de errores no efectivos (columna *No_effect*) y de averías debidas a errores no detectados (columna *ND_Fail*) presentan el comportamiento opuesto. Es decir, el menor valor corresponde a *intermittent pulse* y el mayor a *intermittent delay*.

El porcentaje de averías debidas a errores no detectados es, en general, muy bajo, excepto para el modelo *intermittent delay*, con valores por encima del 9% para la carga que ejecuta el algoritmo *bubblesort*.

El mecanismo de detección de errores más efectivo es la paridad (columna *D_PAR*), como ya se había observado en la sección 3.4.3.1. No obstante, para el modelo *intermittent delay*, el temporizador de guardia (columna *D_WD*) detecta un porcentaje de errores no despreciable (entre el 6% y el 9%).

En cuanto a la recuperación de errores (columna *Recov*), se observan porcentajes con poca variación en general para errores simples, mientras que para fallos múltiples los valores correspondientes a *intermittent delay* son sensiblemente menores. Las averías provocadas por errores detectados pero no recuperados (columna *NR_Fail*) muestra la tendencia contraria, alcanzando un porcentaje superior al 20% para *intermittent delay*.

Tabla 3.7. Efectos de los fallos intermitentes en los buses (serie aritmética)

Fallos simples											
Modelo de fallo	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Int. pulse	83.10	19.98	77.98	2.05	97.53	2.47	86.73	12.96	56.58	4.09	39.32
Int. open	68.50	33.58	63.07	3.36	96.06	3.94	88.66	11.11	52.22	2.61	45.17
Int. short	68.40	21.20	75.73	3.07	94.79	5.21	81.66	16.41	54.14	3.78	42.08
Int. delay	59.30	33.56	60.20	6.24	93.84	6.16	86.83	11.48	50.00	1.94	48.06
Fallos múltiples											
Modelo de fallo	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Int. pulse	97.60	10.04	89.45	0.51	98.85	1.15	93.59	6.19	52.51	2.08	45.41
Int. open	95.00	17.47	81.16	1.37	96.76	3.24	86.64	12.97	55.54	2.69	41.77
Int. short	94.90	17.49	81.35	1.16	96.89	3.11	87.82	11.66	55.60	3.10	41.30
Int. delay	92.40	29.65	64.39	5.95	91.43	8.57	75.97	22.69	44.25	4.87	50.88

Tabla 3.8. Efectos de los fallos intermitentes en los buses (*bubblesort*)

Fallos simples											
Modelo de fallo	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Int. pulse	83.10	19.25	76.90	3.85	98.90	1.10	90.61	9.23	54.75	1.04	44.21
Int. open	68.80	34.59	60.32	5.09	96.63	3.37	93.49	6.02	45.62	1.55	52.84
Int. short	67.90	20.62	75.26	4.12	94.91	5.09	87.28	12.33	46.19	2.69	51.12
Int. delay	50.40	44.25	46.03	9.72	90.95	9.05	84.05	15.52	40.51	4.10	55.38
Fallos múltiples											
Modelo de fallo	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Int. pulse	97.70	10.24	88.84	0.92	99.42	0.58	95.28	4.72	48.97	1.33	49.70
Int. open	95.50	17.70	80.10	2.20	96.99	3.01	86.41	13.46	51.44	1.36	47.20
Int. short	94.90	15.38	79.98	4.64	96.97	3.03	90.38	9.62	50.15	1.90	47.96
Int. delay	93.00	27.53	63.23	9.25	93.20	6.80	81.12	18.71	43.40	3.14	53.46

Los mecanismos de recuperación muestran tendencias similares a las ya observadas en la sección 3.4.3.1. La recuperación mediante puntos de control (columna *R_CP*) tiene una aportación residual. La recuperación mediante ciclo de reintento (columna *R_BOFF*) y la recuperación mediante procesador de repuesto (columna *R_SP*) se reparten, a partes aproximadamente iguales, el grueso de los errores recuperados. Es decir, si el ciclo de reintento no es capaz de recuperar el error, el sistema lo considera causado por un fallo permanente, lo que hace que se ponga en marcha la recuperación mediante procesador de repuesto.

La conclusión en cuanto a los distintos modelos de fallos es que, en general, *intermittent pulse*, *intermittent short* e *intermittent open* ofrecen porcentajes similares. Esto es debido a que la forma en que se manifiestan es similar, aunque las causas y mecanismos físicos puedan ser distintos. Por el contrario, el comportamiento del modelo *intermittent delay* difiere de los anteriores, y se manifiesta como un error de temporización. Se espera un incremento en la incidencia de este tipo de errores, por lo que deben ser tenidos muy en cuenta.

3.4.4 Respuesta ante fallos en memoria

En esta sección se presentan algunos resultados de experimentos de inyección de fallos intermitentes en la memoria externa del sistema bajo estudio. En este caso se estudia la influencia de los parámetros temporales de la ráfaga, con respecto a la multiplicidad de los fallos y con distintas cargas de trabajo. Este análisis es similar al realizado para los buses del sistema en la sección 3.4.3.1. Para ello se ha realizado una serie de experimentos de inyección, con estos parámetros:

Sistema bajo estudio: MARK2 tolerante a fallos.

Cargas de trabajo (implica experimentos distintos): serie aritmética y *bubblesort*.

Punto de inyección: memoria externa del sistema (componente MEM del modelo, ver figura 3.20).

Técnica de inyección: *Órdenes del simulador*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelo de fallo: *intermittent stuck-at (0, 1)*.

Tiempo de actividad (t_A): generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Tiempo de inactividad (t_I): generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Longitud de la ráfaga (L_{Burst}): generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: todos los definidos en la sección 3.4.2.

Los resultados de estos experimentos indican que los porcentajes de errores activados son muy pequeños. Esto es debido a que ambas cargas de trabajo utilizan una porción reducida de la memoria (inferior al 5%). Dado que los puntos de inyección se

seleccionan aleatoriamente entre toda la memoria, la probabilidad de que un fallo inyectado afecte a la carga de trabajo y el fallo se propague a señales externas es muy pequeña. Por este motivo únicamente se muestran, en las tablas 3.9 y 3.10, los resultados obtenidos para fallos múltiples, que son ligeramente superiores y permiten analizarlos mejor. En todo caso, los valores obtenidos son demasiado bajos para ser estadísticamente significativos, por lo que las tendencias observadas hay que considerarlas con mucha prudencia.

Tabla 3.9. Efectos de los fallos intermitentes múltiples en la memoria (serie aritmética)

t_A = [0.01T, 0.1T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	0.00										
[0.1T, 1.0T]	0.00										
[1.0T, 10.0T]	0.10	0.00	100	0.00	100	0.00	100	0.00	100	0.00	0.00
t_A = [0.1T, 1.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	0.00										
[0.1T, 1.0T]	0.10	0.00	100	0.00	100	0.00	100	0.00	100	0.00	0.00
[1.0T, 10.0T]	0.20	0.00	100	0.00	100	0.00	100	0.00	100	0.00	0.00
t_A = [1.0T, 10.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	1.50	0.00	93.33	6.67	100	0.00	92.86	7.14	23.08	0.00	76.92
[0.1T, 1.0T]	1.80	0.00	94.44	5.56	100	0.00	94.12	5.88	18.75	0.00	81.25
[1.0T, 10.0T]	1.20	8.33	91.67	0.00	100	0.00	100	0.00	18.18	0.00	81.82

Tabla 3.10. Efectos de los fallos intermitentes múltiples en la memoria (*bubblesort*)

t_A = [0.01T, 0.1T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	0.00										
[0.1T, 1.0T]	0.00										
[1.0T, 10.0T]	0.10	100	0.00								
t_A = [0.1T, 1.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	0.30	0.00	100	0.00	100	0.00	100	0.00	100	0.00	0.00
[0.1T, 1.0T]	0.50	0.00	100	0.00	100	0.00	100	0.00	60.00	0.00	40.00
[1.0T, 10.0T]	0.30	33.33	66.67	0.00	100	0.00	100	0.00	50.00	0.00	50.00
t_A = [1.0T, 10.0T]											
t_I	Act	No_Effect	Detect	ND_Fail	D_PAR	D_WD	Recov	NR_Fail	R_BOFF	R_CP	R_SP
[0.01T, 0.1T]	3.00	6.67	93.33	0.00	100	0.00	100	0.00	28.57	0.00	71.43
[0.1T, 1.0T]	3.00	3.33	96.67	0.00	100	0.00	100	0.00	34.48	0.00	65.52
[1.0T, 10.0T]	2.30	0.00	100	0.00	95.65	4.35	100	0.00	43.48	4.35	52.17

Como era previsible, los porcentajes de errores activados (columna *Act*) son mayores conforme aumenta el tiempo de actividad. Los porcentajes son más elevados para la carga de trabajo que ejecuta el algoritmo *bubblesort*, debido al mayor número de operaciones de lectura y escritura realizadas en memoria.

La inmensa mayoría de los errores activados son detectados (columna *Detect*). En general, aparecen pocos errores activados no efectivos o averías producidas por errores no detectados (columnas *No_effect* y *ND_Fail*). El mecanismo de paridad detecta casi el 100% de los errores.

Los errores detectados son recuperados casi al 100%. Como en el caso de los buses, la recuperación mediante puntos de control (columna *R_CP*) presenta porcentajes muy pequeños. Esto es debido a la aportación residual del temporizador de guardia a la detección. Los mecanismos de ciclo de reintento y procesador de repuesto recuperan la mayoría de los errores. El mecanismo de ciclo de reintento (columna *R_BOFF*) responde bien ante fallos con menor tiempo de actividad, mientras que para tiempos de actividad mayores el procesador de repuesto (columna *R_SP*) recupera la mayor parte de errores. La conclusión es la misma que se ha obtenido en la sección 3.4.3.1 para los buses: el sistema interpreta que el fallo es permanente.

3.4.5 Respuesta ante fallos en el procesador activo y en los buses del sistema

Los siguientes experimentos de inyección tienen como objetivos la lógica combinacional del procesador principal y sus elementos de almacenamiento internos, así como los buses del sistema. Se excluye de este análisis a la memoria externa, ya que se espera (y los resultados lo confirman, como se ve a continuación) que el comportamiento sea notablemente diferente. Como se ha constatado en la sección anterior, muy pocos de los fallos inyectados en la memoria externa se activan, mientras que el resto de los elementos tienen un porcentaje de activación considerablemente superior, por lo que mezclarlos puede desvirtuar los resultados. No se inyectan fallos en el procesador de repuesto ya que permanece inactivo mientras el sistema no está reconfigurado.

Los parámetros de los experimentos son los siguientes:

Sistema bajo estudio: MARK2 tolerante a fallos.

Cargas de trabajo (implica experimentos distintos): serie aritmética y *bubblesort*.

Punto de inyección: lógica combinacional y elementos de almacenamiento del procesador principal, y buses del sistema (datos, direcciones y control).

Técnica de inyección: *Órdenes del simulador*.

Simulaciones por experimento: 1000.

Multiplicidad espacial (implica experimentos distintos):

- Fallos simples.
- Fallos múltiples en el espacio (adyacentes y no adyacentes).

Modelos de fallo:

- Para elementos de almacenamiento: *intermittent stuck-at (0, 1)*.
- Para buses: *intermittent pulse*.
- Para lógica combinacional: *intermittent {pulse, open, stuck-at, indetermination}*.

Tiempo de actividad (t_A): generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Tiempo de inactividad (t_I): generado mediante una distribución Uniforme en los rangos [0.01T, 0.1T], [0.1T, 1.0T] y [1.0T, 10.0T], donde T es el ciclo de reloj de la CPU. Cada rango supone un experimento distinto.

Longitud de la ráfaga (L_{Burst}): generada mediante una distribución Uniforme en el rango [1, 10].

Parámetros calculados: todos los definidos en la sección 3.4.2.

La figura 3.22 muestra gráficamente los porcentajes de errores activados (valor *Act*). Como ya se ha comprobado en anteriores experimentos, los fallos múltiples tienen mucha mayor influencia que los fallos simples, con valores próximos al 100%. Esto se cumple para ambas cargas de trabajo, con pequeñas diferencias entre sus valores. Es interesante remarcar que los porcentajes de errores activados son, en general, menores que los obtenidos al inyectar sólo en los buses del sistema, lo que indica que los buses son el elemento del sistema más sensible ante la aparición de fallos intermitentes.

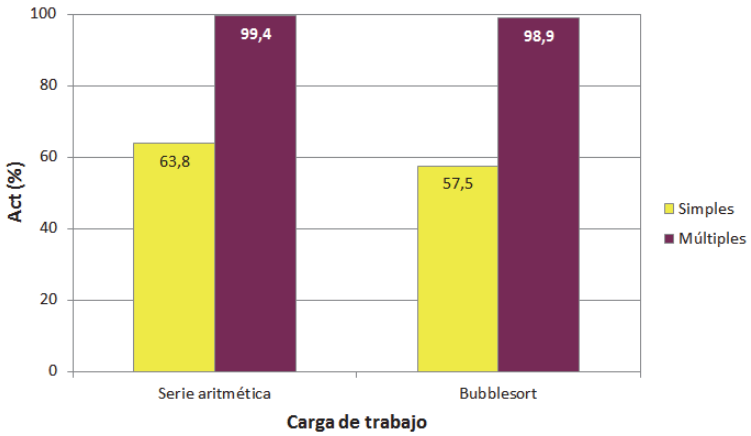

Figura 3.22: Porcentajes de errores activados.

La figura 3.23 muestra los porcentajes de detección de errores, o cobertura de detección (*Cd*). Las columnas están agrupadas por pares. En cada par, la columna de la izquierda representa el porcentaje de errores detectados gracias a los mecanismos de detección (valor *Detect*), mientras que la columna de la derecha representa tanto los errores detectados por los mecanismos como los no efectivos debido a la redundancia intrínseca del sistema (suma de los valores *Detect* y *No_Effect*). Se puede observar que los valores en esta columna son muy elevados, por encima del 90%. Esto indica que se producen pocas averías a causa de errores no detectados, lo que evidencia que el proceso de detección de errores funciona bastante bien. Evidentemente, los fallos simples son más fácilmente enmascarables por la redundancia intrínseca del sistema que los fallos múltiples. También es obvio que se detectan más fallos múltiples que simples, pues al afectar a más localizaciones es más fácil detectarlos.

La figura 3.24 muestra cómo se reparte la cobertura de detección de los errores entre los distintos mecanismos. Lo más remarcable es que los porcentajes de errores detectados por el mecanismo de paridad son mucho mayores que los detectados por el temporizador de guardia.

En la figura 3.25 se pueden observar los porcentajes de recuperación de errores, o cobertura de recuperación (*Cr*). En cada par de columnas, la de la izquierda representa el porcentaje de errores recuperados gracias a los mecanismos de recuperación (valor *Recov*), mientras que la de la derecha representa los errores que no terminan en avería, es decir, incluye tanto los errores recuperados por los mecanismos como los no efectivos a causa de la redundancia intrínseca del sistema (suma de los valores *Recov* y *No_Effect*). Como se puede observar, la diferencia entre ambas columnas es más pronunciada para

los fallos simples, lo que indica que éstos son más fácilmente enmascarables por el sistema.

Evidentemente, y comparando con la figura 3.23, se aprecia que el porcentaje de errores recuperados es menor que el porcentaje de errores detectados, ya que una parte de los errores detectados no pueden ser corregidos. La mayoría de ellos provoca una avería, y una pequeña parte se enmascara por la redundancia intrínseca del sistema.

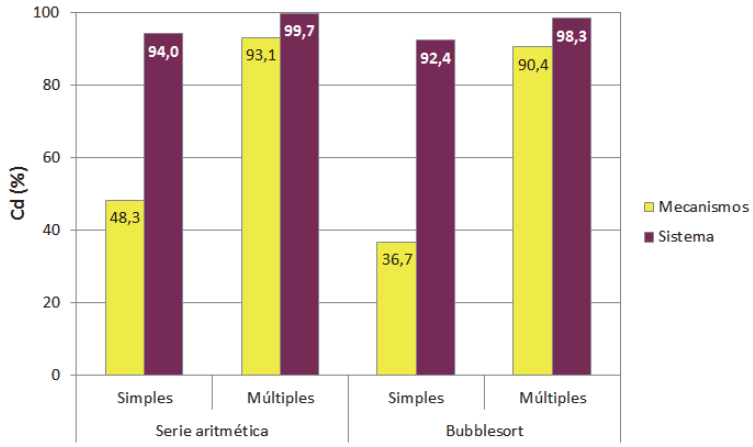

Figura 3.23: Porcentajes de detección de errores.

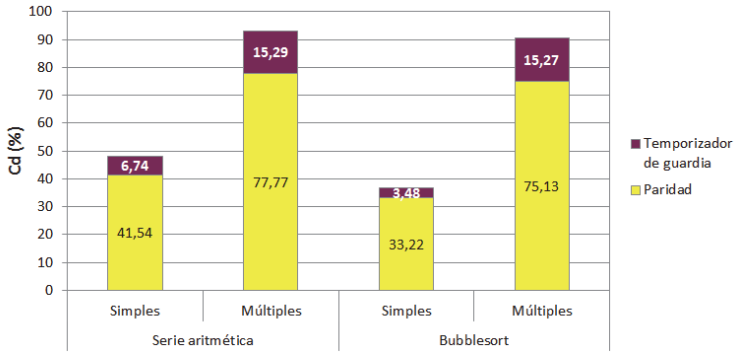

Figura 3.24: Contribución a la cobertura de detección de los diferentes mecanismos.

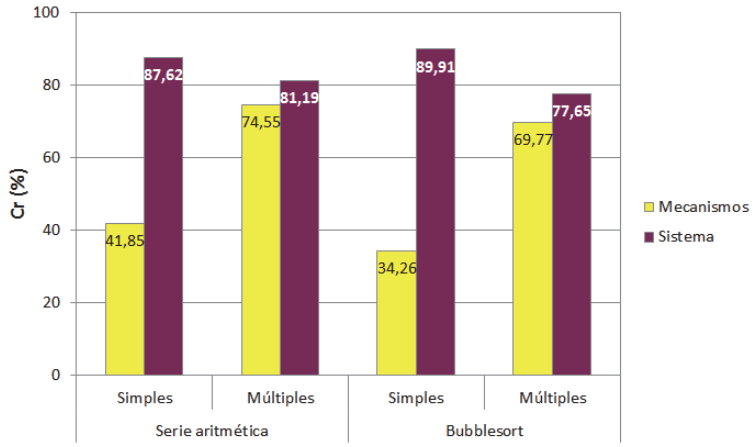

Figura 3.25: Porcentajes de recuperación de errores.

En la figura 3.25 también se puede observar que el porcentaje de recuperación, por parte de los mecanismos de tolerancia a fallos, es mayor para fallos múltiples que para fallos simples. Aunque los fallos múltiples son más difíciles de recuperar una vez son detectados, se detectan en mayor medida. Sin embargo, si observamos el porcentaje de los errores recuperados por el sistema en su conjunto, vemos que es justo al contrario. El motivo es que los fallos simples se pueden enmascarar más fácilmente, aumentando el porcentaje de errores no efectivos.

Es importante resaltar que el porcentaje de averías provocadas por los errores detectados y no recuperados no es despreciable, especialmente para fallos múltiples (con valores alrededor del 20%). La conclusión es que el proceso de recuperación no funciona tan bien como el de detección.

La figura 3.26 muestra cómo se reparte la recuperación de errores entre los distintos mecanismos. Como se puede observar, la recuperación se reparte casi íntegramente entre el ciclo de reintento y el procesador de repuesto. La recuperación por puntos de control tiene una incidencia mínima en el sistema, ya que está relacionada con la detección mediante temporizador de guardia, que tiene unos porcentajes pequeños de detección. La activación del procesador de repuesto es especialmente elevada en fallos múltiples, ya que son más dañinos y el procesador los interpreta como permanentes.

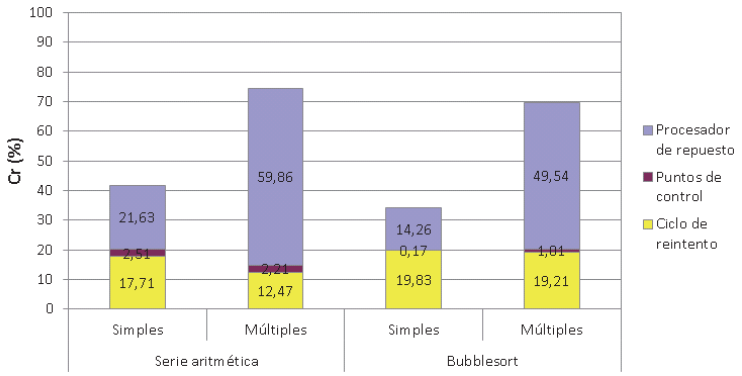

Figura 3.26: Contribución a la cobertura de recuperación de los diferentes mecanismos.

3.4.6 Comparación con fallos transitorios y permanentes

Finalmente, se han comparado los efectos de los fallos intermitentes con los fallos transitorios y permanentes. Para esta comparación se han realizado nuevos experimentos, inyectando fallos transitorios (modelo *bit-flip* en la memoria externa y modelo *pulse*, con una duración en el rango [0.1T, 1.0T], en buses), y permanentes (modelos *stuck-at (0, 1)*, *indetermination* y *open-line*). Para la comparación se han utilizado los resultados de las inyecciones realizadas en la sección 3.4.3.1, ya que los parámetros han sido los mismos, a excepción de los modelos de fallo utilizados en cada caso. En concreto, para la comparación se han utilizado los resultados de las inyecciones de fallos intermitentes, con tiempo de actividad y tiempo de inactividad en el rango [0.1T, 1.0T]. Como ejemplo de los resultados obtenidos, en la tabla 3.11 se muestra la comparación para fallos simples, y en la tabla 3.12 la comparación para fallos múltiples. En ambos casos se ha utilizado la carga de trabajo que ejecuta el algoritmo *bubblesort*.

Centrándonos en los datos obtenidos en las inyecciones en los buses, en la tabla se puede observar que el porcentaje de errores detectados (columna *Detect*) sigue la tendencia *Detect*(permanentes) > *Detect*(intermitentes) > *Detect*(transitorios). Es decir, los fallos transitorios son los más difíciles de detectar, mientras que los permanentes se detectan casi en su totalidad. El porcentaje de averías provocadas por fallos no detectados es muy bajo en todos los casos. La paridad es el mecanismo de detección que más errores

detecta. El temporizador de guardia tiene una aportación significativa en la detección de fallos simples, pero residual para fallos múltiples, salvo para los transitorios.

Tabla 3.11. Comparación de la respuesta del sistema ante fallos simples transitorios, intermitentes y permanentes (*bubblesort*)

Inyecciones en buses											
Fallos	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Transitorios	84,90	43,70	54,42	1,88	71,86	28,14	78,79	19,05	47,53	2,47	50,00
Intermitentes	94,80	13,92	84,07	2,00	82,81	17,19	75,41	23,96	41,10	3,16	55,74
Permanentes	100,00	13,20	84,30	2,50	81,49	18,51	13,05	86,24	0,00	5,45	94,55
Inyecciones en memoria											
Fallos	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Transitorios	2,70	0,00	100,00	0,00	100,00	0,00	11,11	88,89	66,67	0,00	33,33
Intermitentes	0,40	25,00	75,00	0,00	100,00	0,00	66,67	33,33	100,00	0,00	0,00
Permanentes	1,00	0,00	100,00	0,00	100,00	0,00	10,00	90,00	100,00	0,00	0,00

Tabla 3.12. Comparación de la respuesta del sistema ante fallos múltiples transitorios, intermitentes y permanentes (*bubblesort*)

Inyecciones en buses											
Fallos	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Transitorios	98,50	15,53	83,25	1,22	76,34	23,66	91,46	7,68	23,73	4,00	72,27
Intermitentes	99,20	3,73	95,97	0,30	97,16	2,84	88,66	11,13	19,67	2,49	77,84
Permanentes	99,80	1,40	98,20	0,40	95,71	4,29	2,14	97,76	0,00	0,00	100,00
Inyecciones en memoria											
Fallos	**Act**	**No_Effect**	**Detect**	**ND_Fail**	**D_PAR**	**D_WD**	**Recov**	**NR_Fail**	**R_BOFF**	**R_CP**	**R_SP**
Transitorios	4,20	2,38	95,24	2,38	100,00	0,00	15,00	85,00	50,00	0,00	50,00
Intermitentes	0,50	0,00	100,00	0,00	100,00	0,00	100,00	0,00	60,00	0,00	40,00
Permanentes	4,00	0,00	100,00	0,00	100,00	0,00	25,00	75,00	50,00	0,00	50,00

Respecto al porcentaje de errores recuperados (columna *Recov*) en los buses, la tendencia seguida es *Recov*(transitorios) > *Recov*(intermitentes) ≫ *Recov*(permanentes). Esto indica que, aunque los fallos transitorios son difíciles de detectar, una vez detectados son más fácilmente recuperables; los fallos permanentes, por el contrario, son más fáciles de detectar pero difícilmente recuperables. Esto es debido a que, aunque el sistema sea dual, los buses son comunes, por lo que un fallo permanente en éstos afecta al sistema independientemente del procesador que esté en funcionamiento (sea el principal o el de repuesto), invalidando el efecto de este mecanismo de recuperación. Esto se ve claramente en el porcentaje de averías provocadas por errores detectados y no recuperados (columna *NR_Fail*), que para fallos permanentes tiene valores muy superiores al resto de fallos. En esa misma columna, los fallos intermitentes tienen valores no despreciables, mientras que los fallos transitorios tienen los menores valores. En cuanto a los mecanismos de recuperación, el que más aporta en todos los casos es el

procesador de repuesto, seguido del ciclo de recuperación (con una importante contribución en fallos intermitentes y transitorios), mientras que la recuperación mediante puntos de control es mínima.

Los datos obtenidos para memorias ofrecen tendencias similares. No obstante, dado el bajo porcentaje de errores activados en todos los casos, pero especialmente para fallos intermitentes, aparecen algunas distorsiones. Como ya se ha comentado en la sección 3.4.4, esto es debido a que las cargas de trabajo utilizadas ocupan una porción pequeña respecto del total de la capacidad de la memoria.

3.5 Resumen y conclusiones

En este capítulo se ha hecho un exhaustivo estudio de las causas y efectos de los fallos intermitentes en los sistemas informáticos. En primer lugar se han analizado los mecanismos físicos que pueden provocar los fallos intermitentes, y se han determinado modelos de fallo representativos. Tras este análisis, se han obtenido interesantes conclusiones. Las causas que producen los fallos intermitentes tienen tanto una componente *hardware* (*hardware* defectuoso, variaciones en el proceso de fabricación, desgaste) como una componente ambiental (carga de trabajo, variaciones en la temperatura, la tensión de alimentación o la frecuencia de funcionamiento). Se espera un aumento en la tasa de fallos intermitentes, y que éstos no siempre terminen en fallos permanentes, por lo que adquieren entidad propia y deben dejar de considerarse como el preludio de un fallo permanente.

A partir de este estudio se han propuesto diversos modelos de fallos intermitentes, representativos de lo que sucede en la realidad en los niveles de abstracción de puertas lógicas y de transferencia de registros. Estos modelos de fallo se puedan incorporar en modelos de simulación de sistemas digitales para inyectar fallos. A la hora de parametrizar dichos modelos, hay que tener en cuenta que éstos se activan y desactivan de forma repetitiva y no determinista en el mismo punto, y que generalmente aparecen agrupados en ráfagas (grupos con varias activaciones y desactivaciones seguidas) separadas entre ellas por un (relativamente) largo tiempo de no activación.

Con los modelos obtenidos se han llevado a cabo experimentos de inyección para estudiar los efectos de los fallos intermitentes en sistemas no tolerantes a fallos. Se ha usado la herramienta VFIT para inyectar fallos intermitentes en los modelos en VHDL del microcontrolador 8051, con arquitectura CISC, y el Plasma, un microprocesador segmentado con arquitectura RISC. Se ha estudiado la influencia de distintos parámetros que afectan a los fallos intermitentes: características de una ráfaga (tiempos de actividad e

inactividad y longitud de la ráfaga), multiplicidad espacial, lugar de la inyección y frecuencia de trabajo del sistema.

La principal conclusión obtenida es que los parámetros que más influyen en los efectos de los fallos intermitentes son el tiempo de actividad, la longitud de la ráfaga y la multiplicidad espacial (a tener en cuenta, ya que la complejidad de los procesos de fabricación en las nuevas tecnologías submicrométricas tiende a favorecer la aparición de fallos múltiples intermitentes). Por el contrario, el tiempo de inactividad parece no tener influencia en los resultados. También tienen su incidencia el lugar de inyección (los buses del sistema se han mostrado como puntos enormemente sensibles a la aparición de los fallos intermitentes) y la frecuencia de trabajo del sistema (a mayor frecuencia de trabajo, mayor es la probabilidad de avería tras la aparición de un fallo intermitente, debido a que la probabilidad de capturar un fallo en flancos activos de los componentes síncronos aumenta). Es importante considerar esto último, dado que la variación de la frecuencia de trabajo es una de las tendencias actuales en los sistemas VLSI.

También se han comparado los efectos de los fallos intermitentes con los de los fallos transitorios y permanentes. Como cabía esperar, se ha observado que los fallos intermitentes provocan, en general, un mayor porcentaje de averías que los fallos transitorios, pero menor que los fallos permanentes.

Finalmente, se ha analizado la influencia del sistema bajo estudio, para ver si las distintas tendencias se generalizan. Aunque los valores difieran, las tendencias observadas son las mismas, con algunos matices explicables por las características propias de cada sistema (distinta capacidad de memoria y diferente arquitectura). La principal discrepancia estriba en el mayor impacto de los fallos intermitentes en la lógica combinacional del procesador RISC segmentado, debido a que su ruta de datos se basa principalmente en multiplexores y otros elementos de lógica combinacional, y no sólo en los buses.

Por todo lo expuesto hasta ahora, se puede concluir que son necesarias técnicas de tolerancia a fallos para mitigar los efectos de los fallos intermitentes. El siguiente paso ha sido estudiar algunas de las técnicas de tolerancia a fallos usadas habitualmente, para ver si son adecuadas para los fallos intermitentes. Nuevamente se ha usado VFIT para inyectar fallos intermitentes en un microprocesador tolerante a fallos basado en el MARK2.

Tras analizar los resultados obtenidos, se puede concluir que, aunque los mecanismos de detección de fallos estudiados funcionan adecuadamente, los de recuperación no son todo lo eficaces que deberían ante la aparición de fallos intermitentes. Esto es debido, principalmente, a que dichos mecanismos están orientados, bien para fallos transitorios, o bien para permanentes, mientras que los fallos intermitentes tienen características específicas que los diferencian de los otros dos tipos. Por ello, se abre una línea de investigación de técnicas de tolerancia a fallos específicas para fallos intermitentes, presentándose en el capítulo 4 una nueva propuesta.

4

Flexible Unequal Error Control y otros códigos correctores de errores

4.1 Introducción

En el capítulo 3 se ha demostrado la importancia de los fallos intermitentes y lo dañinos que pueden ser. También se ha visto que las técnicas de tolerancia a fallos existentes son capaces de detectar los errores generados por fallos intermitentes en un porcentaje elevado, pero los mecanismos de recuperación no son, en general, todo lo eficaces que cabría esperar. En este capítulo se presenta una herramienta que permite diseñar sistemas informáticos con mecanismos de tolerancia a fallos que respondan adecuadamente ante la aparición de fallos intermitentes.

Una de las cuestiones a tener en cuenta en los mecanismos de tolerancia a fallos es su latencia, o tiempo transcurrido desde que se produce un error hasta que se recupera el estado correcto del sistema. En el caso de fallos intermitentes, que provoca distintas activaciones en el mismo punto, la detección puede ser consecuencia de las primeras activaciones. Si el mecanismo de recuperación no es lo suficientemente rápido, el proceso de recuperación puede verse afectado por una nueva activación, provocando una avería.

Al mismo tiempo, buena parte de los lugares donde pueden aparecer fallos no son señales individuales, sino que se agrupan en bloques que representan información: buses del sistema, registros del procesador y memoria. Además, se ha demostrado que estos lugares son especialmente sensibles a los fallos intermitentes. Los resultados presentados en el capítulo 3 así lo indican para buses y registros del procesador. Aun en el caso de la memoria, donde nuestros datos ofrecían menores porcentajes de averías, se ha justificado que esto es así porque la carga de trabajo utilizada para las simulaciones utiliza una porción muy reducida de la memoria, por lo que la mayoría de fallos inyectados se manifiestan como errores latentes. En un entorno real, donde buena parte de la memoria esté ocupada, el impacto de los fallos en memoria es más elevado [Schroeder09].

El hecho de poder manejar la información en bloque y no como señales individuales permite utilizar técnicas de redundancia en la información. Los códigos detectores de errores y los códigos correctores de errores (ECC), como se ha explicado en el capítulo 2, utilizan bits adicionales (redundantes) para proteger la información y permitir la detección y, en su caso, la corrección de determinados tipos de errores. Además, si la codificación y la decodificación se implementan utilizando circuitos *hardware*, estos procesos pueden ser muy rápidos, por lo que la latencia de detección y recuperación puede ser baja.

En la arquitectura de los sistemas informáticos, los ECC se utilizan principalmente en memoria, bien sea para detectar errores (códigos de paridad), o para corregir distintos tipos de errores (son típicos los códigos SEC o SEC-DED, o los códigos para tolerar el fallo denominado *chipkill* [Dell97]). En cuanto a los buses, habitualmente no se protegen. Recientemente se está trabajando en evitar en los buses situaciones de diafonía, con los denominados *Crosstalk Avoidance Codes* (CAC) [Sridhara05]. Los registros del sistema tampoco se suelen proteger.

Esta tesis presenta los códigos denominados *Flexible Unequal Error Control* (FUEC) *codes* (códigos flexibles para control desigual de errores). Estos códigos tienen una serie de características que los hacen especialmente útiles para tolerar fallos intermitentes, sobre todo si éstos afectan a más de un bit. La principal cualidad de estos códigos es la posibilidad de aplicar control desigual a los distintos bits de una palabra. Esto es especialmente interesante para situaciones en las que se produce una tasa de error variable en el espacio, es decir, que no todos los bits de una palabra tienen la misma tasa de error, como sucede con los fallos intermitentes.

Es importante remarcar que los códigos, por sí mismos, no pueden resolver los problemas generados por los fallos intermitentes, ya que no tienen "memoria", es decir, no pueden adaptarse a la evolución temporal de las condiciones de fallo. El motivo es que un código codifica y decodifica considerando las entradas que tiene en un instante de tiempo determinado, como si de una "foto fija" se tratase. Dado que los fallos

intermitentes provocan variabilidad en la tasa de fallo, tanto espacial (distintas tasas de fallo para distintos bits de la misma palabra) como temporal (antes y después de la aparición del fallo intermitente, o incluso dentro de una ráfaga y en el tiempo entre ráfagas), es necesario un mecanismo de tolerancia a fallos que incluya ambas vertientes. Los códigos propuestos en esta tesis pueden gestionar adecuadamente la variabilidad espacial de la tasa de fallo, pero se requiere de circuitería adicional que controle la evolución temporal, es decir, que detecte cuándo y dónde se producen fallos intermitentes y adapte adecuadamente el comportamiento del sistema. Aunque el diseño de este mecanismo adicional queda fuera del alcance de esta tesis, en la sección 4.4 se aborda esta cuestión.

La metodología de búsqueda de códigos, descrita en este capítulo, permite diseñar otros códigos. Aunque el hallazgo más importante son los códigos FUEC, ya que su diseño permite explotar la principal característica de los fallos intermitentes (la variabilidad en la tasa de error), se ha trabajado con otros códigos. En concreto, se ha abordado la aparición de fallos múltiples adyacentes, cada vez más frecuentes en circuitos VLSI, y que pueden manifestarse como fallos transitorios, permanentes o como fallos intermitentes, como se justificó en el capítulo 3.

En el resto del capítulo, en primer lugar se detallan las características de los códigos FUEC, y su necesidad dadas las carencias de los códigos existentes para tolerar fallos intermitentes. A continuación se describe en detalle el proceso de diseño de los códigos en función de dichas características, y se presentan algunos ejemplos de diseño, comparándolos con otros códigos existentes. Posteriormente se afronta el problema de los mecanismos adaptativos de tolerancia a fallos, que pueden utilizar los códigos FUEC para tolerar los fallos intermitentes. Finalmente, se presentan otros códigos para corregir o detectar fallos múltiples adyacentes.

4.2 Descripción de los códigos Flexible Unequal Error Control

Algunos códigos correctores de errores ya existentes tienen algunas de las características necesarias para una respuesta adecuada de un sistema informático ante la aparición de los fallos intermitentes. Sin embargo, ninguno de ellos tiene todas esas características. A continuación se hace una pequeña discusión acerca de algunos de los códigos que más se aproximan a lo requerido, detallando sus ventajas y sus carencias, para a continuación centrarnos en describir en detalle los códigos FUEC.

4.2.1 Justificación

Aunque es evidente, para comenzar conviene recordar que los códigos correctores de errores son capaces de proteger bloques de información de varios bits. Este es el primero de nuestros requisitos. Además, en general se pueden construir sus codificadores y decodificadores mediante circuitos *hardware*. No obstante, ello no garantiza que el proceso de decodificación sea suficientemente rápido. Hay que tener en cuenta que los elementos que se pretende proteger son los registros del procesador, los buses internos y la memoria, por lo que la decodificación ha de ser muy rápida para no perjudicar en exceso a las prestaciones del sistema. Por tanto, se descartan códigos que necesiten decodificadores secuenciales e iterativos. Hay que centrarse en decodificadores combinacionales y códigos sencillos.

También es necesario alcanzar un equilibrio entre redundancia y cobertura. La redundancia indica la relación de los dígitos añadidos, necesarios para la codificación, respecto a los dígitos de información. La cobertura representa los tipos de errores corregidos y/o detectados. Una cobertura demasiado baja (no detectar/corregir errores que se pueden producir con relativa frecuencia) hace que el sistema esté demasiado expuesto a las averías, con lo que no se consigue el objetivo de que sea tolerante a fallos. Una cobertura excesiva (detectar/corregir errores que no se van a producir) requiere una mayor redundancia, lo que implica mayor complejidad en la circuitería, incremento en el área de silicio necesaria, mayor consumo y, a su vez, mayor riesgo de nuevos fallos.

Los fallos más habituales son los que provocan errores simples (es decir, errores de un bit). Para corregir este tipo de errores existen códigos SEC (del inglés *Single Error Correction*), como los códigos de Hamming [Hamming50]. Estos códigos garantizan que su redundancia es la mínima necesaria para corregir errores de un bit. También son típicos los códigos SEC-DED (del inglés *Single Error Correction - Double Error Detection*), como los Hamming extendidos o los códigos de Hsiao [Hsiao70], capaces de corregir errores simples y detectar errores dobles. Los decodificadores para estos códigos son sencillos de implementar. No obstante, tal como se ha justificado en el capítulo 3, cada vez son más frecuentes los fallos múltiples (que afectan a varios bits), especialmente los fallos adyacentes. Los errores múltiples son difíciles de corregir, especialmente si son aleatorios (la posición de los bits afectados no sigue un patrón determinado, por ejemplo de vecindad) y, aunque existen códigos capaces de corregirlos, éstos requieren una redundancia elevada. Por ejemplo, el código binario (23, 12) de Golay [Golay49] es capaz de corregir errores aleatorios de tres bits.

Los fallos múltiples adyacentes pueden provocar lo que se conoce como errores en ráfaga. Una ráfaga de longitud l es un error que se extiende a lo largo de l bits dentro

de una palabra y donde, al menos, el primer y el último bit están en error. Es importante no confundir este concepto de ráfaga (que implica una dimensión espacial) con la idea de activaciones y desactivaciones de un fallo intermitente, tal como se han visto en el capítulo 3, y que implica una dimensión temporal.

Hay distintos códigos capaces de corregir errores en ráfaga, como los Reed-Solomon [Reed60] o los códigos Reiger [Reiger60]. Su principal problema es la elevada redundancia. Más recientes son los códigos *Error-Locality-Aware* (ELA) [Shamshiri10], que pueden corregir tanto errores en ráfaga como errores independientes con una baja redundancia y decodificadores combinacionales. También se puede utilizar la técnica del entrelazado para corregir errores en ráfaga, tanto en memorias [Reviriego10] como en buses del sistema [Saiz13a]. Sin embargo, estos códigos no se adaptan a situaciones en las que haya distintas tasas de error en los bits de una palabra. Los fallos intermitentes provocan que la tasa de error del (de los) bit(s) afectado(s) sea mayor que la del resto de bits, por lo que sería interesante disponer de códigos que permitan el control asimétrico de errores. Los códigos *Unequal Error Control* (UEC) [Fujiwara06] sí permiten el control asimétrico de errores, permitiendo aplicar dos funciones de control distintas en dos partes diferenciadas de la palabra. Sin embargo, esas funciones de control son bastante limitadas, especialmente en la zona con menor control, no permitiendo la corrección de errores en ráfaga.

Por tanto, se puede concluir que ningún código de los vistos tiene todas las características necesarias para tolerar fallos intermitentes en buses o elementos de almacenamiento de un procesador. Con el objetivo de cubrir ese hueco se han desarrollado los códigos *Flexible Unequal Error Control* (FUEC) [Saiz13b], cuyas características se detallan a continuación.

4.2.2 Características de los códigos FUEC

Los códigos *Flexible Unequal Error Control* (FUEC) son códigos lineales binarios, con codificadores y decodificadores simples y fácilmente implementables en *hardware*, por lo que su latencia es baja. También aseguran, si se realiza adecuadamente el proceso de búsqueda de los códigos, que la redundancia es la mínima necesaria para la cobertura seleccionada. Esta cobertura se puede elegir de forma flexible, acorde a los requerimientos de diseño, combinando errores simples, múltiples aleatorios o en ráfaga.

Además, como indica su nombre, tienen capacidad para aplicar control asimétrico. A diferencia de los códigos UEC, el control asimétrico no tiene por qué ser únicamente a dos niveles. Es decir, los códigos FUEC permiten dividir la palabra en el número de áreas

requerido, y aplicar en cada área la función de control necesaria. En este punto es importante comentar la diferencia entre códigos UEC y códigos *Unequal Error Protection* (UEP) [Masnick67]. Los códigos UEP sólo tienen una función de control, pero hay unos bits más protegidos que otros (distintos niveles de protección). Es decir, se admiten decodificaciones no correctas, siempre que sean "suficientemente" buenas: si los bits más protegidos son correctos, la decodificación puede ser admisible aunque haya errores en los bits menos protegidos. Aunque los códigos UEP sí pueden ofrecer protección a distintos niveles, su comportamiento no se ajusta al objetivo de mantener la integridad de la información. Una comparación más detalla de las características y diferencias de los códigos UEC y UEP se puede consultar en [Saiz13b].

Otra de las diferencias fundamentales entre los códigos FUEC y los códigos UEC es la posibilidad de aplicar una mayor variedad de funciones en cada área. En el área más protegida, los códigos UEC existentes pueden corregir todos los errores, o bien corregir errores en ráfaga, con una longitud máxima determinada. La corrección de todos los errores puede ser innecesaria y requiere mucha redundancia. La corrección de errores en ráfaga, aunque requiere menos redundancia, no cubre otros patrones de error típicos, como los errores dobles donde los bits afectados están a mayor distancia que la longitud máxima de la ráfaga. En el área menos protegida, los códigos UEC existentes solo permiten la corrección de errores simples, y en ocasiones la detección de errores dobles. Como se ha comentado, esta protección puede resultar, en ocasiones, insuficiente incluso en el área menos protegida. Los códigos FUEC, que se proponen en esta tesis, son capaces de aplicar en cada área el nivel de protección requerido, combinando la detección y/o corrección de errores simples, múltiples en ráfaga y/o múltiples aleatorios.

Para una mejor comprensión, a continuación se presenta un ejemplo de diseño sencillo para entender mejor las características de los códigos FUEC, e introducir la metodología a seguir para obtener códigos a partir de los requisitos del sistema.

4.2.3 Metodología de diseño de los códigos FUEC

A continuación se describe la metodología de diseño de códigos FUEC a partir de unas hipótesis de fallo determinadas. Para ello, consideremos un ejemplo de diseño sencillo, que incluye tres áreas diferenciadas con distintas capacidades de detección y corrección. Esto favorecerá la comprensión de la metodología, y permitirá apreciar las características únicas de estos códigos, que los diferencian de otros códigos existentes y, especialmente, de los códigos UEC.

Sea una palabra de datos de 12 bits, en la que se pueden distinguir tres áreas de 4 bits, como se puede apreciar en la figura 4.1. Aunque el ejemplo sea bastante limitado, la metodología se puede aplicar a un mayor número de bits, como se verá posteriormente.

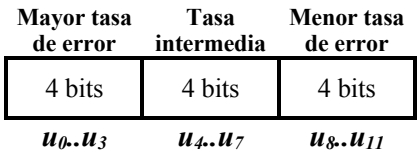

Figura 4.1: Disposición de los bits con distinta tasa de error en la palabra de datos.

Las hipótesis de fallo consideradas son las siguientes:

- En el área más fuertemente controlada se deben corregir todos los errores de un bit, y los errores de dos bits adyacentes (ráfaga de dos bits); además, se deben detectar todos los errores aleatorios de dos bits, y los errores en ráfaga de tres bits.
- En el área con control intermedio se deben corregir todos los errores de un bit, y los errores de dos bits adyacentes, sin detección adicional.
- En el área con menor control se deben corregir todos los errores de un bit, y detectar los errores de dos bits adyacentes.

Estas hipótesis de fallo determinan los vectores de errores a corregir y detectar, tal como se describe a continuación.

4.2.3.1 Descripción de la metodología

Para diseñar un código, en primer lugar es necesario determinar una serie de parámetros: el tamaño de la palabra de datos (*k*), el tamaño de la palabra codificada (*n*), el conjunto de vectores de error que se deben corregir $(\boldsymbol{E}_+)$ y, en su caso, el conjunto de vectores de error que se deben detectar aunque no se corrijan $(\boldsymbol{E}_\Delta)$. En el caso de los códigos FUEC, estos conjuntos dependen, a su vez, de las distintas áreas de control, sus límites (es decir, el primer y el último bit de cada área), y el nivel de control que se aplica en cada área.

Con estos parámetros, es posible encontrar una matriz de paridad **H**, con dimensiones $(n-k)\times n$ que, si existe, define un código que es capaz de corregir y detectar los errores seleccionados. El proceso se realiza comprobando todas las posibles matrices. Aunque la herramienta de búsqueda se describe con más detalle en el apartado 4.2.3.2, en este momento merece la pena comentar que el algoritmo va generando matrices parciales, lo que permite descartar rápidamente las combinaciones intermedias que no conducen a una solución válida. No es, por tanto, una búsqueda basada únicamente en "fuerza bruta".

Para encontrar la matriz de paridad, es importante entender la decodificación mediante síndrome. Tal como se ha explicado en el capítulo 2, el síndrome **s** se define como $\mathbf{s}^T=\mathbf{H}\cdot\mathbf{r}^T$, donde **r** es la palabra codificada recibida. Como se puede observar, el síndrome depende exclusivamente del vector de error (**e**) que afecte a la palabra original durante la transmisión:

$$\mathbf{s}^T = \mathbf{H}\cdot\mathbf{r}^T = \mathbf{H}\cdot(\mathbf{b}\oplus\mathbf{e})^T = \mathbf{H}\cdot\mathbf{b}^T \oplus \mathbf{H}\cdot\mathbf{e}^T = \mathbf{H}\cdot\mathbf{e}^T$$

En la expresión, **b** es la palabra codificada, y se debe cumplir que $\mathbf{H}\cdot\mathbf{b}^T=\mathbf{0}$. Por tanto, para poder determinar que se ha producido un error concreto, es necesario que cada error tenga asociado un síndrome único. Si el síndrome es cero, se estima que el vector de error también es cero, lo que indica una transmisión libre de errores, y que la palabra recibida es correcta. En caso contrario, la decodificación se realiza mediante una tabla que relaciona los síndromes con su vector de error asociado (*syndrome lookup table*). Si el síndrome aparece en esta tabla, el error se puede identificar y corregir; si el síndrome es distinto de cero pero no aparece relacionado con ningún error en concreto, el error ocurrido se puede detectar, pero no corregir. Por tanto, la matriz de paridad buscada debe cumplir las siguientes condiciones:

$$\mathbf{H}\cdot\mathbf{e}_i^T \neq \mathbf{H}\cdot\mathbf{e}_j^T; \forall \mathbf{e}_i, \mathbf{e}_j \in \boldsymbol{E}_+ \mid \mathbf{e}_i \neq \mathbf{e}_j \qquad (4.1)$$

$$\mathbf{H}\cdot\mathbf{e}_i^T \neq \mathbf{H}\cdot\mathbf{e}_j^T; \forall \mathbf{e}_i \in \boldsymbol{E}_\Delta, \mathbf{e}_j \in \boldsymbol{E}_+ \qquad (4.2)$$

La condición 4.1 indica que cada error que se desee corregir ha de producir un síndrome distinto. La condición 4.2 expresa que cada error que se tenga que detectar tiene que generar un síndrome distinto a cualquiera de los producidos por los errores corregibles.

La búsqueda de la matriz de paridad se puede considerar un problema de satisfacibilidad booleana (*Boolean satisfiability*, SAT)[3]. Anteriores propuestas para resolver este problema ([Dutta07], [Shamshiri10]) se centran en aplicaciones específicas. La propuesta presentada en esta tesis es más general: en tres pasos sucesivos, esta metodología es capaz de encontrar cualquier código binario lineal, si existe, en función de los vectores de error que se requiera corregir o detectar. El primer paso es determinar los conjuntos de vectores de error a corregir y/o detectar. A continuación se realiza la búsqueda de la matriz de paridad que cumpla las condiciones expresadas en las ecuaciones 4.1 y 4.2. Finalmente, dado que se pueden dar varias soluciones, se puede seleccionar una de ellas considerando distintos criterios.

En las siguientes secciones se aplican cada uno de estos pasos al diseño del código FUEC para el ejemplo propuesto anteriormente.

DETERMINAR LOS CONJUNTOS DE VECTORES DE ERRORES

En el ejemplo propuesto ya conocemos que el valor de k es 12, pero todavía no se ha determinado el valor de n. Se puede determinar aproximadamente el número de bits de paridad necesarios ($n-k$) en función del número de síndromes necesarios, ya que se debe cumplir que $|\boldsymbol{E}_+| \leq 2^{n-k}$. Es decir, tiene que haber, al menos, tantos síndromes como errores a corregir. En todo caso, hay que tener en cuenta que esta condición es necesaria pero no suficiente, por lo que encontrar el número óptimo de bits de paridad es, a veces, un proceso de prueba y error.

Así pues, el conjunto de errores a corregir en el ejemplo propuesto es $\boldsymbol{E}_+ = \boldsymbol{E}_0 \cup \boldsymbol{E}_1 \cup \boldsymbol{E}_{B2}^{0..7}$. Se incluye el vector de no error $(\boldsymbol{E}_0)$, todos los errores de un bit $(\boldsymbol{E}_1)$, ya que hay que corregirlos en las tres áreas, y todos los errores de dos bits adyacentes, o ráfagas de dos bits, en las dos áreas más protegidas $(\boldsymbol{E}_{B2}^{0..7})$. Por tanto, el valor de n debe cumplir la condición $|\boldsymbol{E}_+| = |\boldsymbol{E}_0| + |\boldsymbol{E}_1| + |\boldsymbol{E}_{B2}^{0..7}| = 1 + n + 7 \leq 2^{n-12}$. De esta expresión se obtiene que $n \geq 17$; esto es, son necesarios al menos 5 bits redundantes o bits de paridad. En el caso de que $n=17$, $|\boldsymbol{E}_+| = 25$ y $2^{n-k} = 32$. La

[3] Dícese del problema de determinar si existe una interpretación semántica que satisfaga determinada fórmula booleana. En otras palabras, dada una fórmula booleana hay que determinar si, dando todos los posibles valores a los términos de la fórmula, alguna de las combinaciones hace que el resultado de evaluar la fórmula sea TRUE. Por ejemplo, $a \wedge \neg b$ es satisfacible (para a=TRUE y b=FALSE), pero $a \wedge \neg a$ no es satisfacible.

disposición de la palabra codificada, incluyendo los bits redundantes, se muestra en la figura 4.2, y los vectores que representan los errores a corregir se incluyen en la tabla 4.1.

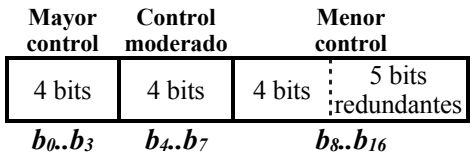

Figura 4.2: Disposición de los bits con distinto nivel de control en la palabra codificada.

Tabla 4.1. Vectores que representan los errores a corregir

Errores a corregir (E_+)	Subconjunto de errores
(0000000000000000)	No error (E_0)
(1000000000000000) (0100000000000000) (0010000000000000) (0001000000000000) (0000100000000000) (0000010000000000) (0000001000000000) (0000000100000000) (0000000010000000) (0000000001000000) (0000000000100000) (0000000000010000) (0000000000001000) (0000000000000100) (0000000000000010) (0000000000000001)	Errores de un bit (E_1)
(1100000000000000) (0110000000000000) (0011000000000000) (0001100000000000) (0000110000000000) (0000011000000000) (0000001100000000)	Errores en ráfaga de 2 bits en las áreas con mayor control ($E_{B2}^{0..7}$)

Es importante remarcar que el hecho de incluir los bits redundantes en el área menos controlada no es un requisito. La metodología permite posicionarlos en cualquier área, pero hay que tener en cuenta que el número de vectores de errores a considerar puede incrementarse, y el número de bits redundantes probablemente también. En definitiva, la posición de los bits redundantes dependerá de las especificaciones de diseño.

Una vez definido el conjunto de vectores de errores a corregir, hay que determinar el conjunto de vectores de errores a detectar. Tal como se han descrito las hipótesis de fallo, $E_\Delta = (E_2^{0..3} - E_{B2}^{0..3}) \cup E_{B3}^{0..3} \cup E_{B2}^{7..16}$. La explicación de cada subconjunto y los vectores que incluye se puede observar en la tabla 4.2. Los siete (32−25) síndromes no utilizados para la corrección de errores, se emplearán para la detección de estos 16 errores.

Tabla 4.2. Vectores que representan los errores a detectar

Errores a detectar (E_Δ)	Subconjunto de errores
(1010000000000000) *(1001000000000000)* *(0101000000000000)*	Errores aleatorios de 2 bits en el área más protegida, excluyendo los que se corrigen ($E_2^{0..3} - E_{B2}^{0..3}$)
(1x100000000000000) (01x10000000000000)	Errores en ráfaga de 3 bits en el área más protegida ($E_{B3}^{0..3}$)
(0000000110000000) (0000000011000000) (0000000001100000) (0000000000110000) (0000000000011000) (0000000000001100) (0000000000000110) (0000000000000011)	Errores en ráfaga de 2 bits en el área con menor protección ($E_{B2}^{7..16}$)

CALCULAR LA MATRIZ DE PARIDAD

Los valores de k y n, y los subconjuntos determinados en el paso anterior son los parámetros de entrada a una aplicación desarrollada como producto de esta tesis, y que permite calcular la matriz de paridad que satisfaga, con esos vectores de error, las condiciones expresadas anteriormente en las ecuaciones 4.1 y 4.2.

Dada la complejidad de esta aplicación y lo extenso de su explicación, se abordará en detalle en la sección 4.2.3.2. En este momento, solamente se va a comentar el tratamiento del resultado. Una vez finalizado el algoritmo de búsqueda, todas las posibles soluciones se encuentran en un conjunto de soluciones. Si el conjunto de soluciones está vacío, se puede afirmar con seguridad que el código buscado, en función de los parámetros n y k, y de los vectores de error a corregir y detectar, no existe. Hay que tener en cuenta que, aunque la condición $|E_+| \leq 2^{n-k}$ se satisfaga, no se garantiza la existencia del código. Esta situación solo se puede solventar de dos maneras: o bien se incrementa la redundancia (aumentando el valor de n) o reduciendo el control de errores (es decir, considerar menos vectores de error a corregir o detectar).

Si hay varias soluciones, la elección de la mejor opción se puede hacer considerando distintos criterios. Estos son algunos de ellos:

- Primera solución encontrada: esta opción permite reducir el tiempo de búsqueda.
- Menor peso Hamming de la matriz de paridad: el peso Hamming indica el número de 1s que tiene una matriz o un vector binario. Un menor peso Hamming en la matriz de paridad permite, por ejemplo, reducir el número de puertas lógicas utilizadas en una implementación *hardware*.

- Menor peso Hamming de la fila con mayor peso de la matriz: en una implementación *hardware*, el nivel lógico (número de puertas de la ruta más larga) de cada generador de los bits de paridad o de síndrome depende del peso Hamming de la fila asociada. Por tanto, la fila con mayor peso determina el nivel global del circuito codificador y decodificador, y por tanto su velocidad.

Es necesario remarcar que se podrían aplicar otros criterios dependiendo, por ejemplo, de la tecnología utilizada y de los requerimientos de codificación y decodificación.

También es importante reseñar que no es necesario esperar al final de la ejecución de la aplicación para obtener resultados. Durante la ejecución se pueden ir filtrando las soluciones con respecto al criterio seleccionado, para ofrecer la mejor solución encontrada hasta el momento. Como se verá en la sección 4.2.3.2, debido al elevado número de combinaciones que hay que comprobar, el tiempo requerido para una ejecución completa es elevado y, salvo para valores pequeños de k, inabordable. Por ello, poder obtener soluciones subóptimas es fundamental.

A continuación se muestra una solución obtenida para el ejemplo propuesto. En este caso, el criterio utilizado para seleccionar la solución ha sido obtener la matriz de paridad con menor peso Hamming.

IMPLEMENTACIÓN DEL CÓDIGO EN FUNCIÓN DE SU MATRIZ DE PARIDAD

Una solución óptima, resultado de ejecutar la aplicación con los parámetros anteriormente determinados y el criterio de selección indicado, es la siguiente:

b/r_0	b/r_1	b/r_2	b/r_3	b/r_4	b/r_5	b/r_6	b/r_7	b/r_8	b/r_9	b/r_{10}	b/r_{11}	b/r_{12}	b/r_{13}	b/r_{14}	b/r_{15}	b/r_{16}
u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	Ver ecuaciones (4.3)				
1	0	1	0	0	1	0	0	1	1	1	1	1	0	0	0	0
0	1	0	1	0	0	1	0	1	1	0	0	0	1	0	0	0
0	0	1	0	1	0	1	1	1	0	1	0	0	0	1	0	0
1	0	0	1	1	0	1	0	0	1	0	1	0	0	0	1	0
0	1	0	0	1	1	0	1	0	0	1	1	0	0	0	0	1

Las fórmulas para la codificación y la decodificación se obtienen fácilmente a partir de la matriz de paridad. En este caso, la palabra de datos (**u**) es parte de la palabra codificada (**b**), y los bits de paridad se sitúan en las columnas con sólo un 1. Cada bit de paridad se calcula haciendo la operación XOR (o exclusiva) de los bits que tienen 1 en la fila correspondiente. Por ejemplo, el bit b_{13} de la palabra codificada es un bit de paridad, porque su columna solo tiene un 1. Si tomamos la fila en que aparece ese 1, encontramos otros 1s en las columnas correspondientes a los bits u_1, u_3, u_6, u_8 y u_9. Haciendo la

operación XOR de todos ellos, se calcula el bit de paridad. De la misma forma se calculan el resto de bits de paridad.

El síndrome se calcula de forma similar a partir de la palabra a decodificar (**r**). Cada fila genera un bit de síndrome haciendo la operación XOR de todos los bits cuyas posiciones tengan un 1 en esa fila. Tanto las fórmulas de los bits de paridad (para la codificación) como los del síndrome (para la decodificación) se presentan en las ecuaciones 4.3.

$$
\begin{array}{ll}
b_i = u_i, \forall i \in \mathrm{N} : 0 \le i \le 11 & \\
b_{12} = u_0 \oplus u_2 \oplus u_5 \oplus u_8 \oplus u_9 \oplus u_{10} \oplus u_{11} & s_0 = r_0 \oplus r_2 \oplus r_5 \oplus r_8 \oplus r_9 \oplus r_{10} \oplus r_{11} \oplus r_{12} \\
b_{13} = u_1 \oplus u_3 \oplus u_6 \oplus u_8 \oplus u_9 & s_1 = r_1 \oplus r_3 \oplus r_6 \oplus r_8 \oplus r_9 \oplus r_{13} \\
b_{14} = u_2 \oplus u_4 \oplus u_6 \oplus u_7 \oplus u_8 \oplus u_{10} & s_2 = r_2 \oplus r_4 \oplus r_6 \oplus r_7 \oplus r_8 \oplus r_{10} \oplus r_{14} \\
b_{15} = u_0 \oplus u_3 \oplus u_4 \oplus u_6 \oplus u_9 \oplus u_{11} & s_3 = r_0 \oplus r_3 \oplus r_4 \oplus r_6 \oplus r_9 \oplus r_{11} \oplus r_{15} \\
b_{16} = u_1 \oplus u_4 \oplus u_5 \oplus u_7 \oplus u_{10} \oplus u_{11} & s_4 = r_1 \oplus r_4 \oplus r_5 \oplus r_7 \oplus r_{10} \oplus r_{11} \oplus r_{16}
\end{array}
\qquad (4.3)
$$

Los bits de síndrome determinan el siguiente paso. Si son todos 0, se asume que la palabra a decodificar es correcta. En caso contrario, se busca en la tabla de síndromes si el síndrome está asociado a un error a corregir, y si es así se corrige. En caso de no estar en la tabla, se detecta el error pero no se puede corregir. La tabla 4.3 muestra la tabla que relaciona los síndromes con el (los) bit(s) erróneos.

Tabla 4.3. Tabla que asocia los síndromes con errores (*syndrome lookup table*)

$s_4\ s_3\ s_2\ s_1\ s_0$	Error en…
0 0 0 0 0	No error
0 0 0 0 1	bit r_{12}
0 0 0 1 0	bit r_{13}
0 0 0 1 1	*Detección*
0 0 1 0 0	bit r_{14}
0 0 1 0 1	bit r_2
0 0 1 1 0	*Detección*
0 0 1 1 1	bit r_8

$s_4\ s_3\ s_2\ s_1\ s_0$	Error en…
0 1 0 0 0	bit r_{15}
0 1 0 0 1	bit r_0
0 1 0 1 0	bit r_3
0 1 0 1 1	bit r_9
0 1 1 0 0	*Detección*
0 1 1 0 1	bits r_4, r_5
0 1 1 1 0	bit r_6
0 1 1 1 1	bits r_2, r_3

$s_4\ s_3\ s_2\ s_1\ s_0$	Error en…
1 0 0 0 0	bit r_{16}
1 0 0 0 1	bit r_5
1 0 0 1 0	bit r_1
1 0 0 1 1	*Detección*
1 0 1 0 0	bit r_7
1 0 1 0 1	bit r_{10}
1 0 1 1 0	bits r_3, r_4
1 0 1 1 1	bits r_1, r_2

$s_4\ s_3\ s_2\ s_1\ s_0$	Error en…
1 1 0 0 0	*Detección*
1 1 0 0 1	bit r_{11}
1 1 0 1 0	bits r_6, r_7
1 1 0 1 1	bits r_0, r_1
1 1 1 0 0	bit r_4
1 1 1 0 1	*Detección*
1 1 1 1 0	*Detección*
1 1 1 1 1	bits r_5, r_6

Todas las operaciones necesarias para la codificación y la decodificación son fácilmente implementables. La codificación es tan simple como un árbol de operaciones XOR (o un circuito o algoritmo equivalente) por cada bit de paridad. El síndrome se calcula del mismo modo. La corrección y la detección se pueden implementar usando un decodificador binario de 5 entradas a 32 salidas y algunas puertas lógicas.

Este código se ha simulado para probar el correcto funcionamiento de la codificación y la decodificación, y que su comportamiento se ajuste a lo esperado. Los

resultados confirman que todos los vectores de errores en $\boldsymbol{E}_{+}$ se corrigen satisfactoriamente, y todos los vectores de errores de $\boldsymbol{E}_{\Delta}$ se detectan.

4.2.3.2 La aplicación de búsqueda de códigos

Como se ha comentado anteriormente, los parámetros de entrada a la aplicación son los valores de k y n y los conjuntos de vectores de los errores a corregir y detectar. La aplicación hace una búsqueda de la matriz de paridad que satisfaga, con esos vectores de error, las condiciones expresadas en las ecuaciones 4.1 y 4.2.

La búsqueda de esa matriz, entre todas las posibles matrices binarias con dimensiones $(n-k)\times n$, requiere un esfuerzo computacional importante, ya que las posibles combinaciones son $2^{((n-k)\times n)}$. Con el objetivo de reducir el coste computacional, el algoritmo empieza con una matriz vacía, va añadiendo columnas para generar una matriz parcial, y va comprobando los vectores de error cuyo síndrome se puede calcular teniendo en cuenta esa matriz parcial. Si todos los vectores de error tratables con la matriz parcial cumplen las ecuaciones 4.1 y 4.2 (es decir, que los errores corregibles generan todos síndromes distintos, y los errores detectables generan síndromes distintos a los generados por los errores corregibles), la matriz parcial es correcta y puede ser parte de una solución, con lo que se añade una nueva columna. En caso contrario, la matriz parcial se descarta, ya que no puede formar parte de una solución válida.

La clave de este proceso es el algoritmo de comprobación de la matriz parcial, cuyo pseudocódigo se presenta en la figura 4.3. Como se puede observar, el algoritmo es recursivo y permite comprobar todas las posibles matrices aplicando la técnica de *backtracking* o vuelta atrás. También aplica la técnica de *branch and bound* (ramificación y poda), lo que permite descartar las matrices parciales que no pueden ser parte de una solución válida.

Los parámetros pasados al algoritmo son la matriz de paridad parcial y sus dimensiones (en concreto, el número de columnas, que irá cambiando). El proceso se inicia con una matriz vacía, a la que se le van añadiendo columnas, pero el número de filas se mantiene (siempre son $n-k$, el número de bits de síndrome).

El algoritmo comienza con un conjunto de síndromes vacío. A continuación, para cada uno de los vectores de error a corregir se comprueba si con la matriz parcial se puede calcular su síndrome. Esto será así si todos los 1s del vector de error se encuentran en las primeras *ncols* columnas. Como se ha explicado anteriormente, el peso Hamming indica el número de 1s en un vector. Por tanto, si el vector de error y el vector de error

parcial (considerando solo los primeros *ncols* bits) tienen el mismo peso, se cumple la condición buscada y se puede calcular su síndrome. Una vez calculado, se comprueba si está en el conjunto de síndromes. Si es así, un vector de error anterior ya había generado el mismo síndrome, por lo que la matriz parcial no podrá ser parte de una solución válida (no se cumple la ecuación 4.1) y se sale del procedimiento. Si el síndrome calculado no está en el conjunto de síndromes, se añade.

Si se llega al final del tratamiento de todos los vectores de error a corregir, el conjunto de síndromes tendrá todos los síndromes que corrigen errores. A continuación se hace un proceso similar para los vectores de error a detectar. Si un error a detectar genera el mismo síndrome que alguno de los ya calculados anteriormente para los errores a corregir, no se cumple la ecuación 4.2 y, por tanto, no puede aportar una solución válida y se sale del procedimiento. La única diferencia entre el tratamiento de los vectores a corregir y a detectar es que los síndromes calculados en este último caso no se incluyen en el conjunto de síndromes. De esta forma, varios errores detectables pueden generar el mismo síndrome.

```
Procedimiento ComprobarMatrizParcial H_parcial (n-k) × ncols   /* ncols ∈ [1..n] */
    ConjuntoDeSíndromes = {}
    Para cada vector de error e en E+
        e_parcial = (e1, e2, ..., encols)
        Si PesoHamming(e) = PesoHamming(e_parcial)
            nuevoSíndrome = CalculaSíndrome(H_parcial × Transpuesta(e_parcial))
            Si nuevoSíndrome en ConjuntoDeSíndromes entonces SalirProc      /* Matriz parcial no válida */
            Sino Añadir nuevoSíndrome a ConjuntoDeSíndromes
        Fin si
    Fin para
    Para cada vector de error e en EΔ
        e_parcial = (e1, e2, ..., encols)
        Si PesoHamming(e) = PesoHamming(e_parcial)
            nuevoSíndrome = CalculaSíndrome(H_parcial × Transpuesta(e_parcial))
            Si nuevoSíndrome en ConjuntoDeSíndromes entonces SalirProc      /* Matriz parcial no válida */
            Sino Hacer Nada   /* nuevoSíndrome no se almacena en este caso */
        Fin si
    Fin para
    Si ncols = n
        Añadir H_parcial a ConjuntoDeSoluciones
        SalirProc
    Sino
        Para cada nueva_columna posible          /* n-k bits, excluyendo la columna 0: 2^(n-k)-1 valores posibles */
            ComprobarMatrizParcial [H_parcial | nueva_columna] (n-k) × (ncols+1)
        Fin para
    Fin si
Fin procedimiento
```

Figura 4.3: Procedimiento de comprobación de matrices parciales.

Si se llega al final del tratamiento de los vectores de error a detectar, se puede afirmar que se cumplen las ecuaciones 4.1 y 4.2 para todos los errores tratables (es decir, con todos sus bits en error en las primeras *ncols* columnas). Si el procedimiento ha sido llamado con una matriz completa (con n columnas) se ha encontrado una solución válida, que se añade al conjunto de soluciones, y se sale del procedimiento. En caso de que la matriz sea parcial, se añade una nueva columna, se le asignan todos los posibles valores y se llama de nuevo al procedimiento de comprobación.

Una vez finalizado el algoritmo, todas las posibles soluciones se encuentran en el conjunto de soluciones. Si hay varias soluciones, la selección de la mejor se puede hacer considerando distintos criterios, tal como se ha comentado anteriormente. Si el conjunto de soluciones está vacío, el código buscado no existe.

COSTE COMPUTACIONAL Y TIEMPO DE EJECUCIÓN

Como ya se ha comentado anteriormente, la búsqueda de la matriz de paridad requiere un esfuerzo computacional importante, ya que las posibles combinaciones son $2^{((n-k)\times n)}$. Afortunadamente, gracias al uso de las técnicas de *backtracking* o vuelta atrás y *branch and bound* (ramificación y poda) muchas de estas combinaciones son descartadas en etapas tempranas de la búsqueda, lo que permite que el número real de combinaciones comprobadas sea muy inferior. Aun así, desafortunadamente, el número de combinaciones a comprobar es tan elevado que una ejecución completa del algoritmo es inabordable, salvo para valores pequeños de n.

No es sencillo determinar el orden del coste computacional, como ocurre habitualmente con los algoritmos de *backtracking*, ya que el coste depende del número de nodos del árbol de búsqueda que se visitan para conseguir todas las posibles soluciones (en nuestro caso, todas las posibles matrices parciales). En general, este coste es exponencial.

Por tanto, el coste computacional no depende exclusivamente del algoritmo, ni de los valores de n y k, sino que también depende de los vectores de errores a corregir o detectar, ya que éstos determinan las matrices parciales que son válidas o se descartan. No sólo de su número, sino de los patrones de error que se consideren en cada búsqueda.

En cuanto al tiempo de ejecución, éste dependerá de la implementación concreta y de la máquina sobre la que se ejecute, además de los parámetros anteriormente considerados. Nuestra implementación del algoritmo está escrita en C, y se ejecuta en equipos con sistema operativo Windows™. La mayoría de las matrices se han buscado en una máquina equipada con un procesador Intel® CORE™ i5 con una frecuencia nominal de 2.5 GHz (frecuencia pico de 3.1 GHz). En esta máquina, el código FUEC (17, 12) calculado como ejemplo se obtiene de manera casi inmediata, y la ejecución

completa tarda 5 segundos (manteniendo fijas las posiciones de los bits de paridad). En otros ejemplos que se muestran posteriormente se dan algunas indicaciones sobre el tiempo de ejecución de cada uno de ellos. No obstante, para hacerse una idea de la velocidad y cantidad de comprobaciones a realizar, es interesante resaltar que, en general, se verifican por encima del millón de matrices parciales por segundo. En el ejemplo anterior se comprueban alrededor de 2 millones de matrices por segundo. Este dato, evidentemente, depende del código que se esté buscando, llegando en algunos casos a ser superior a los 6 millones de comprobaciones por segundo.

De todas maneras, no es necesario completar la ejecución para obtener resultados aceptables. Si no es necesario optimizar (o una solución sub-óptima es suficiente) y el código existe, normalmente se encuentra una matriz válida en los primeros segundos, minutos u horas (nuevamente, esto dependerá de los valores de n y k, de los vectores a corregir o detectar, etc.) Para encontrar soluciones sub-óptimas, puede ser interesante alargar la ejecución durante un tiempo mayor (la búsqueda de algunos de los ejemplos incluidos en este capítulo se ha mantenido en marcha durante una semana, aproximadamente). Únicamente si se desea confirmar la no existencia de un código, o si queremos encontrar la solución óptima para un determinado criterio de optimización, es necesario completar la ejecución del algoritmo.

Como se ha comentado en el apartado 4.2.3.1, cuando no existe un código es necesario reducir los vectores a corregir o detectar, o aumentar la redundancia. Esta última alternativa también puede resultar útil cuando el algoritmo de búsqueda no encuentra una solución, porque no existe o porque tarda demasiado en encontrarla. Efectivamente, aumentar la redundancia puede ayudar a reducir el tiempo de búsqueda, ya que al aumentar el número de síndromes disponibles también se puede aumentar la cantidad de matrices válidas para el código buscado.

Aunque se ha trabajado mucho en mejorar la implementación del algoritmo, y la versión actual es muy eficiente comparada con las primeras implementaciones utilizadas, es necesario seguir mejorándola. Por ejemplo, dado que las filas de una matriz de paridad de un código se pueden cambiar de orden y la matriz resultante sigue siendo válida, esta característica se podría aprovechar para reducir el número de combinaciones a comprobar. O también, dado que las búsquedas pueden ser largas, convendría evitar la pérdida de trabajo por cortes del suministro eléctrico, bloqueos del sistema, etc. Aunque ya se ha trabajado este aspecto, es mejorable.

No obstante, dado que el propio algoritmo de búsqueda y su implementación son secuenciales, paralelizarlos permitiría búsquedas más complejas y resolverlas en menor tiempo. Éste es uno de los principales retos a abordar en un futuro inmediato.

4.3 Comparación con otros códigos

Los códigos FUEC tienen algunas características que no se pueden encontrar en otros códigos, especialmente la capacidad de utilizar distintas funciones de control de errores en distintas partes de la palabra, y aplicar la función deseada en cada parte. Esto dificulta la comparación con otros códigos existentes. En todo caso, en este apartado se intenta hacer una comparación con un código *Unequal Error Control* existente. Los códigos UEC son, por sus características, los más parecidos a los FUEC, como se ha comentado anteriormente en la sección 4.2.1.

En concreto, como ejercicio de comparación se propone un código UEC y, con la misma redundancia, se van a presentar dos códigos FUEC. Esto permite comparar las distintas capacidades de detección y corrección de los nuevos códigos propuestos.

Sea el código (72, 64) SEC-F7EC, un código UEC presentado en [Fujiwara06], cuya matriz es esta:

$$\mathbf{H} = \begin{bmatrix}
1000000 & 0000000000000001111110000000000000000000001111111111111111110000000 \\
0100000 & 0000000000111111000001000000000011111111110000000000111111010000000 \\
0010000 & 0000011110000100001000001111110000001111000000111100001100100000 \\
0001000 & 0001110001000100001000111000111000111000100011100010001010001000 \\
0000100 & 0110010010001000010001011011001011001001001100100100010010000100 \\
0000010 & 1010100100010000100001101101010101010010010101001000100000000100 \\
0000001 & 1101001000100001000001110110100110100100011010010001000000000010 \\
1111111 & 1111111111111111111110000000000000000000000000000000000000000001
\end{bmatrix}$$

Este código divide la palabra en dos áreas, una de siete bits donde se corrigen todos los posibles errores, y el resto de la palabra (65 bits, que incluyen los bits de paridad), donde se corrigen errores de un bit. Es decir, según la notación indicada anteriormente, el conjunto de vectores de error a corregir es $\boldsymbol{E}_{+} = \boldsymbol{E}_{*}^{0..6} \cup \boldsymbol{E}_{1}^{7..71}$ (donde $\boldsymbol{E}_{*}^{0..6}$ representa todos los posibles errores en la zona indicada). Como se ha comentado anteriormente, la corrección de todos los posibles errores en un área requiere mucha redundancia: en este caso, sólo para la corrección de los posibles errores en el área de 7 bits son necesarios 2^7 síndromes distintos. Además, salvo aplicaciones muy específicas, es muy improbable un error de hasta 7 bits. Por otra parte, la corrección de errores simples en el área menos controlada puede no ser suficiente en determinadas circunstancias.

Para evitar estos inconvenientes de los códigos UEC, y como ejemplo del potencial de los códigos FUEC, se propone un primer código (72, 64) [Saiz13b], también dividido en dos partes y del mismo tamaño que el código UEC. En este caso, en el área

más controlada, de 7 bits, se corrigen todos los errores de un bit, de dos bits, de tres bits y los errores en ráfaga de 4 bits. Esto reduce la redundancia necesaria y puede ser suficiente para la mayoría de aplicaciones. En el área menos controlada se corrigen los errores simples y los errores en ráfaga de dos bits, lo que puede ajustarse más a determinadas situaciones. Las hipótesis de error propuestas para este código pueden resultar más realistas, en general, que las utilizadas en el código UEC, aunque esto dependerá de la aplicación concreta en la que se vaya a utilizar cada código.

Por tanto, el conjunto de vectores de error a corregir por el código FUEC es $\boldsymbol{E}_+ = \boldsymbol{E}_0 \cup \boldsymbol{E}_1 \cup \boldsymbol{E}_{B2} \cup \boldsymbol{E}_2^{\boldsymbol{0..6}} \cup \boldsymbol{E}_3^{\boldsymbol{0..6}} \cup \boldsymbol{E}_{B4}^{\boldsymbol{0..6}}$. Utilizando la metodología propuesta, se obtiene la siguiente matriz de paridad:

$$\mathbf{H} = \left[\begin{array}{l}
1100000\ 0001000000010010111011000110101010110010110010000001001001000010 \\
1010000\ 0100000000010000100110110011010111010010000100011101010010010000 \\
0101000\ 0010010000010010001100101101010000000101010011010010001001001000 \\
0010100\ 0000000100100000100101010100001100100011111000100101010100001001001 \\
0001010\ 0000001001001010010000010001001010011000001001010001100101001011 0 \\
0000101\ 0010001010000100010001001000100001010000010110001010000010100110 1 \\
0000010\ 1000000010100001001010101000100001001010100000100101011011010111 0 \\
0000001\ 0000100000001001000000000010010100100101001010110110110110111101 1
\end{array}\right]$$

Este código ha sido implementado en *software* y se comprueba que cumple las especificaciones propuestas. También es importante remarcar que el codificador y el decodificador para este código tienen una complejidad similar a los del código UEC presentado anteriormente, por lo que el área de silicio (en una implementación *hardware*) y el tiempo de codificación y decodificación son similares.

Otro de los inconvenientes de los códigos UEC existentes es su limitación a la hora de aplicar más de dos funciones de control. La metodología utilizada para encontrar códigos FUEC permite hacerlo, como se ha visto anteriormente en el ejemplo básico de la sección 4.2.3. Para poder hacer una comparación de las funciones de corrección aplicadas en distintas áreas, a igualdad de redundancia, se ha diseñado otro código FUEC (72, 64) [Saiz13b] que divide la palabra codificada en tres partes, de 7, 20 y 45 bits. En la parte más protegida (7 bits) se corrigen errores de uno, de dos y de tres bits, y errores en ráfaga de cuatro y cinco bits de longitud. En la parte con protección intermedia (20 bits) se corrigen errores de un bit y errores en ráfaga de dos y tres bits. Finalmente, en la parte menos protegida (45 bits) se corrigen errores de un bit. Por tanto, el conjunto de vectores de error a corregir es $\boldsymbol{E}_+ = \boldsymbol{E}_0 \cup \boldsymbol{E}_1 \cup \boldsymbol{E}_2^{\boldsymbol{0..6}} \cup \boldsymbol{E}_3^{\boldsymbol{0..6}} \cup \boldsymbol{E}_{B4}^{\boldsymbol{0..6}} \cup \boldsymbol{E}_{B5}^{\boldsymbol{0..6}} \cup \boldsymbol{E}_{B2}^{\boldsymbol{7..26}} \cup \boldsymbol{E}_{B3}^{\boldsymbol{7..26}}$. La matriz de paridad obtenida en este caso es:

$$\mathbf{H} = \begin{bmatrix}
1100000\ 0110010010110110110\ 0 & 1000000000001101101001000101000100110100100 & 10 \\
1010000\ 1010110010101101001\ 0 & 0100000000010001010010000001000001010000100 & 0 \\
0101000\ 0010000101010011010\ 0 & 0010000010011010110010010010010010000001000 & 100 \\
0010100\ 0001010100110010011\ 0 & 0001000000010110001110001110001001011001000 & 00 \\
0001010\ 0000100011100101011\ 1 & 0000100001001110000001111001111000000111000 & 000 \\
0000101\ 0000001001101000110\ 1 & 0000010000000001111111111000000111000000111 & 01 \\
0000010\ 1001001001001010100\ 1 & 0000001000100000000000000111111111000000000 & 11 \\
0000001\ 0100100100100100110\ 0 & 0000000111100000000000000000000000011111111 & 111
\end{bmatrix}$$

Al igual que en el caso anterior, el código se ha implementado en *software* y se ha comprobado que cumple las especificaciones propuestas. Nuevamente, el codificador y el decodificador para este código tienen una complejidad similar a los del código UEC presentado.

La principal conclusión de esta comparativa es la flexibilidad que demuestran los códigos FUEC a la hora de aplicar las funciones de control de errores requeridas, acorde con las hipótesis de fallo específicas de un diseño, y permitiendo aplicar distintas funciones en cualquier número de áreas. Hasta donde el autor de la presente tesis tiene conocimiento, no hay ningún código que tenga esta capacidad.

4.4 Mecanismos adaptativos para tolerar fallos intermitentes

Como se ha comentado en la introducción de este capítulo, un código corrector de errores no puede, por sí mismo, adaptarse a la aparición de fallos intermitentes. Los códigos correctores de errores, en general, no tienen memoria de anteriores procesos de decodificación. Cada decodificación trabaja con la palabra codificada que le llega en un instante de tiempo determinado. Por tanto, el decodificador trabaja con una "foto fija", por sí mismo no tiene noción de los errores que se hayan podido producir anteriormente.

Los fallos intermitentes provocan variabilidad en la tasa de fallo, tanto espacial (distintas tasas de fallo para distintos bits de la misma palabra) como temporal (antes y después de la aparición del fallo intermitente, o incluso dentro de una ráfaga y en el tiempo entre ráfagas). Un código que en un momento dado pueda ser adecuado para las tasas de fallo de ese momento, puede no serlo en un instante de tiempo distinto.

Para un adecuado tratamiento de los fallos intermitentes mediante el uso de códigos correctores, por tanto, es necesario un mecanismo de tolerancia a fallos que incluya tanto la variabilidad temporal como la espacial. Con la utilización de los códigos

presentados en este capítulo se puede gestionar la variabilidad espacial. Para controlar la variabilidad temporal se tiene que utilizar algún tipo de mecanismo adaptativo y reconfigurable que haga cambiar el comportamiento del sistema para hacerle evolucionar según las necesidades de cada momento.

Aunque en este momento dicho mecanismo no está todavía implementado y queda fuera del ámbito de esta tesis, el autor (junto al grupo de investigación al que pertenece) está trabajando en un modelo sencillo que pueda servir como prueba de concepto de que un mecanismo de tolerancia a fallos intermitentes utilizando los códigos FUEC como parte integrante es posible.

Para esta prueba de concepto se está trabajando con el modelo en VHDL de un computador segmentado (en este caso, el Plasma, descrito en el apartado 3.3.2.2). Se está intentando proteger la memoria RAM contra fallos intermitentes. En una primera hipótesis de fallos, se asume que los errores que pueden aparecer originalmente son errores de un bit (por lo que hay que corregirlos), y muy raramente pueden aparecer errores dobles (solo se detectan). Por tanto, sobre el modelo original del Plasma se están haciendo las modificaciones necesarias para que los datos en memoria estén protegidos y codificados utilizando un código SEC-DED. Para ello, es necesario ampliar el tamaño de palabra de memoria (respecto del modelo no protegido), para incluir los bits de paridad necesarios.

También se asume que en un momento dado puede aparecer un fallo intermitente que afecte a un bit. A partir de ese momento, la tasa de error de ese bit aumenta considerablemente, por lo que podría incluso coincidir un error en ese bit con un error en cualquier otro bit producido por otra causa. Si se mantuviese el código anterior, el error doble que se puede producir se detectaría pero no se podría corregir, por lo que el dato original no sería recuperable, lo que causaría la avería del sistema. Si, en el momento en que se detecta que hay un fallo intermitente, se cambia el comportamiento del sistema para que los datos se codifiquen con un código que tenga en cuenta esta circunstancia, se podrían corregir adecuadamente más errores. El sistema debe disponer de dos codificadores y dos decodificadores, uno para el código original (SEC-DED) y otro para el nuevo código (un código FUEC que, además de corregir errores simples, corrija errores dobles si uno de los bits erróneos es el afectado por el fallo intermitente; en este caso, como la redundancia de ambos códigos es la misma, no es necesario modificar el tamaño de palabra ni la capacidad de la memoria). Para conseguir el nuevo comportamiento, hacen falta varios elementos, que se enumeran en los siguientes párrafos.

En primer lugar, hay que detectar el fallo intermitente. Para conseguirlo se pueden utilizar las señales de salida del decodificador original que corrigen los posibles bits erróneos, y un contador por cada bit. Cuando se produce un error en un bit, se

incrementa el contador asociado a ese bit, y se decrementa el de los demás si son distintos de cero. De esta forma, los errores generados por fallos transitorios incrementarán el contador un número reducido de veces (por lo general, una), que se descontará probablemente en un instante posterior del tiempo, cuando aparezca otro fallo en una posición distinta. Por el contrario, si aparece un fallo intermitente su contador irá subiendo de valor en lecturas sucesivas. Cuando dicho contador alcance un determinado umbral se asumirá que hay un fallo intermitente en ese bit. Para minimizar el riesgo de falsos positivos (asumir que hay un fallo intermitente cuando en realidad no lo hay) o falsos negativos (no detectar a tiempo un fallo intermitente), habrá que determinar el valor adecuado del umbral. También habrá que estudiar la posible aplicación de una ventana temporal, para no acumular errores muy separados en el tiempo. Este esquema de detección está todavía en una fase incipiente, y serán necesarias pruebas para comprobar su adecuado funcionamiento.

Una vez detectado que hay un fallo intermitente que afecta a un bit determinado, lo siguiente es recodificar todo el contenido de la memoria con el código nuevo. Este proceso de reconfiguración requiere que el *hardware* esté preparado para pasar del uso de un par codificador-decodificador al otro. Cada uno de los datos almacenados en memoria debe leerse, decodificarse con el decodificador original, codificar el dato obtenido con el nuevo codificador y volver a escribirlo en memoria. Como se ha comentado anteriormente, ambos códigos tienen la misma redundancia, con lo que el ancho de palabra de la memoria no varía. Este proceso puede resultar pesado, ya que ha de hacerse para todas las palabras de la memoria. Se puede relacionar con los mecanismos de refresco típicos de las memorias RAM dinámicas, o con mecanismos de *scrubbing* [Saleh90] también típicos de las memorias tolerantes a fallos. En el caso del modelo propuesto, se dotará a la memoria de un mecanismo de *scrubbing*.

Inicialmente, la posible detección y corrección de errores solamente se produce al leer un dato de memoria y decodificarlo, lo que puede producir que lleguen a acumularse varios errores en la misma palabra. El *scrubbing* fuerza periódicamente una lectura, decodificación, codificación y escritura de todos los datos de la memoria. De esta forma, se reduce el riesgo de acumulación de errores. La periodicidad y demás detalles de organización dependen de cada implementación, siempre intentando impactar lo mínimo posible las prestaciones. En el caso del diseño propuesto, se implementará un mecanismo de *scrubbing* que tenga un primer modo de funcionamiento clásico, utilizando el codificador y decodificador del código inicial, pero que tenga un modo de funcionamiento especial que sea capaz de realizar el proceso de recodificación en toda la memoria, leyendo y decodificando con el código inicial, pero recodificando y escribiendo con el nuevo código. A diferencia del funcionamiento clásico, en el que el proceso se puede hacer en distintas etapas dividiendo la memoria en varias partes, en este caso hay

que recodificar toda la memoria antes de que el procesador continúe trabajando. Finalmente, se volverá a un modo clásico de funcionamiento, pero con el código nuevo.

Esta prueba de concepto pretende demostrar que es posible adaptar el comportamiento del sistema ante la presencia de fallos intermitentes. Pero también es útil para comprobar los posibles inconvenientes. En primer lugar, hay que analizar si vale la pena toda la circuitería necesaria, en lugar de usar métodos estáticos, como códigos con mayor redundancia y, por tanto, mayor capacidad de corrección. En general, no valdrá la pena para registros individuales. Podría resultar interesante para los bloques de registros de propósito general, donde probablemente no sería necesario hacer *scrubbing*, dado que los datos almacenados suelen variar con mayor frecuencia. También puede resultar adecuado para bloques mayores de información, como la memoria, aunque con el esquema aquí descrito se asume que se presenta un único fallo intermitente en todo el bloque de datos, o el fallo afecta al mismo bit en todas las palabras (lo que no es infrecuente, si el fallo se produce en las interconexiones, por ejemplo).

Otro de los problemas es el tiempo necesario para reconfigurar y recodificar toda la memoria. Durante ese tiempo, el procesador debe dejar de ejecutar instrucciones, con lo que se pueden penalizar de forma importante las prestaciones. Se podría pensar en hacer este proceso por etapas, lo que complicaría el controlador de memoria y aumentaría el riesgo de que un fallo intermitente acabe en avería, pero reduciría el impacto en las prestaciones.

Una de las cuestiones que merecen especial atención es la siguiente: ¿el nuevo código es capaz de corregir los errores dobles, que incluyan al bit afectado por el fallo intermitente, independientemente de la posición de éste? La respuesta es no. Una alternativa poco práctica sería disponer de tantos códigos distintos como bits de la palabra codificada, y seleccionar el adecuado según el bit afectado. Esta alternativa podría ser válida en caso de implementar el mecanismo en un dispositivo reconfigurable, como una FPGA. En este caso los codificadores y decodificadores no estarían efectivamente implementados, sino que estarían en memoria y sería necesaria una reconfiguración para poder utilizar un código u otro. Otra alternativa, la utilizada en nuestro modelo, es tener un único código preparado para tratar los fallos intermitentes en un bit en concreto. En este caso, además, será necesaria circuitería adicional (principalmente registros de desplazamiento a la salida del codificador y del decodificador) para hacer coincidir el bit más controlado del código con el bit afectado por el fallo intermitente.

Finalmente, aunque en esta prueba de concepto no se ha considerado, habría que tener en cuenta la posibilidad de que un fallo intermitente desaparezca (o se desactive durante un largo período de tiempo), o que aparezca otro fallo intermitente en una posición distinta.

En resumen, se abre una línea de investigación que pretende obtener mecanismos de tolerancia a fallos intermitentes que usen los códigos propuestos en este capítulo, y que permitan adaptarse a la evolución temporal de la tasa de fallos. No es el objetivo de esta tesis, ya que esta línea de investigación está todavía en una fase incipiente. No obstante, en este apartado se han esbozado las primeras ideas. Avanzar en esta línea supone el principal reto del autor de esta tesis a corto plazo.

4.5 Otras aplicaciones de la metodología de generación de códigos

La metodología descrita en este capítulo no solamente permite generar códigos FUEC, sino que permite encontrar cualquier tipo de código que se desee, únicamente generando los conjuntos de vectores de errores a corregir y/o detectar. La única limitación, como se ha descrito anteriormente, viene impuesta por el tamaño de la palabra de datos, ya que para tamaños por encima de 64 bits el coste computacional y, por consiguiente, el tiempo de búsqueda, pueden ser demasiado elevados.

Aunque la principal característica de los fallos intermitentes es la variabilidad en la tasa de error (aprovechada en el diseño de los códigos FUEC), hay otras características importantes. En concreto, en los códigos presentados en este apartado se ha abordado la aparición de fallos múltiples adyacentes, cada vez más frecuentes en circuitos VLSI, y que pueden manifestarse como fallos intermitentes, como se justificó en el capítulo 3, aunque también como fallos transitorios o permanentes. Los fallos múltiples adyacentes se manifiestan como errores en ráfaga, y pueden ser interesantes códigos que sean capaces de detectarlos o corregirlos rápidamente, aunque sea de forma estática.

En primer lugar se presenta un trabajo donde se proponen códigos Hamming modificados para detectar errores en ráfaga de dos y tres bits en memorias de semiconductores [Saiz14]. A continuación se introduce otro trabajo donde se presentan códigos capaces de corregir errores en ráfaga de dos y tres bits, también en memorias, con poca redundancia y baja latencia de codificación y decodificación [Saiz15]. Uno de estos códigos ha sido implementado en VHDL y sintetizado en una FPGA como parte de la investigación llevada a cabo en el Instituto Nacional de Técnica Aeroespacial (INTA), dependiente del Ministerio de Defensa, Gobierno de España.

4.5.1 Códigos Hamming modificados para detectar errores en ráfaga en memorias

Los códigos Hamming [Hamming50] corrigen errores de un bit con una baja redundancia y codificadores y decodificadores sencillos y de baja latencia. Esto los hace atractivos de cara a su uso en memorias de semiconductores para proteger la información almacenada. Dado que el tamaño de palabra utilizado en las memorias es normalmente una potencia de dos (son típicos valores de 8, 16, 32 o 64 bits), los códigos Hamming utilizados son códigos no perfectos (excepto para 4, que sí genera un código perfecto; en ese caso, no se puede aplicar esta metodología). En un código no perfecto, no todos los síndromes que se pueden obtener en la decodificación están asignados a un error a corregir. Si se produce uno de esos síndromes, se puede afirmar que se ha producido un error, pero no es posible identificarlo. Por tanto, el error se detecta, pero no se corrige. El problema de los códigos Hamming originales es que esta detección no es sistemática, es decir, se detectan algunos errores de dos bits, algunos errores de tres bits, etc.

El objetivo del trabajo presentado en [Saiz14], y descrito en este apartado, es sistematizar la detección de errores en ráfagas cortas, es decir, obtener códigos Hamming modificados para aprovechar esos síndromes para detectar el 100% de los errores en ráfaga de dos y tres bits, que son los errores múltiples más frecuentes. Los códigos obtenidos para palabras de 32 y 64 bits logran este objetivo, mejorando los presentados en [Sanchez12] y [Sanchez14], que únicamente detectan el 100% de los errores en ráfaga de dos bits (también llamados errores dobles adyacentes).

A continuación se presentan los resultados y conclusiones obtenidos para los tamaños de palabra más frecuentes (8, 16, 32 y 64 bits).

4.5.1.1 Códigos para palabras de datos de 8 bits

El código Hamming (12, 8) tiene 4 bits de paridad, y por tanto 16 síndromes distintos. Como solo 13 de ellos están asignados (el síndrome cero indica que la palabra es correcta, y otros 12 síndromes son necesarios para los errores simples en cada uno de los bits de la palabra codificada), quedan 3 síndromes disponibles para detección. El código Hamming original solamente detecta uno de los once posibles errores dobles adyacentes, lo que representa el 9%. En [Sanchez12] se presenta un código Hamming modificado, con los mismos bits de paridad, capaz de detectar 9 de 11 (82%) de los errores dobles adyacentes. Su matriz de paridad es esta:

$$\mathbf{H} = \begin{bmatrix} 100100110011 \\ 001110100101 \\ 010010101010 \\ 010001010101 \end{bmatrix}$$

¿Sería posible encontrar un código (12, 8) capaz de detectar todos los errores dobles adyacentes, además de corregir los errores simples? Gracias a nuestra metodología, se puede asegurar que dicho código no existe. Para ello, se obtienen los vectores de errores a corregir y detectar, que se presentan en la tabla 4.4, y se ejecuta el algoritmo de búsqueda. En este caso, como el tamaño de la palabra es pequeño, se puede realizar una búsqueda exhaustiva y el algoritmo finaliza sin encontrar ninguna matriz, entre todas las posibles, que satisfaga los requerimientos. En nuestra implementación, la ejecución completa para esta búsqueda tarda tan solo 7 segundos.

También se ha intentado mejorar la propuesta de [Sanchez12], reseñada anteriormente, intentando encontrar un código (12, 8) que detectase 10 de los 11 posibles errores dobles adyacentes (91%), pero ese código tampoco existe. Así pues, la detección de errores dobles adyacentes de la propuesta citada no se puede mejorar.

Tabla 4.4. Vectores que representan los errores a corregir y detectar

Errores a corregir (E_+)	**Subconjunto de errores**
(000000000000)	No error ($\boldsymbol{E}_0$)
(100000000000) (010000000000) (001000000000) *(000100000000) (000010000000) (000001000000)* *(000000100000) (000000010000) (000000001000)* *(000000000100) (000000000010) (000000000001)*	Errores de un bit ($\boldsymbol{E}_1$)
Errores a detectar (E_Δ)	**Subconjunto de errores**
(110000000000) (011000000000) (001100000000) *(000110000000) (000011000000) (000001100000)* *(000000110000) (000000011000) (000000001100)* *(000000000110) (000000000011)*	Errores en ráfaga de 2 bits ($\boldsymbol{E}_{B2}$)

4.5.1.2 Códigos para palabras de datos de 16 bits

El código Hamming (21, 16) tiene 5 bits de paridad. Por tanto, hay 32 síndromes distintos, y solo están asignados 22 de ellos, por lo que quedan 10 síndromes disponibles para detección. El código Hamming original únicamente detecta el 5% de los errores dobles adyacentes (1 de 20). En [Sanchez14] se presenta un código Hamming modificado, con los mismos bits de paridad, que detecta todos los errores dobles adyacentes (es decir, el código es SEC-DAED). Su matriz de paridad es ésta:

$$\mathbf{H} = \begin{bmatrix} 010101010101010100001 \\ 111111001000000110101 \\ 111000110100001001010 \\ 100110110010010111000 \\ 001011100100110100110 \end{bmatrix}$$

Utilizando nuestra metodología se ha obtenido una matriz de paridad distinta, con dos pequeñas mejoras respecto de la propuesta anterior. La primera de ellas es que la matriz de paridad está representada en forma sistemática, es decir, las columnas correspondientes a los bits de paridad están agrupadas en un extremo de la matriz, y las columnas correspondientes a los bits de datos están agrupadas en el otro extremo. Se puede observar en la matriz obtenida, que se muestra a continuación, como las 5 columnas de la izquierda forman una matriz identidad. Esto normalmente simplifica las operaciones de codificación y decodificación. Esta es la matriz obtenida:

$$\mathbf{H} = \begin{bmatrix} 100001101001110011001 \\ 010000010101101101011 \\ 001001000010011101101 \\ 000100110001010110111 \\ 000010001110101010110 \end{bmatrix}$$

La segunda mejora es que la nueva matriz de paridad tiene menor peso Hamming (número de 1s) que la propuesta anterior (48 frente a 49). Aunque sea una mejora pequeña, esto se suele traducir en menos puertas lógicas (por ejemplo, si se usan puertas XOR de dos entradas) al implementar el codificador y el cálculo del síndrome en el decodificador. También tiene el peso Hamming más equilibrado entre filas, teniendo la fila más pesada menos peso que la más pesada de la propuesta anterior (11 frente a 10). Esto normalmente conduce a circuitos más rápidos.

Finalmente, comentar que la implementación del algoritmo de búsqueda, desarrollada por el autor de esta tesis, encontró la matriz presentada casi inmediatamente (en menos de un segundo de ejecución).

4.5.1.3 Códigos para palabras de datos de 32 bits

El código Hamming (38, 32) tiene 64 síndromes distintos (6 bits de paridad) y únicamente necesita 39 para corregir errores o indicar que la palabra es correcta. Quedan, por tanto, 25 síndromes disponibles para detección. En [Sanchez14] se propone un código (38, 32) SEC-DAED, mediante eliminación selectiva y reordenación de las columnas de la matriz.

Como puede observarse, el número de síndromes disponibles para detección aumenta conforme lo va haciendo el tamaño de palabra. ¿Sería posible aprovechar ese incremento para encontrar un código capaz de mejorar su capacidad de detección? La metodología presentada en esta tesis permite responder a esta pregunta. Además de la corrección de errores simples y de la detección de errores en ráfaga de dos bits (o errores dobles adyacentes), se han incluido los vectores de errores necesarios para detectar errores en ráfaga de 3 bits. Estos vectores contendrán los patrones (…111…) y (…101…).

Nuestra implementación del algoritmo ha permitido, tras 40 minutos de ejecución, y comprobando alrededor de 6 millones de matrices por segundo, encontrar un código que cumple con estas especificaciones. Es decir, un código (38, 32) capaz de corregir errores simples y detectar errores en ráfaga de dos y tres bits. Esta es su matriz de paridad:

$$\mathbf{H} = \begin{bmatrix} 10001101101001110001100010010010001111 \\ 01001001000100011001000111001110010011 \\ 00100100010001011011010010011101101101 \\ 00011100100000111101110110110110110110 \\ 00000011011101111110011110001000100111 \\ 00000000000010101111101111111111111111 \end{bmatrix}$$

Al igual que todos los de esta tesis, este código se ha simulado utilizando inyección de fallos, para comprobar si cumple con sus especificaciones y para compararlo con otros códigos. En este caso, se ha comparado con el código Hamming original y con la propuesta de [Sanchez14], estudiando en los tres casos su capacidad de corrección y detección frente a errores en ráfaga. Los resultados se muestran en la figura 4.4, donde la corrección se representa con columnas punteadas, mientras que la detección se representa con las columnas de color uniforme. Como se puede observar, todos los códigos corrigen errores de un bit (o ráfagas de longitud uno). El código Hamming original presenta un porcentaje de detección muy bajo para errores en ráfaga de longitud dos o superior. El código propuesto en [Sanchez14] mejora ostensiblemente estos resultados, consiguiendo detectar el 100% de errores en ráfaga de dos bits, y porcentajes

no despreciables para tamaños de ráfaga superiores. Sin embargo, el código obtenido utilizando nuestra metodología es capaz, además, de detectar el 100% de errores en ráfaga de tres bits, con lo que se cubre la práctica totalidad de errores múltiples que pueden aparecer en las memorias de semiconductores fabricadas con tecnologías actuales [Baeg09], salvo que la memoria trabaje en un entorno especialmente agresivo o se produzca un fallo *hardware* de dimensiones superiores.

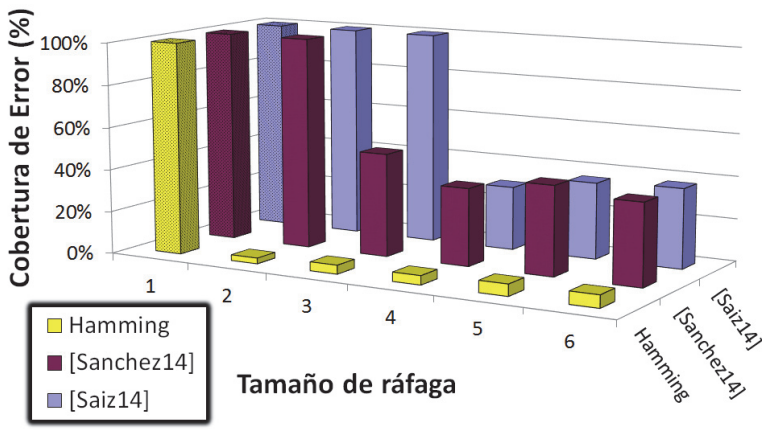

Figura 4.4: Comparativa de códigos (38, 32). Cobertura según el tamaño de ráfaga.

4.5.1.4 Códigos para palabras de datos de 64 bits

El código Hamming (71, 64) tiene 7 bits de paridad (128 síndromes distintos) y solamente asigna 71 síndromes para corrección, más el síndrome cero para indicar que la palabra es correcta. Quedan, por tanto, 56 síndromes disponibles para detección. En [Sanchez14] se utiliza eliminación selectiva y reordenación de las columnas de la matriz del código Hamming original para proponer un código (71, 64) SEC-DAED.

Nuevamente, se ha utilizado la metodología descrita en esta tesis para incluir los vectores de errores necesarios para detectar errores en ráfaga de 3 bits, con los patrones (…111…) y (…101…), aprovechado la cantidad de síndromes disponibles. En este caso, la implementación del algoritmo utilizada ha obtenido una solución en menos de un segundo, comprobando 1,5 millones de matrices por segundo, aproximadamente. Por tanto, se ha encontrado un código (71, 64) capaz de corregir errores simples y detectar errores en ráfaga de dos y tres bits. Esta es su matriz de paridad:

$$\mathbf{H} = \begin{bmatrix}
1000110110100111000010011011100111000111000111000010110010101110101010 \\
0100100100010010001000110010010101110010010101001101010110110010110 \\
0010010001000110111010100011011001010010100100101011001001100011110000 \\
0001110010000010101110001010101100001001001001100100101001001010011001 \\
0000001101110110100001111000000101001000010010011110111000101010010 \\
0000000000001001011111111011111101001000000000000000100011111111111111 \\
0000000000000000000000000100000010110111111111111111111111111111111111
\end{bmatrix}$$

Nuevamente, se han simulado los códigos (71, 64) utilizando inyección de fallos y se han comparado sus coberturas. Los resultados se muestran en la figura 4.5. Como en el caso anterior, la corrección se representa con columnas punteadas y la detección con columnas de color uniforme. Todos los códigos corrigen errores de un bit, como era previsible. El código Hamming original presenta de nuevo un porcentaje de detección muy bajo para errores en ráfaga de longitud dos o superior. El código propuesto en [Sanchez14] detecta el 100% de errores en ráfaga de dos bits, y porcentajes no despreciables para tamaños de ráfaga superiores. El código obtenido utilizando la metodología propuesta en esta tesis detecta el 100% de errores en ráfaga de dos y tres bits, lo que mejora notablemente la cobertura de los códigos anteriores, manteniendo la misma redundancia y latencias de codificación y decodificación similares.

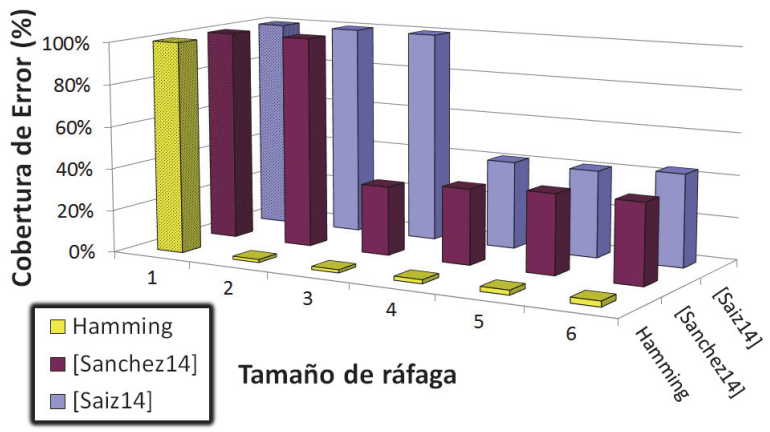

Figura 4.5: Comparativa de códigos (71, 64). Cobertura según el tamaño de ráfaga.

4.5.1.5 Conclusiones

Los códigos presentados en este apartado 4.5.1 son códigos Hamming modificados que aprovechan la capacidad de detección de los códigos no perfectos, es decir, aquellos en los que el número de síndromes disponibles distintos de cero es superior al número de errores a corregir (en este caso, tantos como el tamaño en bits de la palabra codificada). Estos códigos están especialmente ideados para memorias de semiconductores, donde el tamaño de la palabra de datos (sin codificar) es normalmente una potencia de dos.

Los códigos propuestos permiten (para palabras de datos de 32 y 64 bits) la detección de errores en ráfaga de dos y tres bits, manteniendo la misma redundancia, latencias de codificación y decodificación similares, y sólo con un pequeño incremento en la complejidad del decodificador, respecto de los códigos Hamming originales. Esta mayor complejidad es debida a la necesidad de gestionar la detección, no incluida en los códigos Hamming. En todo caso, esta complejidad no implica una mayor latencia. Únicamente requerirá un ligero aumento en el área de silicio requerida para implementar el decodificador.

A pesar de ello la propuesta es interesante, ya que la detección de errores en ráfaga de dos y tres bits cubre la práctica totalidad de errores múltiples que pueden aparecer en las memorias de semiconductores fabricadas con tecnologías actuales [Baeg09], salvo casos muy puntuales. Las matrices de paridad están disponibles para que los códigos puedan ser usados directamente por diseñadores de sistemas.

4.5.2 Códigos para corrección de errores en ráfaga de tres bits en memorias

Las memorias SRAM (*static random access memory*, memorias de acceso aleatorio estáticas) son un componente importante en la mayoría de sistemas electrónicos, incluyendo ASICs (*application specific integrated circuits*, circuitos integrados de aplicación específica). Se espera un incremento de la importancia de las memorias empotradas en estos sistemas, siendo su confiabilidad crítica para la confiabilidad global del sistema. Uno de los retos a los que se enfrentan son los errores causados por el entorno (*soft errors*) inducidos por radiación [Baumann05]. Con la reducción de tamaño de los transistores y el incremento en la escala de integración, un único evento puede causar cambios en los valores almacenados en varias celdas de memoria próximas entre sí.

La combinación del *interleaving* (entrelazado) de varias palabras, cada una de ellas protegida por un código corrector de errores de un bit, es una técnica utilizada habitualmente para evitar los efectos de estos fallos múltiples. Es habitual el uso de códigos SEC (como Hamming) o SEC-DED (como Hsiao o Hamming extendido). El entrelazado permite que bits de una misma palabra se encuentren físicamente separados, reduciendo el riesgo de que un único evento pueda afectar a más de un bit de una palabra [Reviriego10]. Sin embargo, el uso del entrelazado tiene implicaciones en el diseño de la memoria, y puede afectar al área de silicio necesaria para los circuitos y a la latencia de codificación y decodificación. El encaminamiento de los datos es más complejo, incrementando el consumo de energía. Además, no siempre se puede utilizar o no es práctico hacerlo, como en memorias pequeñas o en CAMs (*content addressable memories*, memorias de contenido direccionable). El entrelazado también afecta a la relación de aspecto, lo que puede suponer un problema en memorias empotradas [Neale13].

Cuando el entrelazado no es una buena opción, son necesarios códigos que corrijan varios bits próximos. En su forma más sencilla, serían códigos SEC-DAEC, es decir, que corrigen errores en un bit y en dos bits adyacentes (o error en ráfaga de dos bits). Estos códigos se pueden implementar con la misma redundancia que los códigos SEC-DED, y con un moderado incremento en la complejidad del decodificador. Diversos autores han propuesto este tipo de códigos [Dutta07, Neale13].

Aunque en las actuales tecnologías de fabricación de memorias la mayoría de los errores múltiples causados por un único evento afectan a dos bits adyacentes, se observa un porcentaje no despreciable de fallos que afectan a bits con distancia tres, y se espera un incremento conforme aumenta la escala de integración [Baeg09]. Dada esta situación, los códigos SEC-DAEC pueden dejar de ser efectivos para corregir *soft errors*.

Los fallos que afectan a bits con distancia tres pueden manifestarse como errores triples adyacentes, pero también como errores dobles casi adyacentes (es decir, dos bits en error separados por un bit correcto). En este apartado se presentan códigos SEC-DAEC-TAEC (que solo incluyen los errores triples adyacentes, y que necesitan los mismos bits de paridad que los códigos SEC-DED y SEC-DAEC) y códigos SEC-DAEC que además corrigen errores en ráfaga de tres bits (lo que incluye los errores dobles casi adyacentes, y necesitan un bit de paridad adicional). Estos códigos se presentan para los tamaños de palabra más habituales en las memorias (16, 32 y 64 bits de datos), y se han sintetizado en una biblioteca de componentes de 45nm para compararlos con anteriores propuestas.

Para obtener los códigos se ha aplicado la metodología descrita en este capítulo, aplicando los mismos vectores de error que en el apartado 4.5.1, ajustando los valores de n y k, e incluyendo todos los vectores de error en el conjunto de errores a corregir.

4.5.2.1 Códigos SEC-DAEC-TAEC

En este apartado se presentan códigos capaces de corregir errores simples, dobles adyacentes y triples adyacentes, obtenidos utilizando nuestra metodología, para tamaños de palabra de datos (es decir, el valor de k) de 16, 32 y 64 bits. Uno de los requisitos fijados, además, ha sido mantener la misma redundancia que los códigos SEC-DED correspondientes. Por tanto, se han buscado códigos (22, 16), (39, 32) y (72, 64).

El primer caso requiere especial atención. Tras un largo tiempo de búsqueda, no se ha encontrado ningún código (22, 16) que sea SEC-DAEC-TAEC. Esto no quiere decir que ese código no exista, pero no se ha encontrado. Por número de síndromes disponibles podría existir, ya que los seis bits de paridad hace que se disponga de 63 síndromes distintos de cero, y los posibles errores a corregir son 22+21+20=63 (22 simples, 21 dobles adyacentes y 20 triples adyacentes). Pero esto implica que todos los síndromes deben coincidir exactamente con todos los errores a corregir, por lo que es difícil encontrar la matriz adecuada. Con la implementación actual del algoritmo de búsqueda, utilizando distintas alternativas para gestionarlas (distintos órdenes a la hora de comprobar las columnas, o fijando valores en algunas columnas, por ejemplo) y ejecutando el algoritmo durante un tiempo razonable, no se ha conseguido encontrar el código buscado.

Se podría haber obtenido el código de forma sencilla utilizando un bit de paridad más. Pero dado que no se desea incrementar la redundancia, se ha optado por buscar un código que corrija tantos errores triples adyacentes como sea posible, detectando el resto. Esta solución de compromiso permite, además, demostrar la flexibilidad de la metodología presentada. Se ha conseguido encontrar un código SEC-DAEC-quasiTAEC, que corrige el 90% de los errores triples adyacentes (es decir, 18 de 20), mientras que el 10% restante (2 de 20) son detectados pero no corregidos. Su matriz de paridad es la siguiente:

$$\mathbf{H} = \begin{bmatrix} 1000001010010011100010 \\ 0100000101001110101000 \\ 0010001000110111110110 \\ 0001000100010001000101 \\ 0000100101000100111010 \\ 0000010010111010101101 \end{bmatrix}$$

El código (39, 32) SEC-DAEC-TAEC se ha encontrado sin problemas. Esta es la matriz de paridad obtenida:

$$\mathbf{H} = \begin{bmatrix}
1000000101010100001011001101100100100 10 \\
0100000010100011001000101001000011001 01 \\
0010000100001000100001010101001000011 01 \\
0001000010010000011101100111010010100 10 \\
0000100001001001000000101011101001010 01 \\
0000010000100100100100000100111110101 00 \\
0000001000000010010010010010010101010 10
\end{bmatrix}$$

La metodología presentada en esta tesis también ha permitido encontrar el código (72, 64) SEC-DAEC-TAEC, cuya matriz de paridad es la siguiente:

$$\mathbf{H} = \begin{bmatrix}
100000001010101000010100010101001000110000010010001101010010100000100110 \\
010000000101000110000010010000011101101111100101010000010001001010010100 \\
001000001000010001000001000111010011001000010100100100110111011010110010 1 \\
000100000100100000101000110100000100010101001010010010001110001000100001 \\
000010000010010010000000001001010110000001010100010001001010010100101100 0 \\
000001000001001001001000000000100010101010010011011001001001010101101000 \\
000000100000000100100101000000001001010001001001000101100100010010010010 \\
000000010000000000010010101010100000000100100100101010101011101111001111
\end{bmatrix}$$

Estos códigos presentan la ventaja de que, con la misma redundancia que los códigos SEC-DED (como los Hsiao) o los códigos SEC-DAEC (como los presentados en [Dutta07]), se consigue mejorar la capacidad de corrección de patrones de errores que cada vez son más frecuentes. Sin embargo, las causas físicas que provocan un error triple adyacente también podrían causar un error doble casi adyacente. La figura 4.6 ayuda a ilustrar esta cuestión. Supongamos, por ejemplo, que varias celdas de memoria adyacentes se ven afectadas por el impacto de una partícula de alta energía, cuyo resultado es forzar el bit almacenado en todas ellas a 1. El fallo transitorio que se produce se manifestará como error sólo si el dato almacenado era un 0. En la figura, el fallo afecta a tres celdas adyacentes, pero el error que se manifiesta es un error doble casi adyacente. En el siguiente apartado se presentan códigos capaces de corregir también estos errores.

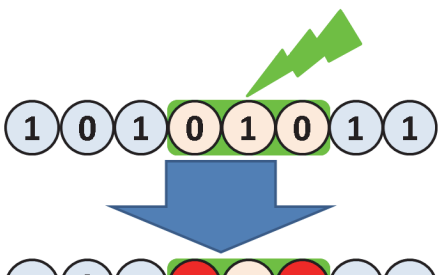

Figura 4.6: Ilustración de un fallo que afecta a 3 bits adyacentes (en verde), que se manifiesta como un error doble casi adyacente (en rojo).

4.5.2.2 Corrección de errores en ráfaga de tres bits

Tal como se ha justificado, sería interesante disponer de códigos capaces de corregir errores simples, errores en ráfaga de dos bits (o errores dobles adyacentes) y errores en ráfaga de tres bits (lo que incluye los errores triples adyacentes y los errores dobles casi adyacentes). Por el número de síndromes necesario, no es posible encontrar estos códigos manteniendo el mismo nivel de redundancia para ninguno de los tamaños de palabra considerados. Por tanto, se ha intentado encontrar estos códigos añadiendo un bit de paridad adicional, y en este caso sí ha sido posible encontrar los códigos para tamaños de palabra de datos (es decir, el valor de k) de 16, 32 y 64 bits. En este apartado se presentan códigos (23, 16), (40, 32) y (73, 64).

El código (23, 16) capaz de corregir errores simples y errores en ráfaga de dos y tres bits que se ha encontrado utilizando nuestra metodología tiene esta matriz de paridad:

$$\mathbf{H} = \begin{bmatrix} 10000001001110010000001 \\ 01000000100100001110010 \\ 00100000010011100010100 \\ 00010001000100101001100 \\ 00001000100010010011100 \\ 00000100010000101100101 \\ 00000010001001001001011 \end{bmatrix}$$

El código (40, 32) que cumple los requerimientos buscados se puede representar con esta matriz de paridad:

$$\mathbf{H} = \begin{bmatrix} 1000000010010110010001100011100100000000 \\ 0100000001001001001000100100110001010010 \\ 0010000000100100010100001001001100110100 \\ 0001000010000100100000010110010011001001 \\ 0000100001000000010011000100101000011100 \\ 0000010000100010001000100010010110000101 \\ 0000001000010001000100010001001010100110 \\ 0000000100001000100010001000000001101011 \end{bmatrix}$$

También se ha encontrar el código (73, 64) con los mismos requisitos. Esta es la matriz de paridad obtenida:

$$\mathbf{H} = \begin{bmatrix}
0000000010000010000100001000100100010000000000000000100100001001001100100 \\
0000000100000100001000010001000010000000001001001101000001001001001000001 \\
0000001000001000010000100000001000000100100001100001001010110100100101010 \\
0000010000010000100001000000000100101001000101000110010101000100010010100 \\
0000100000100001000000010010010001011010001010100100001001001000100011001 \\
0001000001000000010010000100110010000001001001001001010000011001100000011 \\
0010000000010010000010000011001001001100100110010011000010000111001001000 \\
0100000000100100100100100100000001100110011100001000111001100010000100100 \\
1000000001001001001011001110010000100011110000110100100100100000011000010
\end{bmatrix}$$

4.5.2.3 Comparación y conclusiones

Con el objetivo de valorar si las propuestas realizadas son interesantes, se ha realizado una comparación de varios códigos desde distintos puntos de vista. Los códigos presentados en este apartado se han comparado con los códigos SEC-DAEC presentados en [Dutta07], para los distintos tamaños de palabra de datos considerados (16, 32 y 64 bits). También se ha comparado el código publicado en [She12], que corrige errores simples, y errores en ráfaga de dos y tres bits, pero únicamente se propone para palabras de datos de 16 bits.

En la tabla 4.5 se muestran todos estos códigos, indicando la redundancia o número de bits de paridad que requieren ($n-k$). Además, la complejidad a la hora de implementar los codificadores y decodificadores se compara utilizando dos parámetros adicionales. El número total de 1s en la matriz de paridad, o peso Hamming de dicha matriz, da una referencia del número de puertas lógicas necesarias para implementar los circuitos [Dutta07]. Por tanto, a mayor peso Hamming de la matriz, suele ser necesaria mayor área de silicio. Igualmente, el número de 1s en una fila suele estar relacionado con la latencia necesaria para el cálculo del bit de paridad o de síndrome correspondiente (ya que cada fila de la matriz de paridad se utiliza para calcular un bit de paridad en el codificador, y un bit de síndrome en el decodificador). Dado que hay que esperar a que se calculen todos los bits de paridad (o de síndrome), la fila con mayor peso Hamming es la que marca la latencia global del circuito correspondiente [Dutta07].

Como se puede observar en la tabla, los códigos propuestos en este apartado (con fondo resaltado) igualan o mejoran prestaciones, y tienen pesos Hamming similares o mejores que los códigos equivalentes. Para palabras de datos de 16 bits, se puede observar que la propuesta SEC-DAEC-quasiTAEC aquí presentada mejora la capacidad de corrección y detección, manteniendo la misma redundancia y pesos Hamming similares. Con un bit de paridad más, se pueden corregir errores en ráfaga de 3 bits. En este caso, el código propuesto mejora ligeramente los datos de pesos Hamming sobre la propuesta de [She12].

Tabla 4.5. Comparación de redundancia y complejidad de distintos códigos

k	*Código*	*n−k*	*Número total de 1s en* **H**	*Máximo de 1s en una fila*
16	SEC-DAEC [Dutta07]	6	54	10
	SEC-DAEC-quasiTAEC	6	51	11
	ECC para ráfagas de 3 bits [She12]	7	54	8
	ECC para ráfagas de 3 bits	7	49	7
32	SEC-DAEC [Dutta07]	7	103	15
	SEC-DAEC-TAEC	7	92	15
	ECC para ráfagas de 3 bits	8	91	12
64	SEC-DAEC [Dutta07]	8	232	32
	SEC-DAEC-TAEC	8	189	27
	ECC para ráfagas de 3 bits	9	180	25

Para 32 y 64 bits se puede observar la misma tendencia en cuanto a la redundancia, y mejoras sensibles en los pesos Hamming de las propuestas presentadas en esta tesis, respecto de las de [Dutta07].

Para completar la comparación, todos estos códigos se han implementado mediante un lenguaje de descripción de *hardware* y se han sintetizado utilizando una biblioteca de componentes de 45nm. La síntesis se ha realizado dos veces, una optimizando el área de silicio ocupada por el circuito y otra optimizando su latencia.

Estos datos se muestran en la tabla 4.6. Como en la tabla anterior, los códigos obtenidos utilizando la metodología propuesta en esta tesis tienen el fondo resaltado. Se pueden distinguir dos grupos de columnas. El primero de ellos contiene los datos obtenidos al optimizar reduciendo el área de silicio ocupada por los circuitos, mientras que el segundo grupo muestra los datos obtenidos al optimizar reduciendo la latencia de los circuitos. En todo caso, se muestran los datos de área (expresada en μm^2) y de latencia (expresada en ns) en ambas optimizaciones, tanto para el circuito codificador como para el decodificador. Las columnas optimizadas aparecen remarcadas en negrita.

Como se puede observar en la tabla, los códigos presentados en este apartado mejoran las capacidades de corrección de los códigos publicados en [Dutta07], con una sobrecarga espacial y temporal moderada. Incluso, en casos puntuales, los resultados son mejores en los códigos aquí presentados. En cuanto al código presentado en [She12], el código propuesto equivalente mantiene la capacidad de corrección y mejora o iguala los datos para los que se ha optimizado la síntesis (área en la primera parte y latencia en la segunda).

En definitiva, los códigos propuestos mejoran la capacidad de corrección de los códigos SEC-DAEC existentes y pueden ser implementados eficientemente con una moderada sobrecarga espacial y/o temporal y la misma redundancia (o un bit de paridad

adicional). Todos estos códigos se han simulado utilizando inyección de fallos para comprobar su correcto funcionamiento. Las matrices de paridad están presentadas en forma sistemática y pueden utilizarse directamente por diseñadores de sistemas que necesiten proteger sus memorias contra fallos triples adyacentes.

Tabla 4.6. Comparación de área y latencia de los circuitos para distintos códigos

		OPTIMIZACIÓN EN ÁREA				*OPTIMIZACIÓN EN LATENCIA*			
		CODIFICADOR		*DECODIFICADOR*		*CODIFICADOR*		*DECODIFICADOR*	
k	***Código***	***área***	***latencia***	***área***	***latencia***	***área***	***latencia***	***área***	***latencia***
16	SEC-DAEC [Dutta07]	**156**	0.47	**1006**	1.06	229	**0.23**	1439	**0.47**
	SEC-DAEC-quasiTAEC	**91**	0.44	**1135**	1.23	119	**0.21**	1528	**0.49**
	ECC para ráfagas de 3 bits [She12]	**131**	0.38	**1404**	1.35	172	**0.19**	1963	**0.47**
	ECC para ráfagas de 3 bits	**124**	0.33	**1356**	1.20	179	**0.19**	1998	**0.46**
32	SEC-DAEC [Dutta07]	**322**	0.53	**1902**	1.33	387	**0.29**	2751	**0.53**
	SEC-DAEC-TAEC	**279**	0.49	**2202**	1.34	328	**0.26**	3000	**0.54**
	ECC para ráfagas de 3 bits	**264**	0.45	**2628**	1.32	376	**0.24**	3599	**0.54**
64	SEC-DAEC [Dutta07]	**678**	0.61	**3106**	1.75	812	**0.33**	4227	**0.61**
	SEC-DAEC-TAEC	**583**	0.65	**3672**	1.67	703	**0.30**	4928	**0.62**
	ECC para ráfagas de 3 bits	**566**	0.58	**5279**	1.81	695	**0.30**	7165	**0.62**

4.6 Conclusiones

En este capítulo se han presentado los códigos *Flexible Unequal Error Control* (FUEC) *codes* (códigos flexibles para control desigual de errores). La principal cualidad de estos códigos es la posibilidad de aplicar control desigual a los distintos bits de una palabra, lo que es especialmente interesante para situaciones en las que se produce una tasa de error variable en el espacio, es decir, que no todos los bits de una palabra tienen la misma tasa de error, como sucede con los fallos intermitentes.

Tras justificar su necesidad, se han presentado sus características principales y se ha descrito la metodología para obtener estos códigos. Se han presentado distintos ejemplos y se han comparado con códigos similares para demostrar sus características y posibilidades. La principal ventaja de la metodología aquí propuesta es su flexibilidad, que le permite obtener cualquier código, si existe, en función de los vectores de errores a corregir y/o detectar. El principal inconveniente es que si el tamaño de palabra es grande (>64 bits, aproximadamente), la búsqueda puede llevar demasiado tiempo. Una de las cuestiones de cara al trabajo futuro es seguir mejorando las prestaciones del programa de búsqueda, haciendo una versión paralelizada del algoritmo.

A continuación se ha introducido la cuestión de implementar un mecanismo de tolerancia a fallos capaz de tolerar fallos intermitentes utilizando los códigos FUEC. Como se ha justificado, los códigos no pueden tolerar por sí mismos fallos intermitentes, ya que no son capaces de manejar la variabilidad temporal de la tasa de fallos. Aunque se está trabajando en un modelo que sirva como prueba de concepto, todavía no está totalmente implementado. Esta línea de investigación, y todos los problemas que pueden surgir a la hora de implementar este mecanismo de tolerancia a fallos intermitentes, son los principales retos a los que el autor de esta tesis, junto con el grupo de investigación, se enfrenta en estos momentos, siendo una parte fundamental del trabajo futuro.

En la parte final del capítulo se han presentado otras aplicaciones de la metodología de diseño de códigos, que demuestran su flexibilidad. En este caso, en lugar de la tasa de error variable, se ha explotado otra de las características de los fallos en general, y de los fallos intermitentes en particular, como es la aparición cada vez más frecuente de fallos múltiples adyacentes. Estos fallos se suelen manifestar como errores en ráfaga, y su detección o corrección resulta interesante para mejorar la confiabilidad del sistema. En primer lugar, se ha visto que con la metodología presentada en esta tesis se podían encontrar códigos Hamming modificados para mejorar la capacidad de detección de errores en ráfaga de dos y tres bits, sin incrementar la redundancia. A continuación, se han buscado códigos capaces de corregir esos errores en ráfaga, intentando mantener la redundancia lo más baja posible.

En resumen, se ha desarrollado una metodología que permite obtener códigos con las características deseadas. En concreto, se ha demostrado que los códigos FUEC tienen unas características únicas, que los hacen especialmente interesantes cuando se dan tasas de error variables en el espacio, y que pueden formar parte de mecanismos de tolerancia a fallos intermitentes que sean capaces de detectarlos y adaptarse a las nuevas circunstancias con el objetivo de reducir el riesgo de avería.

5

Conclusiones y trabajo futuro

En este capítulo se realiza un resumen del trabajo expuesto en la presente tesis. En primer lugar se extraen las conclusiones más importantes del trabajo desarrollado. A continuación se abordan las futuras líneas de investigación en las que continuar el trabajo presentado. Posteriormente se indican los proyectos que han dado cobertura económica y científica a esta tesis. Por último se exponen las publicaciones a las que, directa o indirectamente, ha dado lugar.

5.1 Conclusiones

Los sistemas informáticos ocupan un papel muy importante en el día a día de nuestra sociedad, y es prioritario que estos sistemas duren el mayor tiempo posible y con un elevado nivel de eficacia, especialmente aquellos en los que un mal funcionamiento puede causar impacto económico, suspensión de servicios básicos o pérdida de vidas humanas.

Los Sistemas Tolerantes a Fallos son sistemas que disponen de mecanismos especiales que proporcionan una cierta inmunidad a la ocurrencia de fallos que puedan causar un cese o deterioro del servicio prestado. Se denomina confiabilidad a la propiedad

de un sistema informático que permite depositar una confianza justificada en el servicio que proporciona. La confiabilidad se debe ver desde el punto de vista de sus atributos, amenazas y medios [Laprie92, Avizienis04].

Los atributos de la confiabilidad permiten expresar las propiedades que se esperan de un sistema así como valorar la calidad del servicio entregado. Los atributos de la confiabilidad son disponibilidad, fiabilidad, inocuidad, confidencialidad, integridad y mantenibilidad.

Las amenazas son circunstancias no deseadas que provocan la pérdida de la confiabilidad. Las amenazas de la confiabilidad son fallos, errores y averías. La aparición de fallos en el sistema provoca errores que, a su vez, pueden desencadenar averías, que son comportamientos anómalos que hacen incumplir la función del sistema.

Los medios para conseguir la confiabilidad son los métodos y técnicas que capacitan al sistema para entregar un servicio en el que se pueda confiar, y que permiten al usuario tener confianza en esa capacidad. Los medios son la prevención de fallos, la tolerancia a fallos, la eliminación de fallos y la predicción de fallos. La prevención de fallos se corresponde con las técnicas generales de diseño de sistemas. La tolerancia a fallos consiste en la utilización de técnicas que permitan al sistema cumplir con su función a pesar de la existencia de fallos. La eliminación de fallos intenta, mediante las etapas de verificación, diagnóstico y corrección, reducir la existencia de fallos en el sistema. La predicción de fallos intenta estimar el número de fallos en un sistema y su gravedad.

Una de las amenazas de la confiabilidad que está cobrando especial importancia en los últimos años son los fallos intermitentes [Constantinescu03]. Estos fallos se producen por un *hardware* defectuoso que por lo general funciona correctamente, pero que ante determinadas variaciones del entorno, como alta temperatura o carga de trabajo elevada, puede no hacerlo. Con la reducción del tamaño de los dispositivos electrónicos aparecen nuevos mecanismos físicos que pueden provocar fallos intermitentes. Estudios recientes confirman su importancia, tanto en el procesador como en memoria [Constantinescu08, Schroeder09, Nightingale11].

En esta tesis se ha hecho un exhaustivo estudio de las causas y efectos de los fallos intermitentes en los sistemas informáticos. Se han analizado los mecanismos físicos que pueden provocarlos y se han determinado modelos de fallo representativos. A la hora de parametrizar estos modelos, hay que tener en cuenta que los fallos intermitentes se activan y desactivan de forma repetitiva y no determinista en el mismo punto, y que generalmente aparecen agrupados en ráfagas (varias activaciones y desactivaciones seguidas en el tiempo) separadas entre ellas por un (relativamente) largo tiempo de no activación.

Con los modelos obtenidos se han llevado a cabo campañas de inyección de fallos para estudiar los efectos de los fallos intermitentes en sistemas no tolerantes a fallos. Se ha estudiado la influencia de distintos parámetros que afectan a los fallos intermitentes: características de una ráfaga (tiempos de actividad e inactividad y longitud de la ráfaga), multiplicidad espacial, lugar de la inyección, etc. La principal conclusión es que los parámetros que más influyen en los efectos de los fallos intermitentes son el tiempo de actividad, la longitud de la ráfaga y la multiplicidad espacial. También tienen su incidencia el lugar de inyección (los buses del sistema se han mostrado como puntos enormemente sensibles a la aparición de los fallos intermitentes) y la frecuencia de trabajo del sistema (a mayor frecuencia de trabajo, mayor es la probabilidad de avería tras la aparición de un fallo intermitente, debido a que la probabilidad de capturar un fallo en flancos activos de los componentes síncronos aumenta). Es importante considerar esto último, dado que la variación de la frecuencia de trabajo es una de las tendencias actuales en los sistemas VLSI.

Por todo lo expuesto hasta ahora, se puede concluir que son necesarias técnicas de tolerancia a fallos específicas para mitigar los efectos de los fallos intermitentes. El siguiente paso ha sido estudiar algunas de las técnicas de tolerancia a fallos más usadas habitualmente, para ver si son adecuadas. Para ello se han inyectado fallos en un microprocesador tolerante a fallos basado en el microprocesador académico MARK2. Tras analizar los resultados obtenidos, se puede concluir que, aunque los mecanismos de detección de fallos funcionan adecuadamente, los mecanismos de recuperación utilizados no son todo lo eficaces que deberían ante la aparición de fallos intermitentes. Esto es debido, principalmente, a que dichos mecanismos están orientados, bien para fallos transitorios, o bien para permanentes, mientras que los fallos intermitentes tienen características específicas que los diferencian de estos dos tipos de fallos.

Con el objetivo de encontrar nuevas técnicas de tolerancia a fallos específicas para fallos intermitentes, esta tesis presenta una metodología para diseñar códigos correctores de errores con unas características específicas. Esta metodología se basa en una búsqueda exhaustiva de matrices de paridad, cuya principal ventaja es su flexibilidad, ya que permite obtener cualquier código, si existe, en función de los vectores de errores a corregir y/o detectar. El principal inconveniente es que si el tamaño de palabra es grande (>64 bits, aproximadamente), la búsqueda puede llevar demasiado tiempo. Con el objetivo de reducir el coste computacional, se van generando y comprobando matrices parciales con un algoritmo recursivo, que permite comprobar todas las posibles matrices aplicando la técnica de *backtracking* o vuelta atrás. También aplica la técnica de *branch and bound* (ramificación y poda), lo que permite descartar las matrices parciales que no pueden ser parte de una solución válida.

Utilizando esta metodología se han obtenido los códigos *Flexible Unequal Error Control* (FUEC) *codes* (códigos flexibles para control desigual de errores). La principal

cualidad de estos códigos es la posibilidad de aplicar control desigual a los distintos bits de una palabra, lo que es especialmente interesante para situaciones en las que se produce una tasa de error variable en el espacio, es decir, que no todos los bits de una palabra tienen la misma tasa de error, como sucede con los fallos intermitentes.

Como se ha justificado en esta tesis, ningún otro código tiene todas las características necesarias para tolerar fallos intermitentes en buses o elementos de almacenamiento de un procesador. Los códigos FUEC son códigos lineales binarios con capacidad para aplicar control asimétrico, con codificadores y decodificadores simples y fácilmente implementables en *hardware*, por lo que su latencia es baja. También aseguran, si se realiza adecuadamente el proceso de búsqueda de los códigos, que la redundancia es la mínima necesaria para la cobertura seleccionada. Esta cobertura se puede elegir de forma flexible, acorde a los requerimientos de diseño, combinando la detección y/o corrección de errores simples, múltiples en ráfaga y/o múltiples aleatorios.

En cualquier caso, los códigos no pueden tolerar por sí mismos fallos intermitentes, ya que no son capaces de manejar la variabilidad temporal de la tasa de fallos. Aunque se queda fuera del alcance de esta tesis, se ha introducido la cuestión de implementar un mecanismo de tolerancia a fallos capaz de tolerar fallos intermitentes utilizando los códigos FUEC.

También se han presentado otras aplicaciones de la metodología de diseño de códigos, que demuestran su flexibilidad. En este caso, se ha explotado otra de las características de los fallos en general, y de los fallos intermitentes en particular, como es la aparición cada vez más frecuente de fallos múltiples adyacentes, que suelen manifestarse como errores en ráfaga. Con nuestra metodología se han encontrado códigos Hamming modificados para mejorar la capacidad de detección de errores en ráfaga de dos y tres bits, sin incrementar la redundancia, y códigos capaces de corregir esos mismos errores, intentando mantener la redundancia tan baja como sea posible.

5.2 Trabajo futuro

Como se ha comentado anteriormente, utilizando únicamente códigos no se pueden resolver los problemas generados por los fallos intermitentes, ya que no pueden adaptarse por sí mismos a la evolución temporal de las condiciones de fallo. Dado que los fallos intermitentes provocan variabilidad espacial y temporal en la tasa de fallo, es necesario un mecanismo de tolerancia a fallos que incluya ambas vertientes. Los códigos propuestos en esta tesis pueden gestionar adecuadamente la variabilidad espacial de la tasa de fallo, pero se requiere circuitería adicional que controle la evolución temporal, es decir,

que detecte cuándo y dónde se producen fallos intermitentes y adapte adecuadamente el comportamiento del sistema. Esta es mi principal línea de investigación en este momento, y donde se está enfocando el esfuerzo de cara al futuro inmediato.

El segundo gran reto a abordar es la paralelización del algoritmo de búsqueda de matrices utilizado para la obtención de los códigos propuestos en esta tesis. Ya se ha hablado del elevado coste computacional que esta búsqueda comporta. A pesar de las importantes mejoras acometidas, que han permitido acelerar notablemente el proceso desde las primeras versiones hasta la actual, el tiempo necesario para realizar una ejecución completa suele ser inabordable. Convertir la actual versión secuencial del algoritmo a una versión paralelizada permitiría aprovechar toda la potencia de cálculo que puede ofrecer, por ejemplo, un *cluster* de computadores, y permitiría búsquedas más complejas y resolverlas en menor tiempo.

Junto a la paralelización del algoritmo de búsqueda, es importante acometer mejoras en cuanto al manejo de la implementación de dicho algoritmo: parametrización, facilidad de uso, mejoras en el interfaz con el usuario y protección frente a caídas del sistema o cortes del suministro eléctrico. Esta última cuestión es importante, dado que las búsquedas pueden ser largas. Disponer de un mecanismo que registre cada cierto tiempo el punto actual de la búsqueda, y que permita continuar la búsqueda a partir de un punto intermedio en caso de caída, minimizaría el riesgo de pérdida de trabajo por estas causas.

Con todas estas mejoras, se podrá seguir buscando códigos detectores y correctores de errores que sean útiles para los sistemas informáticos actuales y futuros. Está previsto trabajar en códigos ultrarrápidos para los registros del procesador, lo que permitiría codificaciones y decodificaciones que afecten lo menos posible a las prestaciones del sistema. Además, en el caso de los fallos intermitentes, reducirían el riesgo de que una nueva activación del fallo afecte a la recuperación y provoque una avería. Otra posible propuesta es plantear códigos para memorias, para tolerar fallos como el denominado *chipkill* [Dell97], con menor redundancia o menos requisitos estructurales que las propuestas actuales. La búsqueda de nuevos códigos para sistemas VLSI debe ser una de las líneas de investigación del autor de esta tesis.

En cuanto al conocimiento del comportamiento de los sistemas digitales ante la ocurrencia de fallos intermitentes, se seguirá trabajando en la inyección de fallos en distintos sistemas para comprobar su comportamiento. En concreto, la implementación del mecanismo adaptativo de tolerancia a fallos intermitentes debe ser estudiada a fondo para comprobar si, efectivamente, mejora la confiabilidad del sistema y su respuesta ante la aparición de fallos intermitentes. Con el sistema en el que se implemente el nuevo mecanismo se hará un estudio similar al llevado a cabo con el microprocesador MARK2 tolerante a fallos.

Finalmente, es deseable que todas estas investigaciones redunden en algo útil para la sociedad. Para ello, la intención del autor, junto con su grupo de investigación, es abordar proyectos de colaboración con empresas y entidades donde estas líneas de investigación puedan resultar interesantes. Seguir colaborando con entidades como el Instituto Nacional de Técnica Aeroespacial (INTA), o abordar proyectos con empresas del sector automovilístico o aeroespacial, por ejemplo, es prioritario.

5.3 Proyectos de investigación

Esta tesis ha sido parcialmente financiada por los siguientes proyectos de investigación:

- Generación e Integración automática de Mecanismos para la mejora de la Confiabilidad y la Seguridad de circuitos VLSI (GIMCS), PAID-06-10-2388. Proyecto financiado por la Universitat Politècnica de València.
- Sistemas EMpotrados SEguros y Confiables bAsados en comPonentes (SEMSECAP), TIN-2009-13825. Proyecto financiado por el Ministerio de Ciencia e Innovación, Gobierno de España.
- DESarrollo de Técnicas de Tolerancia a fallos adaptativas basadas en redundancia de la información (DESTT), SP20120806. Proyecto financiado por la Universitat Politècnica de València.
- *Adaptive and REsilient Networked Embedded Systems* (ARENES), TIN2012-38308-C02-01. Proyecto financiado por el Ministerio de Economía y Competitividad, Gobierno de España.

5.4 Resultados de investigación

El trabajo de investigación relacionado con esta tesis ha dado lugar a 18 publicaciones, entre ellas cuatro artículos en revistas de alto impacto (Q1 o Q2 del índice *Journal Citation Reports* (JCR)) y numerosas ponencias en congresos internacionales, indexados en la lista *Computing Research and Education* (CORE) y/o recomendados por el *International Federation for Information Processing* (IFIP) *Working Group 10.4 on Dependable Computing and Fault Tolerance*. La investigación plasmada en estas

publicaciones ha permitido al autor de esta tesis la evaluación positiva de un tramo de la actividad investigadora, equivalente a un sexenio, para el período 2007-2012.

También es de destacar que algunas de las publicaciones aquí presentadas han sido referenciadas en trabajos publicados por otros investigadores. Las principales referencias se incluyen en el apartado 5.4.4.

5.4.1 Artículos en revistas de alto impacto

[DGil12a] Daniel Gil-Tomás, Joaquín Gracia-Morán, J.-Carlos Baraza-Calvo, **Luis-J. Saiz-Adalid** y Pedro-J. Gil-Vicente, *"Analyzing the impact of intermittent faults on microprocessors applying fault injection"*, *IEEE Design and Test of Computers*, volumen 29, número 6, páginas 66–73, Noviembre–Diciembre 2012.

Mi contribución a este artículo es el estudio bibliográfico de los mecanismos físicos de los fallos intermitentes y de trabajos de observación de errores en sistemas reales que son atribuidos a fallos intermitentes, así como la definición de los modelos de fallos intermitentes. Participé en la selección del modelo adecuado de procesador RISC, y en el análisis de los resultados de los experimentos; trabajé en la selección y organización de los contenidos del artículo, su redacción, maquetación y revisión.

La revista *IEEE Design and Test of Computers* está indexada en JCR con un factor de impacto de 1,623 en 2012 (año de publicación), lo que la sitúa en la posición 10 de 50, por tanto en el **primer cuartil**, en la categoría *"COMPUTER SCIENCE, HARDWARE & ARCHITECTURE"*.

[DGil12b] Daniel Gil-Tomás, Joaquín Gracia-Morán, J.-Carlos Baraza-Calvo, **Luis-J. Saiz-Adalid** y Pedro-J. Gil-Vicente, *"Studying the effects of intermittent faults on a microcontroller"*, *Microelectronics Reliability*, volumen 52, número 11, páginas 2387–2846, Noviembre 2012.

Mi contribución al artículo fue el estudio de la bibliografía, la selección del modelo adecuado de un procesador RISC, la planificación y realización de los experimentos de inyección, y el análisis de los resultados; colaboré en la selección y organización de los contenidos del artículo, su redacción, maquetación y revisión.

La revista *Microelectronics Reliability* tiene un factor de impacto en JCR de 1,137 en 2012 (año de publicación), lo que la sitúa en la posición 119 de 243, por tanto en el **segundo cuartil**, en la categoría *"ENGINEERING, ELECTRICAL & ELECTRONIC"*.

[Gracia14a] Joaquín Gracia-Moran, J.-Carlos Baraza-Calvo, Daniel Gil-Tomas, **Luis-J. Saiz-Adalid** y Pedro-J. Gil-Vicente, *"Effects of Intermittent Faults on the Reliability of a Reduced Instruction Set Computing (RISC) Microprocessor"*, *IEEE Transactions on Reliability*, volumen 63, número 1, páginas 144–153, Marzo 2014.

Mi contribución a este artículo es el estudio bibliográfico, colaborando en la definición de los modelos de fallos intermitentes utilizados, en la selección del modelo de procesador RISC, y en el análisis de los resultados de los experimentos; también trabajé en la selección de los contenidos del artículo, su redacción y revisión.

La revista *IEEE Transactions on Reliability* alcanza un factor de impacto en JCR de 1,934 en 2014 (año de publicación), lo que la sitúa en la posición 7 de 50, por tanto en el **primer cuartil**, en la categoría *"COMPUTER SCIENCE, HARDWARE & ARCHITECTURE"*.

[Saiz15] **Luis-J. Saiz-Adalid**, Pedro Reviriego, Pedro Gil, Salvatore Pontarelli y Juan-A. Maestro, *"MCU Tolerance in SRAMs through Low Redundancy Triple Adjacent Error Correction"*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volumen 23, número 10, páginas 2332–2336, Octubre 2015.

Como primer autor de este artículo, me encargué de coordinar un equipo internacional formado por miembros de tres instituciones distintas. Desarrollé la herramienta que permite el diseño de los códigos correctores de errores y realicé los experimentos de validación de los mismos. Fui el redactor principal del artículo, organizando y revisando los contenidos y su maquetación.

La revista *IEEE Transactions on VLSI* está indexada en JCR con un factor de impacto de 1,356 en 2014 (últimos datos disponibles), lo que la sitúa en la posición 16 de 50 (**segundo cuartil**) en la categoría *"COMPUTER SCIENCE, HARDWARE & ARCHITECTURE"*.

5.4.2 Ponencias en congresos destacados

[DGil08] Daniel Gil-Tomás, **Luis-J. Saiz-Adalid**, Joaquín Gracia-Morán, J.-Carlos Baraza-Calvo y Pedro-J. Gil-Vicente, *"Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers"*, *IEEE International High Level Design, Validation and Test Workshop* 2008 (HLDVT'08), Incline Village (Nevada-EE.UU.), Noviembre 2008.

Mi aportación a este artículo fue el estudio bibliográfico, la definición de los nuevos modelos de fallos intermitentes, la planificación y realización de experimentos de inyección y el análisis de los resultados; también colaboré en la selección y organización de los contenidos del artículo, su redacción, maquetación y revisión. Finalmente, realicé la presentación en el congreso.

La conferencia *High Level Design, Validation and Test* (HLDVT) está patrocinada por el *IEEE Computer Society Test Technology Technical Council* y por el *IEEE Computer Society Design Automation Technical Committee*. Estaba considerada de **categoría C** según la clasificación CORE de 2010.

[Gracia10a] Joaquín Gracia-Morán, Daniel Gil-Tomás, **Luis-J. Saiz-Adalid**, J.-Carlos Baraza-Calvo y Pedro-J. Gil-Vicente, *"Experimental Validation of a Fault Tolerant Microcomputer System against Intermittent Faults"*, *40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2010), Chicago (Illinois-EE.UU.), Junio 2010.

Mis aportaciones a este trabajo fueron el estudio de la bibliografía, la realización de los experimentos de inyección y el análisis de los resultados, la selección y organización de los contenidos del artículo, su redacción, maquetación y revisión.

El congreso *Dependable Systems and Networks* (DSN) es el más importante en el campo de la confiabilidad a nivel mundial. Está patrocinado por IEEE y por el *IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance*. Está considerado como de **categoría A*** según la clasificación CORE de 2013. Además, la base de datos Scopus equipara las actas de este congreso a revistas de muy alto impacto: la edición de 2010 del DSN obtuvo un índice de impacto de 0,870, que lo situaba en la posición 34 de 165 (es decir, en el **primer cuartil**) dentro de la categoría *"Computer Science (miscellaneous)"* del *SCImago Journal Rank*.

[Gracia10b] Joaquín Gracia-Moran, Daniel Gil-Tomas, J.-Carlos Baraza-Calvo, **Luis-J. Saiz-Adalid** y Pedro-J. Gil-Vicente, *"Searching Representative and Low Cost Fault Models for Intermittent Faults in Microcontrollers: A Case Study"*, *16th IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC 2010), Tokio (Japón), Diciembre 2010.

Mi contribución al artículo fue la implementación de las nuevas cargas de trabajo utilizadas, además del estudio bibliográfico y la definición de los nuevos modelos de fallos intermitentes; asimismo, colaboré en la realización de los

experimentos de inyección y el análisis de los resultados, la selección y organización de los contenidos, así como en la redacción, maquetación y revisión del artículo.

La conferencia *Pacific Rim International Symposium on Dependable Computing* (PRDC) está patrocinada por el *IEEE Computer Society Technical Committee on Dependable Computing and Fault Tolerance* y por el *IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance*. Además, está considerada de **categoría B** según la clasificación CORE de 2013.

[Gracia13] Joaquín Gracia-Moran, Daniel Gil-Tomas, **Luis-J. Saiz-Adalid**, J.-Carlos Baraza-Calvo y Pedro-J. Gil-Vicente, *"Defining a Representative and Low Cost Fault Model Set for Intermittent Faults in Microprocessor Buses"*, *Sixth Latin-American Symposium on Dependable Computing* (LADC 2013), Río de Janeiro (Brasil), Abril 2013.

Las aportaciones que realicé a este artículo fueron el estudio de la bibliografía, la definición de los modelos de fallos intermitentes utilizados, colaborando en la selección y organización de los contenidos, y en la redacción, maquetación y revisión del artículo.

El congreso *Latin-American Symposium on Dependable Computing* (LADC), donde se presentó, está patrocinado por *IEEE Computer Society* y por el *IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance*. De hecho, para esta organización, ésta es una de las conferencias recomendadas en el campo de la confiabilidad a nivel mundial.

[Saiz13b] **Luis-J. Saiz-Adalid**, Pedro-J. Gil-Vicente, Juan-Carlos Ruiz-García, Daniel Gil-Tomás, J.-Carlos Baraza-Calvo y Joaquín Gracia-Morán, *"Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels"*, *32nd International Conference on Computer Safety, Reliability and Security* (SAFECOMP 2013), Toulouse (Francia), Septiembre 2013.

Como primer autor de este artículo, además de su presentación en el congreso, coordiné el trabajo realizado. Tras un estudio bibliográfico para justificar el problema presentado y obtener la información necesaria para desarrollar una posible solución, implementé la aplicación que permite diseñar los códigos correctores de errores y los experimentos para su validación. Fui el redactor principal del artículo, organizando y revisando los contenidos y su maquetación.

El congreso *Computer Safety, Reliability and Security* (SAFECOMP) está patrocinado por el *European Workshop on Industrial Computer Systems, Technical*

Committee 7 on Reliability, Safety and Security y por el *IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance*. Además, está considerado de **categoría B** según la clasificación CORE de 2014.

[Saiz14] **Luis-J. Saiz-Adalid**, Pedro Gil, J.-Carlos Baraza-Calvo, Juan-Carlos Ruiz, Daniel Gil-Tomás y Joaquín Gracia-Morán, *"Modified Hamming Codes to Enhance Short Burst Error Detection in Semiconductor Memories"*, *10th European Dependable Computing Conference* (EDCC 2014), Newcastle (Reino Unido), Mayo 2014.

Tras el necesario estudio bibliográfico, como primer autor coordiné el equipo de investigación, utilizando la herramienta de diseño de códigos para resolver el problema propuesto. Fui el redactor principal del artículo, organizando y revisando los contenidos y su maquetación.

La conferencia *European Dependable Computing Conference* (EDCC), donde se presentó [Saiz14], está patrocinado por *IEEE Computer Society* y por el *IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance*. De hecho, para esta organización, ésta es una de las conferencias recomendadas en el campo de la confiabilidad a nivel mundial.

Es de destacar que el *IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance*, institución referente a nivel mundial en el campo de la confiabilidad, es coorganizador de la conferencia *Dependable Systems and Networks* (DSN). Además, da soporte y coopera en la organización de los congresos *Computer Safety, Reliability and Security* (SAFECOMP), *Pacific Rim International Symposium on Dependable Computing* (PRDC), *European Dependable Computing Conference* (EDCC) y *Latin-American Symposium on Dependable Computing* (LADC).

5.4.3 Otras publicaciones

- **[Saiz06]** **Luis-J. Saiz-Adalid**, *"Fast and Early Validation of VLSI Systems"*, presentado al *student forum* del *Sixth European Dependable Computing Conference* (EDCC-6), Coimbra (Portugal), Octubre 2006.

- **[Gracia08]** Joaquín Gracia-Morán, **Luis-J. Saiz-Adalid**, J.-Carlos Baraza-Calvo, Daniel Gil-Tomás y Pedro-J. Gil-Vicente, *"Analysis of the Influence of Intermittent Faults in a Microcontroller"*, *11th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems* (DDECS'08), Bratislava (Eslovaquia), Abril 2008.

- **[Saiz08]** **Luis-J. Saiz-Adalid**, Joaquín Gracia-Moran, J.-Carlos Baraza-Calvo, Daniel Gil-Tomas y Pedro-J. Gil-Vicente, *"Applying Fault Injection to Study the Effects of Intermittent Faults"*, *fast abstract* presentado en el *7th European Dependable Computing Conference* (EDCC-7), Kaunas (Lituania), Mayo 2008.

- **[Saiz09]** **Luis-J. Saiz-Adalid**, *"Intermittent Faults: Analysis of Causes and Effects, New Fault Models, and Mitigation Techniques"*, presentado al *student forum* del *39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2009), Estoril (Portugal), Junio 2009.

- **[Saiz10a]** **Luis-J. Saiz-Adalid**, J.-Carlos Baraza-Calvo, Joaquín Gracia-Moran y Daniel Gil-Tomas, *"A Proposal of a Fault-Tolerant Mechanism for Microprocessor Buses"*, *fast abstract* presentado en el *8th European Dependable Computing Conference* (EDCC-8), Valencia (España), Abril 2010.

- **[Saiz10b]** **Luis-J. Saiz-Adalid**, J.-Carlos Baraza-Calvo, Joaquín Gracia-Moran, Daniel Gil-Tomas y Pedro Gil, *"A Proposal to Tolerate Intermittent Faults in Microprocessor Buses"*, *fast abstract* presentado en el *40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2010), Chicago (Illinois-EE.UU.), Junio 2010.

- **[Saiz13a]** **Luis-J. Saiz-Adalid**, Pedro Gil, Joaquín Gracia-Moran y J.-Carlos Baraza-Calvo, *"Using Interleaving to Avoid the Effects of Multiple Adjacent Faults in On-chip Interconnection Lines"*, *fast abstract* presentado en el *14th European Workshop on Dependable Computing* (EWDC2013), Coimbra (Portugal), Mayo 2013.

- **[Gracia14b]** Joaquín Gracia-Moran, **Luis-J. Saiz-Adalid**, Pedro-J. Gil-Vicente, Daniel Gil-Tomas y J.-Carlos Baraza-Calvo, *"Selectable Error Detection and Correction Levels Error Control Codes"*, *First Workshop on Innovation on Information and Communication Technologies* (ITACA-WIICT 2014), Valencia (España), Julio 2014.

5.4.4 Referencias a mis artículos

En este apartado se presentan trabajos de otros autores que citan mis artículos, anteriormente enumerados. Para ello se han utilizado, principalmente, búsquedas en páginas web de confianza y en otras fuentes (comprobando su veracidad):

- *IEEE Xplore Digital Library*: http://ieeexplore.ieee.org
- *Scopus*: http://www.scopus.com
- *Web of Science*: http://www.webofknowledge.com
- *ACM Digital Library*: http://dl.acm.org
- *Google Scholar*: http://scholar.google.es
- *ResearchGate*: http://www.researchgate.net

A continuación se muestran el número de referencias encontradas para cada uno de los artículos enumerados anteriormente, según las distintas búsquedas. Posteriormente, se listan las referencias más relevantes, indicando los artículos citados.

- [Saiz06]: se han encontrado dos referencias a este artículo.
- [Gracia08]: según *Google Scholar*, este artículo se ha referenciado en 22 ocasiones. Según *Scopus*, en 8. La *IEEE Xplore Digial Library* muestra 6 referencias, la *Web of Science* también 6, y la *ACM Digital Library* presenta 2 referencias. Finalmente, *ResearchGate* muestra 11 referencias.
- [Saiz08]: se han encontrado dos referencias a este artículo.
- [DGil08]: según *Google Scholar*, este artículo se ha referenciado en 10 ocasiones. Según *Scopus*, en 7. La *IEEE Xplore Digial Library* muestra 3 referencias y la *Web of Science* presenta 6. Finalmente, *ResearchGate* muestra 5 referencias.

- [Gracia10a]: según *Google Scholar*, este artículo se ha referenciado en 9 ocasiones. Según *Scopus*, en 6. La *IEEE Xplore Digial Library* muestra 6 referencias y la *Web of Science* presenta 4. Finalmente, *ResearchGate* muestra 7 referencias.

- [Gracia10b]: según *Google Scholar*, este artículo se ha referenciado en 4 ocasiones. Según *Scopus*, en 3. La *IEEE Xplore Digial Library* muestra 1 referencia y *ResearchGate* presenta 2.

- [DGil12a]: según *Google Scholar*, este artículo se ha referenciado en 7 ocasiones. Según *Scopus*, en 4. La *IEEE Xplore Digial Library* muestra 4 referencias y la *Web of Science* también 4. Finalmente, *ResearchGate* muestra 4 referencias.

- [DGil12b]: según *Google Scholar*, este artículo se ha referenciado en 5 ocasiones. Según *Scopus*, en 3. La *Web of Science* presenta 2 y *ResearchGate* 3 referencias.

- [Saiz13b]: según *Google Scholar*, este artículo se ha referenciado en 3 ocasiones, y *Scopus* muestra 1 referencia.

- [Gracia14a]: presenta una referencia, tanto en *Google Scholar* como en *Scopus* y en la *IEEE Xplore Digial Library*.

Los artículos más relevantes que citan a estos trabajos se listan a continuación:

- J. Pardo, "Contribución a la validación experimental no intrusiva de la confiabilidad en los sistemas empotrados basados en componentes COTS", Tesis Doctoral, Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, Valencia (España), Abril 2007.

  Artículo citado: [Saiz06].

- S. Pan, Y. Hu, X. Li, *"IVF: Characterizing the Vulnerability of Microprocessor Structures to Intermittent Faults"*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volumen 20, número 5, páginas 777–790, Mayo 2012.

  Artículo citado: [Gracia08].

- P.K. Lala, *"Transient and Permanent Fault Injection in VHDL Description of Digital Circuits"*, *Circuits and Systems*, volumen 3, número 2, páginas 192–199, Abril 2012.

  Artículos citados: [Gracia08, Saiz08].

- K.S. Yim, V. Sidea, Z. Kalbarczyk, D. Chen, R.K. Iyer, *"A Fault-Tolerant Programmable Voter for Software-based N-Modular Redundancy"*, en *Proceedings of the 2012 IEEE Aerospace Conference*, Big Sky (Montana-EE.UU.), Marzo 2012.

  Artículo citado: [Gracia10a].

- L. Rashid, *"Tolerating Intermittent Hardware Errors: Characterization, Diagnosis and Recovery"*, Tesis Doctoral, *Department of Electrical and Computer Engineering*, *University of British Columbia*, Vancouver (Canadá), Enero 2013.

  Artículos citados: [Gracia10a, DGil12a].

- Zhou D.H., Shi J.T., He X, *"Review of Intermittent Fault Diagnosis Techniques for Dynamic Systems"*, *Acta Automatica Sinica*, volumen 40, número 2, páginas 161–171, Febrero 2014.

  Artículos citados: [Gracia08, Gracia10b].

- R. Bakhshi, S. Kunche, M. Pecht, *"Intermittent Failures in Hardware and Software"*, *Journal of Electronic Packaging*, volumen 136, número 1, páginas 1–5, Marzo 2014.

  Artículo citado: [Gracia08].

- O.O. Karzova, *"Influence of contact network parameters on value of current rise speed during short circuit in power circuits of electric rolling stock"*, *Science and Transport Progress*, *Bulletin of Dnipropetrovsk National University of Railway Transport*, número 2(50), páginas 49–57, Abril 2014.

  Artículo citado: [DGil12b].

- C.S. Wang, Z.C. Fu, H.S. Chen, D.S. Wang, *"Characterizing the Effects of Intermittent Faults on a Processor for Dependability Enhancement Strategy"*, *The Scientific World Journal*, volumen 2014, identificador de artículo 286084, páginas 1–12, Abril 2014.

  Artículos citados: [Gracia08, DGil08, Gracia10a, DGil12a].

- A. Zammali, A. Bonneval, Y. Crouzet, *"A multi-function error detection policy to enhance communication integrity in critical embedded systems"*, *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion* (SERE-C), San Francisco (California-EE.UU.), Junio 2014.

  Artículo citado: [Saiz13b].

- M. Dimopoulos, Y. Gang, L. Anghel, M. Benabdenbi, N.E. Zergainoh, M. Nicolaidis, *"Fault-tolerant adaptive routing under an unconstrained set of node and link failures for many-core systems-on-chip"*, *Microprocessors and Microsystems*, volumen 38, número 6, páginas 620–635, Agosto 2014.

  Artículo citado: [DGil12a].

- D. Goswami, D. Müller-Gritschneder, T. Basten, U. Schlichtmann, S. Chakraborty, *"Fault-tolerant Embedded Control Systems for Unreliable Hardware"*, *2014 14th International Symposium on Integrated Circuits* (ISIC), Marina Bay Sands (Singapur), Diciembre 2014.

  Artículo citado: [Gracia14a].

- G. Yalcin, *"Designs for Increasing Reliability While Reducing Energy and Increasing Lifetime"*, Tesis Doctoral, *Department of Computer Architecture*, Universitat Politècnica de Catalunya, Barcelona (España), Diciembre 2014.

  Artículo citado: [Gracia08].

- L. Rashid, K. Pattabiraman, S. Gopalakrishnan, *"Characterizing the Impact of Intermittent Hardware Faults on Programs"*, *IEEE Transactions on Reliability*, volumen 64, número 1, páginas 297–310, Marzo 2015.

  Artículo citado: [DGil12a].

- A. Eghbal, P.M. Yaghini, N. Bagherzadeh, M. Khayambashi, *"Analytical Fault Tolerance Assessment and Metrics for TSV-based 3D Network-on-Chip"*, aceptado para su publicación en *IEEE Transactions on Computers* (doi: 10.1109/TC.2015.2401016).

  Artículo citado: [DGil12a].

Bibliografía

[Aidemark01] "GOOFI: Generic Object-Oriented Fault Injection Tool"; J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson; Procs. 2001 International Conference on Dependable Systems and Networks (DSN 2001), pp. 83-88, Göteborg (Suecia), Julio 2001.

[Akerlund06] "ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects"; O. Akerlund y otros; Proc. European Congress on Embedded Real-Time Software (ERTS), Toulouse (Francia), Enero 2006.

[Alderighi03] "A Tool for Injecting SEU-like Faults into the Configuration Control Mechanism of Xilinx Virtex FPGAs"; M.Alderighi, F. Casini, S. D'Angelo, M. Mancini, A. Marmo, S. Pastore, G.R. Sechi; Procs. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03), pp. 71-78, Boston, (Massachusetts, EE.UU.), Noviembre 2003.

[Amendola96] "Fault Behavior Observation of a Microprocessor System through a VHDL Simulation-Based Fault Injection Experiment"; A.M. Amendola, A. Benso, F. Corno, L. Impagliazzo, P. Marmo, P. Prineto, M. Rebaudengo, M. Sonza Reorda; in Procs. European Design Automation Conference with EURO-VHDL, (EURO-DAC with EURO-VHDL – EURO-DAC'96), Génova, Suiza, Septiembre 1996.

[Amerasekera97] "Failure Mechanisms in Semiconductor Devices"; E.A. Amerasekera y F.N. Najm; 2nd edition, John Wiley & Sons, 1997.

[Arlat90] "Fault injection for dependability validation: a methodology and some applications"; J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, D. Powell; IEEE Transactions on Software Engineering, 16(2):166-182, Febrero 1990.

[Arlat93] "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems"; J. Arlat, A. Costes, Y. Crouzet, J. Laprie, D. Powell; IEEE Transactions on Computers, 42(8):913-923, 1993.

[Armstrong89] "Chip-Level Modelling with VHDL"; J.R. Armstrong; Prentice Hall, 1989.

[Armstrong92] "Test generation and Fault Simulation for Behavioral Models"; J.R. Armstrong, F.S. Lam, P.C. Ward; en Performance and Fault Modelling with VHDL. (J.M.Schoen ed.), pp.240-303, Englewood Cliffs, Prentice-Hall, 1992.

[Asadi07] "Fast co-verification of HDL models"; G. Asadi, S.G. Miremadi, A. Ejlali; Microelectronic Engineering, 84(2):218–228, Febrero 2007.

[Avizienis04] "Basic Concepts and Taxonomy of Dependable and Secure Computing", A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr, en IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, January-March 2004.

[Aylor90] "A Fundamental Approach to Uninterpreted/Interpreted Modeling of Digital Systems in a Common Simulation Environment"; J.H. Aylor, R.D. Williams, R. Waxman, B.W. Johnson, R.L. Blackburn; Technical Report 900724.0, University of Virginia, 1990.

[Baeg09] "SRAM interleaving distance selection with a soft error failure model"; S. Baeg, S. Wen, R. Wong; IEEE Transactions on Nuclear Science, vol. 56, no. 4, pp. 2111–2118, Agosto 2009.

[Baldini03] "BOND: An Agents-Based Fault Injector for Windows NT"; A. Baldini, A. Benso, P. Prinetto; en Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Edited by Alfredo Benso and Paolo Prinetto, Kluwer Academic Publishers, pp. 111-123, Octubre 2003.

[Ball69] "Effects and detection of intermittent failures in digital systems"; M. Ball, F. Hardie; Proceedings of the ACM fall joint computer conference, pp. 329-335, Noviembre 1969.

[Baraza00] "A Prototype of a VHDL-Based Fault Injection Tool"; J.C. Baraza, J. Gracia, D. Gil, P.J. Gil; IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2000), pp. 396-404, Yamanashi, Japan, Octubre 2000.

[Baraza02] "A Prototype of a VHDL-Based Fault Injection Tool. Description and Application"; J.C. Baraza, J. Gracia, D. Gil, P.J. Gil; Journal of Systems Architecture, 47(10):847-867, 2002.

[Baraza03] "Contribución a la Validación de Sistemas Complejos Tolerantes a Fallos en la Fase de Diseño. Nuevos Modelos de Fallos y Técnicas de Inyección de Fallos"; J.C. Baraza; *Tesis doctoral*, Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, Octubre 2003.

[Baraza08] "Enhancement of fault injection techniques based on the modification of VHDL code", J.C. Baraza, J. Gracia, S. Blanc, D. Gil, P.J. Gil; IEEE Transactions on VLSI Systems, 16(6), pp. 693-706, Junio 2008.

[Barros04] "Modeling and Simulation of Time Domain Faults in Digital Systems", D. Barros Jr, F. Vargas, M.B. Santos, I.C. Teixeira, and J.P. Teixeira, Procs. 10th IEEE International On-Line Testing Symposium (IOLTS'04), pp. 5-10, Portugal, July 2004.

[Baumann05] "Radiation-induced soft errors in advanced semiconductor technologies"; R.C. Baumann, IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 305–316, Septiembre 2005.

[Benso99] "FlexFi: A Flexible Fault Injection Environment for Microprocessor-Based Systems"; A. Benso, M. Rebaudengo, M. Sonza Reorda; Procs. 18th International Conference on Computer Safety, Reliability and Security (SAFECOMP'1999), pp. 323-335, Toulouse (Francia), Septiembre 1999.

[Benso03] "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation"; A. Benso, P. Prinetto (editores), Kluwer Academic Publishers, pp. 159-176, Octubre 2003.

[Birner09] "ARROW - A Generic Hardware Fault Injection Tool for NoCs"; M. Birner, T. Handl; 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD '09), pp. 465–472, Patras (Grecia), Agosto 2009.

[Bolchini08] "Fault models and injection strategies in SystemC specifications", C. Bolchini, A. Miele, D. Sciuto; 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD), pp. 88–95, 2008.

[Borkar03] "Parameter Variations and Impact on Circuits and Microarchitecture", S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, V. De; Procs. 40th Conference on Design Automation (DAC 2003), pp. 338 – 342, April 2003.

[Boué98] "MEFISTO-L: A VHDL-Based Fault Injection Tool for the Experimental Assessment of Fault Tolerance"; J. Boué, P. Pétillon, Y. Crouzet; Procs. 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 168-173. Munich, Alemania, Junio 1998.

[Bouricius69] "Reliability modeling techniques for self-repairing computer systems"; W. Bouricius, W. Carter, P. Schneider; Procs. 24th ACM National Conference, pp. 295-309, 1969.

[Campelo99a] "Design and Validation of a Distributed Industrial Control System's Nodes", J.C. Campelo, F. Rodriguez, P.J. Gil, J.J. Serrano; Proc. 18th IEEE Symposium on Reliable Distributed Systems, pp. 300-301, 1999.

[Campelo99b] "Diseño y Validación de Nodos de Proceso Tolerantes a Fallos de Sistemas Industriales Distribuidos"; José Carlos Campelo; *Tesis doctoral*, Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, Junio 1999.

[Carreira95] "Xception: Software fault injection and monitoring in processor functional units"; J. Carreira, H. Madeira, J.G. Silva; Procs. 5th Working Conference on Dependable Computing for Critical Applications (DCCA-5), pp. 135-148, Urbana-Champaign (Illinois, EE.UU.), Setiembre 1995.

[Carreira98] "Xception: a Technique for the Experimental Evaluation of Dependability in Modern Computers"; J. Carreira, H. Madeira, J.G. Silva; IEEE Transactions on Software Engineering, 24(2):125-136, Febrero 1998.

[Cheng95] "Fault Emulation: A New Approach to Fault Grading"; K.T. Cheng, S.Y. Huang, W.J. Dai; Procs. 1995 International Conference on Computer-Aided Design (ICCAD'95), pp. 681-686, 1995.

[Civera01] "FPGA-Based Fault Injection for Microprocessor Systems"; P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante; Procs. 10th Asian Test (ATS 2001*)*, pp. 304-312, Kyoto, Japón, Noviembre 2001.

[Clark92] "REACT: Reliable Architecture Characterization Tool"; J.A. Clark, D.K. Pradhan; Technical Report TR-92-CSE-22, University of Massachusetts, Junio 1992.

[Constantinescu02] "Impact of Deep Submicron Technology on Dependability of VLSI Circuits"; C. Constantinescu; Procs. International 2002 Conference on Dependable Systems and Networks (DSN'02), pp. 205-209, Washington (D.C., EE.UU.), Junio 2002.

[Constantinescu03] "Trends and challenges in VLSI circuit reliability"; C. Constantinescu; IEEE Micro, vol. 23, nº 4, pp. 14-19, Julio 2003.

[Constantinescu05] "Dependability Benchmarking using Environmental Test Tools", C. Constantinescu; Procs. Reliability and Maintainability Symposium (RAMS 2005), pp. 567–571, Enero 2005.

[Constantinescu06] "Intermittent Faults in VLSI Circuits", C. Constantinescu; 2nd Workshop on Silicon Errors in Logic - System Effects (SELSE2), Abril 2006.

[Constantinescu07] "Impact of Intermittent Faults on Nanocomputing Devices", C. Constantinescu; WDSN-07, Edinburgh, UK, Junio 2007. http://www.laas.fr/WDSN07.

[Constantinescu08] "Intermittent faults and effects on reliability of integrated circuits"; C. Constantinescu; Reliability and Maintainability Symposium (RAMS 2008), pp. 370–374, Las Vegas (Nevada-EE.UU.), Enero 2008.

[Courbon15] "Combining image processing and laser fault injections for characterizing a hardware AES"; F. Courbon, J. Fournier, P. Loubet-Moundi, A. Tria; aceptado para publicación en IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015.

[Damm88] "Experimental Evaluation of error detection and self checking coverage of components of a distributed real-time systems"; A. Damm; Tesis doctoral, Universität Wien, Viena, Austria, Octubre 1988.

[DBench01] "Preliminary Dependability Benchmark Framework", Deliverable CF2 of Dependability Benchmarking Project (DBench), IST-2000-25425, Agosto 2001. Available online at: http://www.laas.fr/dbench.

[DBench02] "Fault Representativeness", Deliverable ETIE2 of Dependability Benchmarking Project (DBench), IST-2000-25425, Julio 2002. Disponible online en: http://www.laas.fr/dbench.

[DaSilva07] "Exhaustif: A Fault Injection Tool for Distributed Heterogeneous Systems"; A. DaSilva, J.F. Martínez, L. López, A.B. García, L. Redondo; Euro American conference on Telematics and information systems, artículo nº 17, pp. 1–8, Faro (Portugal), Mayo 2007.

[deAndrés08] "Speeding-up model-based fault injection of deep-submicron CMOS fault models through dynamic and partially reconfigurable FPGAS"; D. de Andrés; *Tesis doctoral*, Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, Valencia (España), Mayo 2008.

[DeLong96] "A Fault Injection Technique for VHDL Behavioural-Level Models"; T.A. DeLong, B.W. Johnson, J.A. Profeta III; IEEE Design & Test of Computers, 13(4): 24-33, Winter 1996.

[Dell97] "A White Paper on the Benefits of Chipkill-correct ECC for PC Server Main Memory";T.J. Dell; IBM Microelectronics Division, Noviembre 1997.

[Dewey92] "VHDL: Toward a Unified View of Design"; A. Dewey, A.J. de Geus; IEEE Design and Test of Computers, 9(2):8-17, Abril/Junio 1992.

[DGil99] "Validación de Sistemas Tolerantes a Fallos mediante Inyección de Fallos en Modelos VHDL"; Daniel Gil Tomás; *Tesis doctoral*, Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, Julio 1999.

[DGil03] "VHDL Simulation-Based Fault Injection Techniques"; D. Gil, J.C. Baraza, J. Gracia, P.J. Gil; en Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Edited by Alfredo Benso and Paolo Prinetto, Kluwer Academic Publishers, pp. 159-176, Octubre 2003.

[DGil04] "Analysis of the influence of processor hidden registers on the accuracy of fault injection techniques"; D. Gil, J. Gracia, J.C. Baraza, P.J. Gil; 9th IEEE International High-Level Design Validation and Test Workshop (HLDVT'04), pp. 173–178, Sonoma Valley (California-EE.UU.), Noviembre 2004.

[DGil08] "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", D. Gil-Tomás, LJ. Saiz-Adalid, J. Gracia-Morán, JC. Baraza-Calvo, P. Gil-Vicente; Procs. IEEE International High Level Design Validation and Test Workshop 2008 (HLDVT'08), Incline Village (Nevada-USA), Noviembre 2008.

[DGil12a] "Analyzing the impact of intermittent faults on microprocessors applying fault injection"; D. Gil-Tomás, J. Gracia-Morán, J.C. Baraza-Calvo, L.J. Saiz-Adalid, P.J. Gil-Vicente; IEEE Design and Test of Computers, volumen 29, número 6, páginas 66–73, Noviembre–Diciembre 2012.

[DGil12b] "Studying the effects of intermittent faults on a microcontroller"; D. Gil-Tomás, J. Gracia-Morán, J.C. Baraza-Calvo, L.J. Saiz-Adalid, P.J. Gil-Vicente; Microelectronics Reliability, volumen 52, número 11, páginas 2387–2846, Noviembre 2012.

[Dugan89] "Coverage modelling for dependability analysis of fault-tolerance systems"; J.B. Dugan, K.S. Trivedi; IEEE Transactions on Computers, 38(6):775-787, Junio 1989.

[Dupuy90] "NEST: A Network Simulation and Prototyping Testbed"; A. Dupuy, J. Schwartz, Y. Yemini, D. Bacon; Communications of the ACM, 33(10):64-74, Octubre 1990.

[Duraes03] "Definition of Software Fault Emulation Operators: a Field Data Study", J. Duraes, H. Madeira; Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03), pp. 105-114, San Francisco (California-EE.UU.), Junio 2003.

[Dutta07] "Reliable network-on-chip using a low cost unequal error protection code"; A. Dutta, N.A. Touba; 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'07), pp. 3–11, Roma (Italia), Septiembre 2007.

[Echtle92] "The EFA fault injector for fault tolerant distributed system testing"; K. Echtle, M. Leu; Procs. IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp. 28-35, Amherst, EE.UU., Julio 1992.

[Ejlali03] "A Hybrid Fault Injection Approach Based on Simulation and Emulation Co-operation"; A. Ejlali, S.G. Miremadi, H. Zarandi, G. Asadi, S.B. Sarmadi; Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03), pp. 479–488, San Francisco (California-EE.UU.), Junio 2003.

[Folkesson98] "A comparison of simulation based and scan chain implemented fault injection"; P. Folkesson, S. Svensson, J. Karlsson; in Procs. 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 284-293, Munich, Alemania, Junio 1998.

[Friesenbichler10] "A deterministic approach for hardware fault injection in asynchronous QDI logic"; W. Friesenbichler, T. Panhofer, A. Steininger; 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 317,322, Viena, Austria, Abril 2010.

[Fujiwara06] "Code Design for Dependable Systems"; E. Fujiwara; John Wiley & Sons, 2006.

[Ghosh91] "On behavior fault modeling for digital design"; S. Ghosh, T.J. Chakraborty; Journal of Electronic Testing: Theory and Applications, nº 2, pp. 135-151, 1991.

[Golay49] "Notes on Digital Coding"; M.J.E. Golay; Proceedings of the Institute of Radio Engineers (IRE), vol. 37, p. 657, Junio 1949.

[Goswami92] "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis"; K.K. Goswami, R.K. Iyer; Technical Report CRHC 92-11, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana (Illinois, EE.UU.), Junio 1992.

[Goswami97] "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis"; K.K. Goswami, R.K. Iyer, L. Young; IEEE Transactions on Computers, 46(1):60-74, Enero 1997.

[Gracia01] "Comparison and Application of different VHDL-Based Fault Injection Techniques"; J. Gracia, J.C. Baraza, D. Gil, P.J. Gil; Procs. 2001 IEEE International. Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2001), pp. 233-241, San Francisco (EE.UU.), Octubre 2001.

[Gracia08] "Analysis of the influence of intermittent faults in a microcontroller"; J. Gracia-Morán, LJ. Saiz-Adalid, JC. Baraza-Calvo, D. Gil-Tomás, P. Gil-Vicente; Procs. 11 th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'08), Bratislava (Eslovaquia), Abril 2008.

[Gracia10a] "Experimental Validation of a Fault Tolerant Microcomputer System against Intermittent Faults"; J. Gracia-Morán, D. Gil-Tomás, L.J. Saiz-Adalid, J.C. Baraza-Calvo, P.J. Gil-Vicente; 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010), Chicago (Illinois-EE.UU.), Junio 2010.

[Gracia10b] "Searching Representative and Low Cost Fault Models for Intermittent Faults in Microcontrollers: A Case Study"; J. Gracia-Moran, D. Gil-Tomas, J.C. Baraza-Calvo, L.J. Saiz-Adalid, P.J. Gil-Vicente; 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2010), Tokio (Japón), Diciembre 2010.

[Gracia13] "Defining a Representative and Low Cost Fault Model Set for Intermittent Faults in Microprocessor Buses"; J. Gracia-Moran, D. Gil-Tomas, L.J. Saiz-Adalid, J.C. Baraza-Calvo, P.J. Gil-Vicente; Sixth Latin-American Symposium on Dependable Computing (LADC 2013), Río de Janeiro (Brasil), Abril 2013.

[Gracia14a] "Effects of Intermittent Faults on the Reliability of a Reduced Instruction Set Computing (RISC) Microprocessor"; J. Gracia-Moran, J.C. Baraza-Calvo, D. Gil-Tomas, L.J. Saiz-Adalid, P.J. Gil-Vicente; IEEE Transactions on Reliability, volumen 63, número 1, páginas 144–153, Marzo 2014.

[Gracia14b] "Selectable Error Detection and Correction Levels Error Control Codes"; J. Gracia-Moran, L.J. Saiz-Adalid, P.J. Gil-Vicente, D. Gil-Tomas, J.C. Baraza-Calvo; First Workshop on Innovation on Information and Communication Technologies (ITACA-WIICT 2014), Valencia (España), Julio 2014.

[Hamming50] "Error detecting and error correcting codes"; R.W. Hamming; Bell System Technical Journal, vol. 29, pp. 147–160, Abril 1950.

[Han95] "DOCTOR: An IntegrateD Software Fault InjeCTiOn EnviRonment for Distributed Real-Time Systems", S. Han, K.G. Shin, H.A. Rosenberg; Proc. IEEE mt. Symp. Computer Performance and Dependability, 1995, pp. 204-213.

[Hardie67] "Design and Use of Fault Simulation for Saturn Computer Design"; F.H. Hardie, R.J. Suhocki; IEEE Transactions on Electronic Computers, vol. EC-16, número 4, Agosto 1967.

[Hawkins00] "CMOS IC Failure Mechanism and Defect Based Testing"; C. Hawkins; 2nd Summer Course on Selected Microelectronic Design & Test Topics, Palma de Mallorca, Julio 2000.

[Hsiao70] "A class of optimal minimum odd-weight column SEC-DED codes"; M.Y. Hsiao; IBM Journal of Research and Development, vol. 14, no. 4, pp. 395–401, Julio 1970.

[Hsueh97] "Fault Injection Techniques and Tools"; M. Sueh, T. Tsai, R.K. Iyer; IEEE Computer, 20(4):75-82, Abril 1997.

[Hwang98] "Sequential Circuit Fault Simulation Using Logic Emulation"; S.A. Hwang, J.H. Hong, C.W. Wu; IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17(8):724-736, Agosto 1998.

[IEC10] IEC/EN 61508: International standard 61508 functional safety: safety related systems; International Electrotechnical Commission; Second Edition, 2010.

[IEEE93] IEEE Standard VHDL Language Reference Manual; IEEE Std 1072-1993.

[IEEE95] IEEE Standard Verilog Language Reference Manual, IEEE Std. 1364-1995.

[IEEE05] IEEE Standard SystemC Language Reference Manual, IEEE Std. 1666-2005.

[ISO11] ISO 26262 Road vehicles-Function Safety; International Organization for Standardization, 2011.

[Islam14] "Binary-Level Fault Injection for AUTOSAR Systems"; M.M. Islam, N.M. Karunakaran, J. Haraldsson, F. Bernin, J. Karlsson; Tenth European Dependable Computing Conference (EDCC 2014), pp. 138–141, Newcastle (Reino Unido), Mayo 2014.

[Iyer95] "Experimental Evaluation"; R.K. Iyer; Procs. 25th International Symposium on Fault-Tolerant Computing (FTCS-25) – Special Issue, pp. 115-132, Pasadena (California, EE.UU.), Junio 1995.

[Jeitler09] "FuSE - a hardware accelerated HDL fault injection tool"; M. Jeitler, M. Delvai, S. Reichor; 5th Southern Conference on Programmable Logic, pp. 89–94, San Carlos, Brasil, Abril 2009.

[Jenn94a] "Sur la validation des systémes tolérant les fautes: injection de fautes dans de modéles de simulation VHDL"; Eric Jenn; Thése; Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS); LAAS Report nº 94-361; 1994.

[Jenn94b] "Fault injection into VHDL models: the MEFISTO tool"; E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, J. Karlsson; Procs. 24th Int. Symposium on Fault-Tolerant Computing (FTCS-24), pp. 356-363. Austin, (Texas, EE.UU.), Junio 1994.

[Kanawati95] "FERRARI: A flexible software based fault and error injection system"; G.A. Kanawati, N.A. Kanawati, J.A. Abraham; IEEE Transactions on Computers, 44(2):248-260, Febrero 1995.

[Kanoun89] "Croissance de la Sûrété de fonctionement des logiciels. Caracterisation – Modelisation – Evaluation"; K. Kanoun; Thése présentée a L'Institut National Polytechnique de Toulouse, Septiembre 1989.

[Kao93] "FINE: a fault injection and monitoring environment for tracing UNIX system behaviour under faults"; W. Kao. R.K. Iyer, D. Tang; IEEE Transactions on Software Engineering, 19(11):1105-1118, Noviembre 1993.

[Kao94] "DEFINE: a distributed fault injection and monitoring environment"; W. Kao, R.K. Iyer; Workshop on Fault Tolerant Parallel and Distributed Systems, Junio 1994.

[Karlsson95] "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber and J. Reisinger, 5th IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-5), pp. 267-287, Urbana Champaign, (Illinois, EE.UU.), Septiembre, 1995

[Kranitis06] "Optimal Periodic Testing of Intermittent Faults In Embedded Pipelined Processor Applications", N. Kranitis, A. Merentitis, N. Laoutaris, G. Theodorou; Design, Automation and Test in Europe (DATE'06), Vol. 1, pp. 1-6, Marzo 2006.

[Kropp98] "Automated Robustness Testing of Off-the-Shelf Software Components", N. Kropp, P. J. Koopman, D. P. Siewiorek; Proc. of the 28th IEEE Int. Symposium on Fault Tolerant Computing, pp. 230–239, Munich, Germany, 1998.

[Lajolo00] "Evaluating System Dependability in a Co-Design Framework"; M. Lajolo, M. Rebaudengo, M. Sonza Reorda, M. Violante, L. Lavagno; Procs. 2000 Design, Automation and Test in Europe (DATE 2000), pp. 586-590, París, Francia, Marzo 2000.

[Laprie92] "Dependability, Basic Concepts and Terminology"; J.C. Laprie; Springer-Verlag, 1992.

[Lee09] "FleXilicon Architecture and Its VLSI Implementation"; J.S. Lee, D.S. Ha; IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volumen 17, número 8, pp. 1021–1033, Agosto 2009.

[Leveugle01] "A Low-Cost Hardware Approach to Dependability Validation of IPs"; R. Leveugle; Procs. 2001 International Symposium on Defect and Fault Tolerance (DFT'01), pp. 242-249, San Francisco (California, EE.UU.), Octubre 2001.

[Lin04] "Error Control Coding (Second Edition)"; S. Lin, D.J. Costello; Prentice-Hall, 2004.

[Madeira94] "RIFLE: A general purpose pin-level fault injector"; H. Madeira, M. Rela, F. Moreira, J.G Silva; Procs. 1st European Dependable Computing Conference (EDCC-1), pp 199-216, Berlín, Alemania, Octubre 1994.

[Madeira00] "On the emulation of software faults by software fault injection", H. Madeira, D. Costa, M. Vieira; Proceedings of the IEEE International Conference on Dependable Systems and Networks, pages 417-426, New York, NY, EE.UU., Junio 2000.

[Masnick67] "On linear unequal error protection codes"; B. Masnick, J. Wolf; IEEE Transactions on Information Theory, volumen 13(4), pp. 600–607, Octubre 1967.

[MC8051-01] http://www.oregano.at

[McPherson06] "Reliability challenges for 45nm and beyond", J.W. McPherson; Procs. Conference on Design Automation (DAC 2006), pp. 176–181, Julio 2006.

[Miczo90] "VHDL as a Modeling-for-Testability Tool"; A. Miczo; Procs. 35th Computer Society International Conference (COMPCON'90), pp.403-409, San Francisco (California, EE.UU.), Febrero-Marzo 1990.

[Misera07] "Fault injection techniques and their accelerated simulation in SystemC", S. Misera, H. T. Vierhaus, A. Sieber; 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD), pp. 587–595, 2007.

[Model01] "ModelSim SE/User's Manual, Version 5.5e", Model Technology, 24 Septiembre 2001.

[Monnet06] "Designing resistant circuits against malicious faults injection using asynchronous logic"; Y. Monnet, M. Renaudin, R. Leveugle; IEEE Transactions on Computers, vol. 55, no. 9, pp. 1104–1115, Septiembre 2006.

[Moore00] "Delay-fault testing and defects in deep sub-micron ICs-does critical resistance really mean anything?", W. Moore, G. Gronthoud, K. Baker, M. Lousberg; Procs. IEEE International Test Conference 2000, pp. 95-104, Atlantic City, NJ, USA, Octubre 2000.

[Neale13] "A New SEC-DED Error Correction Code Subclass for Adjacent MBU Tolerance in Embedded Memory"; A. Neale, M. Sachdev; IEEE Transactions on Device and Materials Reliability, volumen 13, número 1, pp. 223–230, Marzo 2013.

[Neubauer07] "Coding Theory: Algorithms, Architectures and Applications"; A. Neubauer, J. Freudenberger, V. Kühn; John Wiley & Sons, 2007.

[Nightingale11] "Cycles, cells and platters: an empirical analysis of hardware failures on a million consumer PCs"; E.B. Nightingale, J.R. Douceur, V. Orgovan; Sixth conference on Computer systems (EuroSys'11), pp. 343–356, Salzburgo (Austria), Abril 2011.

[Ohlsson92] "A Study of the Effect of Transient Fault Injection into a 32-bit RISC with Built-in Watchdog"; J. Ohlsson, M. Rimén, U. Gunneflo; Procs. 22nd Int. Symposium on Fault-Tolerant Computing (FTCS-22), pp. 316-325, Boston, EE.UU., Julio 1992.

[PGil92] "Sistema Tolerante a Fallos con Procesador de Guardia: Validación mediante Inyección Física de Fallos"; P.J. Gil; Tesis doctoral, Departamento de Ingeniería de Sistemas, Computadores y Automática (DISCA), Universidad Politécnica de Valencia, Septiembre 1992.

[PGil97] "High speed fault injector for safety validation of industrial machinery"; P.J. Gil, J.C. Baraza, D. Gil, J.J. Serrano; Procs. 8th European Workshop of Dependable Computing (EWDC-8): Experimental validation of dependable systems, Göteborg, Suecia, Abril 1997.

[PGil06] "Computación Confiable y Segura: Conceptos Básicos y Taxonomía", P. Gil; Informe interno, DISCA. Adaptación artículo de Avizienis, A., Laprie, J.C., Randell, B. and Landwehr, C., "Basic Concepts and Taxonomy of Dependable and Secure Computing". IEEE transactions on dependable and secure computing, vol. 1, no. 1, pp. 11-33, January-March 2004. Valencia, Septiembre 2006.

[Pan10] "IVF: Characterizing the vulnerability of microprocessor structures to intermittent faults"; S. Pan, Y. Hu, X. Li; Design, Automation & Test in Europe Conference (DATE 2010), pp. 238–243, Dresde (Alemania), Marzo 2010.

[Park09] "Reliability implications of bias-temperature instability in digital ICs"; S.P. Park, K. Roy; IEEE Design and Test of Computers, volumen 26, número 6, pp. 8–17, Noviembre 2009.

[Plasma01] Plasma CPU model. [Online]. Disponible en: http://opencores.org/project,plasma (accedido el 17/05/2015).

[Plasmaw01] Servidor web sobre CPU Plasma. [Online]. Disponible en: http://plasmacpu.no-ip.org:8080/ (accedido el 17/05/2015).

[Pradhan96] "Fault-Tolerant Computer System Design"; D.K. Pradhan; Prentice-Hall; 1996.

[Rashid10] "Modeling the Propagation of Intermittent Hardware Faults in Programs"; L. Rashid, K. Pattabiraman, S. Gopalakrishnan; 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2010), Tokio (Japón), Diciembre 2010.

[Reed60] "Polynomial Codes over Certain Finite Fields"; I.S. Reed, G. Solomon; Journal of the Society for Industrial and Applied Mathematics, volumen 8, número 2, pp. 300–304, June 1960.

[Reiger60] "Codes for the correction of clustered errors"; S.H. Reiger; IRE Transactions on Information Theory, vol. IT-6, no. 1, pp. 16-21, Marzo 1960.

[Reviriego10] "Protection of Memories Suffering MCUs through the Selection of the Optimal Interleaving Distance"; P. Reviriego, J.A. Maestro, S. Baeg, S. Wen, R. Wong; IEEE Transactions on Nuclear Science, vol. 57, no. 4(1), pp. 2124–2128, Agosto 2010.

[Rodder95] "A Scaled 1.8V, 0.18μm Gate Lenght CMOS Technology: Device Design and Reliability Considerations"; M. Rodder, S. Aur, C. Chen; Technical Digest International Electron Devices Meeting (IEDM), pp. 415-418, Washington (D.C., EE.UU.), Diciembre 1995.

[Rodriguez99] "MAFALDA: Microkernel Assessment by Fault Injection and Design Aid", M. Rodriguez, F. Salles, J.C. Fabre, J. Arlat; Proc. European Dependable Computing Conference (EDCC-3), 1999, pp. 143-160.

[Saiz06] "Fast and Early Validation of VLSI Systems"; L.J. Saiz-Adalid; Sixth European Dependable Computing Conference (EDCC-6), Coimbra (Portugal), Octubre 2006.

[Saiz08] "Applying Fault Injection to Study the Effects of Intermittent Faults"; LJ. Saiz-Adalid, J. Gracia-Morán, JC. Baraza-Calvo, D. Gil-Tomás, P. Gil-Vicente; Procs. 7th European Dependable Computing Conference (EDCC-7), Kaunas (Lituania), Mayo 2008.

[Saiz09] "Intermittent Faults: Analysis of Causes and Effects, New Fault Models, and Mitigation Techniques"; Luis-José Saiz-Adalid; Procs. 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009); Estoril (Portugal), Junio 2009.

[Saiz10a] "A Proposal of a Fault-Tolerant Mechanism for Microprocessor Buses"; L.J. Saiz-Adalid, J.C. Baraza-Calvo, J. Gracia-Moran, D. Gil-Tomas; 8th European Dependable Computing Conference (EDCC-8), Valencia (España), Abril 2010.

[Saiz10b] "A Proposal to Tolerate Intermittent Faults in Microprocessor Buses"; L.J. Saiz-Adalid, J.C. Baraza-Calvo, J. Gracia-Moran, D. Gil-Tomas, P. Gil; 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010), Chicago (Illinois-EE.UU.), Junio 2010.

[Saiz13a] "Using Interleaving to Avoid the Effects of Multiple Adjacent Faults in On-chip Interconnection Lines"; L.J. Saiz-Adalid, P. Gil, J. Gracia-Moran, J.C. Baraza-Calvo;.

[Saiz13b] "Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels"; L.J. Saiz-Adalid, P.J. Gil-Vicente, J.C. Ruiz-García, D. Gil-Tomas, J.C. Baraza-Calvo, J. Gracia-Moran; .

[Saiz14] "Modified Hamming Codes to Enhance Short Burst Error Detection in Semiconductor Memories"; L.J. Saiz-Adalid, P. Gil, J.C. Baraza-Calvo, J.C. Ruiz, D. Gil-Tomas, J. Gracia-Moran; .

[Saiz15] "MCU Tolerance in SRAMs through Low Redundancy Triple Adjacent Error Correction"; L.J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli, J.A. Maestro; IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volumen 23, número 10, páginas 2332–2336, Octubre 2015.

[Saleh90] "Reliability of Scrubbing Recovery-Techniques for Memory Systems"; A.M. Saleh, J.J. Serrano, J.H. Patel; IEEE Transactions on Reliability, volumen 39, número 1, pp. 114-122, Abril 1990.

[Samson98] "A Technique for Automated Validation of Fault Tolerant Designs Using Laser Fault Injection (LFI)"; J.R. Samson Jr., W. Moreno, F.J. Falquez; Procs. 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 162-167, Munich, Alemania, Junio 1998.

[Sanchez12] "Enhanced Detection of Double and Triple Adjacent Errors in Hamming Codes through Selective Bit Placement"; A. Sanchez-Macian, P. Reviriego, J.A. Maestro; IEEE Transactions on Device and Materials Reliability, vol. 12(2), pp. 357–362, Junio 2012.

[Sanchez14] "Hamming SEC-DAED and Extended Hamming SEC-DED-TAED codes through selective shortening and bit placement"; A. Sanchez-Macian, P. Reviriego, J. A. Maestro; IEEE Transactions on Device and Materials Reliability, vol. 14(1), pp. 574–576, Marzo 2014.

[Santos03] "Constraints on the use of Boundary Scan for Fault Injection"; L.E. Santos, M.Z. Rela; Proc Latin American Symposium on Dependable Computing, LADC 2003, Sao Paulo, 2003.

[Schroder07] "Negative bias temperature instability: What do we understand?"; D.K. Schroder; Microelectronics Reliability, volumen 47, número 6, pp. 841–852, Junio 2007.

[Schroeder09] "DRAM Errors in the Wild: A Large-Scale Field Study"; B. Schroeder, E. Pinheiro, W.D. Weber; Eleventh ACM international joint conference on Measurement and modeling of computer systems (SIGMETRICS), pp. 193-204, Seattle (Washington-EE.UU.), Junio 2009.

[Segall88] "FIAT–Fault Injection based Automated Testing Environment"; Z. Segall, D. Vrsalovic, D. Soewoprek, D. Yaskin, J. Kownavki, J. Barton, D. Rancey, A. Robinson, T. Lin; Procs. 18th International Symposium on Fault-Tolerant Computing (FTCS-18), pp. 102-107, Tokio, Japón, Junio 1988.

[Shamshiri10] "Error-Locality-Aware Linear Coding to Correct Multi-bit Upsets in SRAMs"; S. Shamshiri, K.T. Cheng; IEEE International Test Conference (ITC 2010), paper 7.1, pp. 1–10, Austin (Texas-EE.UU.), Noviembre 2010.

[She12] "SEU Tolerant Memory Using Error Correction Code"; X. She, N. Li, D.W. Jensen; IEEE Transactions on Nuclear Science, vol. 59, no. 1, pp. 205–210, Febrero 2012.

[Sieh97] "VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions"; V. Sieh, O. Tschäche, F. Balbach; Procs. 27th Annual International Symposium on Fault Tolerant Computing (FTCS-27), pp. 32-36, Seatle, (Washington, EE.UU.), Junio 1997.

[Siewiorek92] "Reliable Computer Systems. Design and Evaluation"; D.P. Siewiorek, R.S. Swarz; 2ª ed., Digital Press, 1992.

[Siewiorek98] "Reliable Computer Systems. Design and Evaluation"; D.P. Siewiorek, R.S. Schwarz; 3ª ed., Digital Press, 1998.

[Smolens07] "Detecting Emerging Wearout Faults", J.C. Smolens, B.T. Gold, J.C. Hoe, B. Falsafi, K. Mai; 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), Abril 2007.

[Sridhara05] "Coding for System-on-Chip Networks: A Unified Framework"; S.R. Sridhara, N.R. Shanbhag; IEEE Transactions on VLSI Systems, vol. 13, n. 6, Junio 2005.

[Stanisavljevic11] "Reliability of nanoscale circuits and systems-Methodologies and circuit architectures"; M. Stanisavljevic, M. Schmid, Y. Leblebici; primera edición, Springer, 2011.

[Stathis01] "Physical and Predictive Models of Ultra Thin Oxide Reliability in CMOS Devices and Circuits"; J.H. Stathis; Procs. 39th International Reliability Physics Symposium (IRPS '01), pp. 132-150, Orlando (Florida, EE.UU.), Abril-Mayo 2001.

[Svenningsson10] "MODIFI: A MODel-Implemented Fault Injection Tool"; R. Svenningsson, J. Vinter, H. Eriksson, M. Törngren; 29th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2010), pp. 210–222, Viena (Austria), Septiembre 2010.

[Sylvester99] "Rethinking Deep-Submicron Circuit Design"; D. Sylvester, K. Keutzer; IEEE Computer, 32(11):25-33, Noviembre 1999.

[Walker85] "A Model of Design Representation and Synthesis"; R.A. Walker, D.E. Thomas; Procs. 22nd ACM/IEEE Design Automation Conference (DAC '85), pp. 453-459, Las Vegas (Nevada, EE.UU.), Junio 1985.

[Wei11] "Comparing the effects of intermittent and transient hardware faults on programs"; J. Wei, L. Rashid, K. Pattabiraman, S. Gopalakrishnan; Dependable Systems and Networks Workshops, pp. 53-58, Hong Kong (China), Junio 2011.

[Wells08] "Adapting to intermittent faults in multicore systems"; P.M. Wells, K. Chakraborty, G.S. Sohi; Procs. 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2008), Seattle, WA, USA, Marzo 1-5, 2008.

[Winter13] "simFI: From single to simultaneous software fault injections"; S. Winter, M. Tretter, B. Sattler, N. Suri; 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013), pp. 1–12, Budapest (Hungría), Junio 2013.

[Yu0l] "A Perspective on the State of Research on Fault Injection Techniques", Y. Yu; Research Report, Mayo 2001.

[Yuste03] "Contribución a la validación de la Confiabilidad en los sistemas empotrados Tolerantes a fallos", Pedro Yuste. *Tesis Doctoral*. Departamento de Informática de Sistemas y Computadores. Universidad Politécnica de Valencia. Dirigida por Dr. Pedro Joaquín Gil Vicente y Dr. Juan José Serrano Martín. Octubre de 2003.

[Zhan14] "Optimizing the NoC Slack Through Voltage and Frequency Scaling in Hard Real-Time Embedded Systems", J. Zhan, N. Stoimenov, J. Ouyang, L. Thiele, V. Narayanan, Y. Xie, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 11, pp. 1632–1643, Noviembre 2014