UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

MEMORIA DE LOS PERÍODOS DE DOCENCIA E INVESTIGACIÓN PARA OBTENER EL DIPLOMA DE ESTUDIOS AVANZADOS

Programa de doctorado: Arquitectura de los Sistemas Informáticos en Red y Sistemas Empotrados

Memoria presentada por **Luis José Saiz Adalid**

Dirigido por Joaquín Gracia Morán y Juan Carlos Baraza Calvo

Valencia, Junio de 2010

Índice

1

Introducción

1.1 Presentación

En este informe se presenta un resumen del trabajo realizado hasta el momento por el autor durante los estudios de doctorado realizados en el Departamento de Informática de Sistemas y Computadores (DISCA) de la Universidad Politécnica de Valencia (UPV), dentro del programa de doctorado de Arquitectura de los Sistemas Informáticos en Red y Sistemas Empotrados (ASIRSE). Se incluye tanto el período de docencia como el de investigación.

En primer lugar se presenta el autor y su situación profesional y formativa. A continuación se dará una relación comentada de las asignaturas cursadas en el período de docencia. Posteriormente se detalla el entorno en que se realiza el período de investigación, incluyendo el trabajo presentado. Después se hace una relación de los artículos publicados hasta el momento. Finalmente, se presentan las conclusiones y el trabajo futuro.

1.2 El autor

El autor, Luis José Saiz Adalid, es Licenciado en Informática (en la especialidad Sistemas Físicos) desde el 20 de octubre de 1995. Trabaja en la empresa privada desde 1991, y a partir de 1999 compagina ese trabajo con una plaza de Profesor Asociado a tiempo parcial en el DISCA de la UPV. A finales de 2005 abandona la empresa privada, pasando a ocupar plaza de Profesor Colaborador en el mismo departamento, hasta la actualidad. A partir de 2006 comienza a trabajar con el Grupo de investigación de Sistemas Tolerantes a Fallos (GSTF) del Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA) de la UPV.

En lo que se refiere a los estudios de doctorado, en 2002 inicia el programa de Arquitectura y Tecnología de los Sistemas Informáticos, impartido por el DISCA. Durante los cursos 2002/03 y 2003/04 realiza el período docente, aprobando los cursos correspondientes. Dado que el programa de doctorado citado anteriormente se extinguió, en 2007 convalida el período docente y se integra en el programa de Arquitectura de los Sistemas Informáticos en Red y Sistemas Empotrados, también impartido por el

DISCA. En diciembre de 2009 presenta el trabajo de investigación correspondiente a este período, dirigido por los doctores Juan Carlos Baraza Calvo y Joaquín Gracia Morán; se titula "Análisis de nuevos modelos de fallos intermitentes para nuevas tecnologías".

1.3 Objetivos

El interés del autor se centra en el diseño y la validación de sistemas tolerantes a fallos. Para ello se ha integrado en el GSTF, con un grupo de investigadores con el Dr. Pedro Gil a la cabeza. En este entorno se ha continuado con una de las líneas de investigación abiertas por el grupo, relacionada con el modelado y la inyección de fallos mediante simulación. En este contexto se pretende proponer nuevos modelos de fallos para fallos intermitentes, estudiando sus causas, mecanismos, efectos y técnicas de mitigación.

Las tecnologías submicrométricas utilizadas hoy en día para diseñar y fabricar los sistemas informáticos actuales, y los avances en las técnicas de integración, tienen como resultado una reducción del tamaño de los transistores empleados, una disminución de la tensión de alimentación utilizada y un incremento importante en la frecuencia de trabajo. Esto conlleva mayores prestaciones, pero por otro lado la confiabilidad se ve comprometida. Se prevé un aumento en la tasa de ocurrencia de fallos, y como consecuencia, los diseñadores de sistemas deben asumir que los circuitos integrados son más sensibles a distintos tipos de fallos.

Atendiendo a la persistencia temporal de los fallos que pueden afectar a los circuitos, se pueden distinguir entre **fallos permanentes** (producidos por defectos físicos irreversibles debidos a defectos de fabricación, o a procesos de desgaste y envejecimiento), **fallos transitorios** (habitualmente provocados por el entorno físico externo y que raramente se vuelven a producir en el mismo lugar), y **fallos intermitentes**. Estos últimos han sido considerados habitualmente como el inicio de un proceso de desgaste, y por tanto como el preludio de un defecto físico irreversible que acabará en un fallo permanente. Sin embargo, en la actualidad se ha observado que existen fallos intermitentes que no conducen a fallos permanentes. Origen de estos fallos pueden ser la complejidad de los procesos de fabricación, que provoca residuos o variaciones en el proceso, junto con mecanismos de desgaste. La ocurrencia de determinados mecanismos físicos, donde se combina un *hardware* defectuoso con unas condiciones de trabajo concretas, puede propiciar la aparición de fallos intermitentes que no conduzcan a un fallo permanente. Por otra parte, diversos estudios destacan la importancia que van a alcanzar los fallos intermitentes en las nuevas tecnologías submicrométricas.

Los errores inducidos por fallos transitorios e intermitentes se manifiestan de forma similar. La principal característica de los fallos intermitentes está en que éstos se activan repetidamente en el mismo lugar, y habitualmente aparecen en ráfagas. Además, cambiar el componente afectado elimina el fallo intermitente, lo que no sucede con los fallos transitorios, que no pueden ser reparados. Los fallos intermitentes también 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*).

En relación con los fallos intermitentes, hay varias cuestiones difíciles de contestar: ¿Dónde se producen? ¿Cuándo se activan? ¿Cuántas veces se activa el fallo en una ráfaga? ¿Cómo se manifiesta el fallo a niveles de abstracción superiores? ... Para responder a estas cuestiones es importante entender los mecanismos físicos que tienen lugar en las tecnologías submicrométricas. Pero esta investigación es compleja, dependiente de la tecnología, y todavía está en una fase temprana de desarrollo.

El conocimiento de las causas de los fallos intermitentes no es suficientemente profundo, ya que se han publicado pocos trabajos de observación de fallos reales, o estudios de sus causas y mecanismos físicos.

Los fallos intermitentes han sido sistemáticamente olvidados a la hora de analizar las causas y mecanismos de los fallos reales que ocurren en los sistemas digitales. No ha sido hasta hace unos pocos años cuando los investigadores han prestado atención a estos tipos de fallos, y alertado de su creciente importancia. En la mayoría de los pocos trabajos publicados se han monitorizado sistemas reales, observando los fallos y averías provocados. Tras su estudio, se han determinado las fuentes de errores y sus manifestaciones más frecuentes, y se ha descubierto que eran causados por fallos intermitentes. Sin embargo, en la literatura existente no se han encontrado trabajos en los que se haya intentado estudiar los efectos de los fallos intermitentes mediante simulación.

A la hora de utilizar modelos de fallos, una de las cuestiones fundamentales es su representatividad, es decir, la equivalencia del modelo de fallo obtenido con el fallo real, para el nivel de abstracción utilizado. La representatividad a nivel de *hardware* de los modelos de fallos utilizados para fallos permanentes y transitorios ha sido profusamente documentada, y los modelos definidos están bien establecidos por la comunidad científica. Sin embargo, para fallos intermitentes esto no ha sido así, y no han sido muchos los trabajos 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 uno permanente. Lo cierto es que con las nuevas tecnologías submicrométricas, la reducción de tamaños, el incremento de la frecuencia de funcionamiento y la necesidad de ajustar el consumo de energía, cada vez es más frecuente la aparición de fallos intermitentes que no dan lugar a fallos permanentes. Para ello, se debe tener en cuenta la influencia de los defectos de fabricación, como residuos o variaciones del proceso. El estudio de las causas, mecanismos y efectos de los fallos intermitentes es una línea de investigación abierta en la que es necesario profundizar.

En todo caso, con la investigación realizada hasta el momento, y los conocimientos actuales de las causas y mecanismos de los fallos intermitentes, ya se pueden deducir algunos modelos de fallos intermitentes. Determinar estos modelos, incluyendo su parametrización, es uno de los objetivos de la investigación del autor.

Utilizando los modelos deducidos se han llevado a cabo algunos experimentos con una herramienta de inyección desarrollada por el grupo de investigación, con el objetivo de iniciar el estudio de los efectos de los fallos intermitentes en un microcontrolador comercial. Estos experimentos se han incluido en el trabajo de investigación y en las distintas publicaciones realizadas hasta la fecha. La metodología utilizada es la inyección de fallos en modelos VHDL (del inglés *Very high speed integrated circuit Hardware Description Language*), que se caracteriza por la gran flexibilidad, observabilidad y controlabilidad de los componentes modelados, y que permitirá obtener un análisis sistemático y exhaustivo de la influencia de los distintos parámetros estudiados, sin necesidad de esperar mucho tiempo observando la aparición de fallos reales.

Es importante remarcar que, aunque el estudio se ha realizado sobre dos ejemplos concretos de microcontroladores (el 8051 y el MARK2 tolerante a fallos), la metodología utilizada se puede generalizar a sistemas más complejos.

2

Período de docencia: asignaturas cursadas

2.1 Introducción

En este capítulo se hace una relación de las asignaturas cursadas en el período de formación, primera etapa a completar para la obtención del título de doctor. Hay que tener en cuenta que las asignaturas no corresponden al actual programa de doctorado (Arquitectura de los Sistemas Informáticos en Red y Sistemas Empotrados), sino al extinto programa de Arquitectura y Tecnología de los Sistemas Informáticos.

Las asignaturas cursadas, correspondientes a este programa de doctorado, fueron:

- Arquitectura y prestaciones de la web (curso 2002/03).
- Sistemas tolerantes a fallos (curso 2002/03).
- Visión por computador (curso 2002/03).
- Arquitectura de redes de altas prestaciones (curso 2003/04).
- Redes multimedia (curso 2003/04).

A la hora de entender la elección de estas asignaturas, y la distribución temporal, incluyendo el tiempo transcurrido entre la docencia y la presentación de esta memoria para la obtención del Diploma de Estudios Avanzados, hay que tener en cuenta la situación profesional en que se encontraba el autor. Desde noviembre de 1999 compaginaba un trabajo a tiempo completo en la empresa privada con la docencia en la UPV como profesor asociado a tiempo parcial. En el curso 2002/2003 el autor se planteó iniciar los estudios de doctorado para completar su formación y poder optar a otro tipo de plazas docentes en la Universidad. Pero la disponibilidad de tiempo era escasa, y en ese momento no estaba integrado en ninguna estructura de investigación. Por ello optó por realizar pocas asignaturas en cada curso, y además estas asignaturas no estaban inicialmente enfocadas a una línea de investigación concreta, sino que respondían al interés general del autor y a la idea de intentar abarcar la mayor cantidad posible de campos, desde Internet a la arquitectura de computadores, pasando por la visión por ordenador y la tolerancia a fallos.

2.2 Arquitectura y prestaciones de la web

La revolución que ha supuesto la generalización del uso de Internet como red global y medio de comunicación de contenidos es uno de los mayores avances de la sociedad moderna. Este avance ha venido impulsado por la mejora de las comunicaciones y los progresos en los sistemas informáticos. Estos progresos no se han dado sólo a nivel tecnológico, sino también a nivel de arquitectura.

El objetivo fundamental de la asignatura es identificar los componentes de la arquitectura de la web y estudiar las técnicas más utilizadas para medir y mejorar sus prestaciones, consiguiendo así un comportamiento cercano al óptimo. Se presenta una visión global de la arquitectura cliente/servidor y se detallan los componentes y técnicas que mayor incidencia tienen en el comportamiento de los sistemas Web: protocolos, servidores *proxies*, técnicas de *caching*, prebúsqueda, etc. La asignatura aborda los problemas de medidas de prestaciones, modelos del sistema, carga, planificación y sintonización.

Los contenidos abordados en la asignatura son:

1. Arquitectura de la Web. Modelo Cliente/Servidor.
2. Protocolos Utilizados
3. Técnicas de *Caching*
4. Réplicas de contenidos
5. Prebúsqueda
6. Caracterización de la carga
7. Medidas y Evaluación de prestaciones

El trabajo realizado en esta asignatura fue un estudio teórico acerca de varios artículos recientes que describían distintas técnicas de *caching* y prebúsqueda en web.

La elección de esta asignatura para mi formación vino motivada por la importancia de la web en la actual sociedad de la información. Conocer más a fondo su arquitectura y las distintas técnicas de mejora de prestaciones es un valor añadido, independientemente del campo concreto en el que se enfoque la investigación.

2.3 Sistemas tolerantes a fallos

En el curso 93/94 cursé la asignatura "Fiabilidad y tolerancia a fallos" como parte de los estudios conducentes al título de Licenciado en Informática. Ya entonces me interesó este tema, que con el tiempo va cobrando cada vez más actualidad. A la hora de plantearme mi situación como estudiante de Doctorado e investigador, ésta fue una de las opciones que más me atrajo.

Como continuación y ampliación de aquella asignatura aparece ésta en el plan de estudios de Doctorado. Dado mi interés en la materia, la elegí en mi primer año del período docente. La confiabilidad, propiedad de un sistema informático que permite depositar una confianza justificada en el servicio que proporciona, es una cuestión de permanente actualidad, y que requiere de investigación continua para responder a los retos que las nuevas tecnologías y arquitecturas proponen.

Los objetivos de la asignatura son:

- Analizar cuáles son los principales impedimentos a la confiabilidad de un sistema informático.
- Evaluar la confiabilidad de los sistemas analizados.
- Enumerar métodos de tolerancia a fallos basados en la redundancia de la información, el *software* y el *hardware* de un sistema informático.
- Aplicar prácticamente los conocimientos adquiridos.
- Manipular documentación en inglés.
- Sintetizar resultados y comunicarlos en público.
- Extrapolar lo aprendido y analizar su aplicación a cualquier tipo de sistema de información en general.

Tras cursar la asignatura, los contactos mantenidos con el personal del GSTF encargados de impartir la asignatura reafirmaron mi interés, que terminó plasmándose en mi integración en este grupo de investigación.

2.4 Visión por computador

Otro de los ámbitos de la Informática que está de actualidad y que tiene una gran importancia es el de la visión por computador. A la hora de elegir las asignaturas a cursar, éste me pareció un campo interesante, a la vez que me permitía ampliar conocimientos en esta área.

El objetivo global de la asignatura es dar a conocer a los alumnos la problemática general de la visión por computador como herramienta multidisciplinar para la solución de problemas de automatización de procesos de inspección en entornos industriales y de visión para robótica. Para ello se introducen, en primera instancia, los elementos básicos de todo sistema de visión: cámaras, objetivos, digitalizadores e iluminación. A continuación se desarrolla la metodología, subdividiendo el problema en un conjunto de etapas típicas en todo problema de visión, como son: procesamiento de imagen, segmentación, extracción de características y reconocimiento de formas. En cada etapa se describen un amplio conjunto de técnicas y algoritmos que dan solución a los subproblemas planteados en cada caso. Finalmente se pone todo junto, mostrando ejemplos de aplicaciones reales en diferentes contextos industriales.

Por su importancia para la visión para robótica, también se introducen los conceptos y técnicas básicas de la visión tridimensional o visión 3D.

Los conocimientos adquiridos permitirán a los alumnos abordar aplicaciones sencillas para la automatización de procesos industriales, como medición sin contacto, inspección de defectos, control de calidad y sencillas aplicaciones de visión para robots.

Como trabajo de la asignatura realicé una implementación de una red neuronal de Kohonen, constituyendo un mapa de características autoorganizado. La asignatura me ayudó a entender los distintos conceptos, tanto *hardware* como *software*, implicados en la visión por computador.

2.5 Arquitectura de redes de altas prestaciones

La mejora continua de prestaciones es fundamental en la computación actual. Independientemente de las mejoras tecnológicas que permitan aumentar la frecuencia y reducir el tamaño de los dispositivos, es fundamental mejorar la arquitectura y buscar nuevos diseños más eficientes. Una muestra evidente es el abandono de las soluciones mononúcleo en los procesadores de los PC, apostando en la actualidad por soluciones con varios núcleos.

Para el óptimo aprovechamiento de los distintos núcleos es necesario que trabajen de forma conjunta, y ello va a requerir el intercambio de información. Por tanto será necesaria una red de interconexión capaz de comunicar de forma eficiente los distintos elementos de un sistema informático. Mi intención al cursar esta asignatura fue conocer el estado del arte de la arquitectura de computadores, principalmente en lo que respecta a las redes de interconexión.

El objetivo global de la asignatura es conocer y comprender los aspectos teóricos y prácticos que intervienen en el diseño y construcción de las redes de interconexión de altas prestaciones, poniendo énfasis en la relación existente entre las distintas variables de diseño y en cómo éstas influyen en el nivel de prestaciones alcanzado por la red.

En particular, el alumno deberá ser capaz de comprender:

- Cómo una adecuada selección de la topología determina el nivel máximo de prestaciones alcanzables por la red.
- Cómo a través tanto de las técnicas de conmutación como de encaminamiento se puede hacer un uso eficiente del ancho de banda disponible y llegar a maximizar las prestaciones.
- Qué técnicas aplicar para evitar un brusco deterioro de las prestaciones, y garantizar un trabajo sostenido de la red cuando ésta se aproxima a la zona de saturación o se detecta la ocurrencia de fallos en sus componentes.

Al finalizar, el alumno habrá debido adquirir una sólida base que le permita entender cómo algunas de estas técnicas se aplican en las tecnologías de red actuales, y ser capaz de identificar aquellos aspectos susceptibles de mejora.

Contenidos:

1. Introducción a las redes de interconexión
2. Topologías de interconexión
3. Técnicas de conmutación y control de flujo
4. Técnicas de encaminamiento eficientes
5. Encaminamiento en redes con topología irregular
6. Técnicas de equilibrado de tráfico
7. Técnicas básicas de control de la congestión
8. Técnicas de encaminamiento tolerantes a fallos
9. Estructura básica del *router* e interfaz de red

El trabajo final de esta asignatura fue un estudio teórico de artículos recientes acerca de redes de interconexión tolerantes a fallos, buscando aproximarme a mi área de interés principal, la tolerancia a fallos.

2.6 Redes multimedia

Otro de los ámbitos de gran actualidad y permanente evolución es el de las tecnologías de red y las comunicaciones, especialmente en lo que se refiere a transmisión de datos multimedia, redes de sensores, de dispositivos móviles y redes *ad-hoc*. Tener conocimientos sobre estas tecnologías es un valor añadido para un doctor en Informática.

El objetivo de esta asignatura es proporcionar al alumno una visión completa de las nuevas tecnologías de red. Se presentan las nuevas tecnologías de redes de acceso y LAN (del inglés *local area networks*, redes de área local), y se da una visión general de la seguridad. Se introducen las distintas estrategias para el transporte eficiente de los datos multimedia (técnicas de difusión de audio/vídeo, mecanismos para la protección y ocultación de errores, calidad de servicio, etc.), y los sistemas de compresión de datos utilizados (principalmente imagen y vídeo).

El trabajo realizado en esta asignatura fue un estudio acerca de algoritmos de encaminamiento en redes *ad-hoc* móviles cuyo objetivo sea reducir el consumo de energía. Efectivamente, uno de los problemas de los dispositivos móviles es su limitada autonomía, ya que generalmente están alimentados por baterías. La utilización de algoritmos de encaminamiento capaces de ahorrar energía permitirá aumentar la autonomía de los dispositivos móviles y la vida de las baterías.

2.7 Resumen y conclusiones

En este capítulo se ha hecho una relación de las asignaturas cursadas en el período de docencia, primera etapa a completar para la obtención del título de doctor. Las asignaturas cursadas, correspondientes al extinto programa de doctorado de Arquitectura y Tecnología de los Sistemas Informáticos, fueron:

- Arquitectura y prestaciones de la web (curso 2002/03).
- Sistemas tolerantes a fallos (curso 2002/03).
- Visión por computador (curso 2002/03).
- Arquitectura de redes de altas prestaciones (curso 2003/04).
- Redes multimedia (curso 2003/04).

Con estas asignaturas el autor ha pretendido abarcar la mayor cantidad de ámbitos relacionados con el *hardware* de los sistemas informáticos, desde Internet a la arquitectura de computadores, pasando por la visión por ordenador y la tolerancia a fallos.

Entre las temáticas menos tratadas por el autor se encuentran la robótica o los sistemas operativos. No obstante, los conocimientos obtenidos en estas asignaturas, centrados en la confiabilidad y tolerancia a fallos, la visión por computador, las tecnologías de red, las técnicas de mejora para la web y la arquitectura de redes de interconexión, son temas de actualidad y en continua evolución.

3

Período de investigación: entorno y trabajo de doctorado

3.1 Introducción

En 2006, el autor se integra en el Grupo de investigación de Sistemas Tolerantes a Fallos (GSTF) del Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA) de la UPV. En concreto, está trabajando con un grupo de investigadores, con el Dr. Pedro Gil a la cabeza, cuyo interés se centra en el diseño y validación de sistemas tolerantes a fallos.

Esta integración coincide con la incorporación del autor como profesor colaborador en el Departamento de Informática de Sistemas y Computadores. La carga docente (33 créditos) deja inicialmente poco tiempo para la investigación, por lo que los primeros resultados tardan en llegar. Poco a poco se va definiendo la línea de investigación, enfocada en la obtención de modelos de fallos intermitentes y el estudio de sus efectos mediante la inyección de fallos sobre modelos en VHDL.

Como resultado de todo este período, se presenta el trabajo de investigación tutelado, bajo el título "Análisis de nuevos modelos de fallos intermitentes para nuevas tecnologías" y codirigido por los doctores Joaquín Gracia Morán y Juan Carlos Baraza Calvo. En las secciones siguientes de este capítulo se resume este trabajo. El documento completo se presenta como anexo II.

Además del trabajo, se han conseguido diversas publicaciones, algunas de ellas en actas de congresos del máximo prestigio a nivel mundial. El listado de las publicaciones se presenta en el capítulo 4.

3.2 Resumen del trabajo de investigación

3.2.1 Introducción y objetivos

Las tecnologías submicrométricas utilizadas para diseñar y fabricar los sistemas informáticos actuales, y los avances en las técnicas de integración, tienen como resultado una reducción del tamaño de los transistores empleados, una disminución de la tensión de alimentación y un incremento importante en la frecuencia de trabajo. Esto conlleva mejores prestaciones, pero se hace a costa de comprometer la confiabilidad de los circuitos integrados. Se prevé un aumento en la tasa de ocurrencia de fallos, y como consecuencia, los diseñadores de sistemas deben asumir que los circuitos integrados son más sensibles a distintos tipos de fallos.

Los fallos intermitentes han sido considerados habitualmente como el inicio de un proceso de desgaste, y por tanto como el preludio de un defecto físico irreversible que acaba en un fallo permanente. Sin embargo, en la actualidad se ha observado que existen fallos intermitentes que no conducen a fallos permanentes. Origen de estos fallos pueden ser la complejidad de los procesos de fabricación, que provoca residuos o variaciones en el proceso, junto con mecanismos de desgaste. La ocurrencia de determinados mecanismos físicos, donde se combina un *hardware* defectuoso con unas condiciones de trabajo concretas, puede propiciar la aparición de fallos intermitentes que no conduzcan a un fallo permanente. Por otra parte, diversos estudios destacan la importancia que van a alcanzar los fallos intermitentes en las nuevas tecnologías submicrométricas.

Los errores inducidos por los fallos transitorios y los intermitentes se manifiestan de forma similar. La principal característica de los fallos intermitentes está en que éstos se activan repetidamente en el mismo lugar, y habitualmente aparecen en ráfagas. Además, cambiar el componente afectado elimina el fallo intermitente, lo que no sucede con los transitorios, que no pueden ser reparados. Los fallos intermitentes también se pueden activar o desactivar por cambios de temperatura, tensión de alimentación o frecuencia de trabajo (cambios conocidos como variaciones PVT, del inglés *Process, Voltage and Temperature*).

En relación con los fallos intermitentes, hay varias cuestiones difíciles de contestar: ¿Dónde se producen? ¿Cuándo se activan? ¿Cuántas veces se activa el fallo en una ráfaga? ¿Cómo se manifiesta el fallo a niveles de abstracción superiores? ... Para responder a estas preguntas es importante entender los mecanismos físicos que tienen lugar en las tecnologías submicrométricas. Pero esta investigación es compleja, dependiente de la tecnología, y todavía está en una fase temprana de desarrollo.

El conocimiento de las causas de los fallos intermitentes no es suficientemente profundo, ya que se han publicado pocos trabajos donde se hayan observado fallos reales, o estudiado sus causas y mecanismos físicos.

Los fallos intermitentes han sido tradicionalmente olvidados a la hora de analizar las causas y mecanismos de los fallos reales que ocurren en los sistemas digitales. No ha sido hasta hace unos pocos años cuando los investigadores han prestado atención a este tipo de fallos, y alertado de su creciente importancia. Se han publicado algunos trabajos en los que se muestra la monitorización de sistemas reales, observando los fallos y averías provocados. Tras su estudio, se han determinado las fuentes de errores y sus manifestaciones más frecuentes, y se ha descubierto que eran causados por fallos intermitentes. Sin embargo, en la literatura existente no se han encontrado trabajos en los que se haya intentado estudiar los efectos de los fallos intermitentes mediante simulación.

A la hora de utilizar modelos de fallos, una de las cuestiones fundamentales es su representatividad, es decir, la equivalencia del modelo de fallo obtenido con el fallo real, para el nivel de abstracción utilizado. El objetivo del trabajo de investigación es determinar y utilizar modelos de fallos a nivel de puertas lógicas y de transferencia entre registros. La representatividad a nivel de *hardware* de los modelos de fallos utilizados para fallos permanentes y transitorios a este nivel de abstracción ha sido profusamente documentada, y los modelos definidos están bien establecidos por la comunidad científica. Sin embargo, para fallos intermitentes esto no ha sido así, y no han sido muchos los trabajos 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 uno permanente. Lo cierto es que con las nuevas tecnologías submicrométricas, la reducción de tamaños, el incremento de la frecuencia de funcionamiento y la necesidad de ajustar el consumo de energía, cada vez es más frecuente la aparición de fallos intermitentes que no dan lugar a fallos permanentes. Para ello, se debe tener en cuenta la influencia de los defectos de fabricación, como residuos o variaciones del proceso. El estudio de las causas, mecanismos y efectos de los fallos intermitentes es una línea de investigación abierta en la que es necesario profundizar.

En todo caso, con la investigación realizada hasta el momento, y los conocimientos actuales de las causas y mecanismos de los fallos intermitentes, ya se pueden deducir algunos modelos de fallos intermitentes.

3.2.2 Conceptos básicos sobre tolerancia a fallos

La confiabilidad 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 reducir la existencia de fallos en el sistema mediante las etapas de verificación, diagnóstico y corrección. 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 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 de propagación, detección y recuperación de errores; y los coeficientes de cobertura de detección y recuperación de errores, de una manera más sencilla que con los métodos analíticos. Se puede realizar mediante inyección de fallos, esto es, introduciendo deliberadamente fallos en un modelo experimental o prototipo del sistema.

3.2.3 Técnicas de inyección de fallos

Las técnicas de inyección de fallos se pueden aplicar tanto sobre modelos como sobre prototipos. Sobre modelos se pueden inyectar fallos mediante simulación (sobre un modelo virtual en un computador) o mediante emulación (sobre un circuito real en el que se ha sintetizado el sistema). Sobre prototipos se puede hacer utilizando herramientas *hardware* o programas.

En el capítulo 3 del trabajo se describen las técnicas más representativas, mostrando las variantes que han aparecido junto con los artículos más relevantes, y se han analizado las ventajas e inconvenientes de cada una de ellas. En cada una de las distintas técnicas revisadas se ha hecho hincapié en los modelos de fallos que se pueden inyectar.

En particular, se ha hecho especial énfasis en las técnicas de inyección de fallos mediante simulación de modelos en VHDL, ya que son las que se han utilizado para la realización de los experimentos y obtención de resultados.

3.2.4 Modelos de fallos

En el capítulo 4 del trabajo se presenta un resumen de los principales mecanismos de fallo en las tecnologías submicrométricas actuales, y se deduce un conjunto de modelos de fallos más amplio que los utilizados tradicionalmente en los experimentos de inyección de fallos (*stuck-at* para los fallos permanentes y *bit-flip* para los transitorios). Estos modelos se han deducido para los niveles lógico y de transferencia de registro (RTL), y para fallos permanentes, intermitentes y transitorios.

La reducción de las geometrías ha potenciado la aparición de nuevos mecanismos de fallos permanentes e intermitentes. Estos mecanismos afectan principalmente a la capa de óxido de los transistores y a las conexiones y el encapsulado de los circuitos integrados.

En lo que se refiere a los fallos permanentes, aparecen defectos irreversibles en los circuitos, producidos durante el proceso de fabricación o por desgaste, y se manifiestan básicamente a nivel electrónico como cortocircuitos y circuitos abiertos, que a nivel lógico se pueden modelar como fallos de tipo *stuck-at*, *open-line*, *stuck-open*, *indetermination*, *delay*, *short* y *bridging*.

En cuanto a los fallos transitorios, la reducción de las geometrías y de las tensiones de alimentación, junto con el aumento de las frecuencias de funcionamiento, han causado una notable reducción del efecto de los mecanismos naturales de enmascaramiento de los circuitos digitales, y la aparición de efectos capacitivos que afectan a la respuesta temporal de los circuitos. También son fuente de errores las radiaciones, tanto internas como externas. Los modelos de fallo establecidos son *bit-flip* (en elementos de almacenamiento), *pulse* (en lógica combinacional), *indetermination* y *delay*.

Se espera también un aumento en la tasa de fallos intermitentes, causado por las nuevas tecnologías submicrométricas, en las que aparecen nuevos mecanismos físicos que provocan fallos intermitentes que no terminan en fallo permanente. Por ello, es necesario profundizar en sus causas y mecanismos.

La complejidad de los procesos de fabricación puede provocar residuos o variaciones en el proceso, que producen contactos intermitentes en las celdas de memoria. Las soldaduras defectuosas pueden producir cortocircuitos o circuitos abiertos intermitentes en líneas de interconexión. La deslaminación de la capa de barrera y la electromigración pueden incrementar la resistencia del material, produciendo retardos,

cortocircuitos o circuitos abiertos de forma intermitente. La diafonía aparece cuando hay un acople capacitivo entre dos líneas adyacentes, y puede producir picos (en una línea que debería permanecer a valor constante), retardos (conmutación más lenta de lo esperado) o aceleraciones (conmutación más rápida de lo esperado). Daños en la capa de óxido, como el mecanismo denominado *soft breakdown*, pueden producir que la corriente de fuga del óxido de la puerta fluctúe, de forma que se obtienen valores erráticos del voltaje de salida que pueden llevar a valores lógicos indeterminados.

Los fallos intermitentes se ven potenciados, y se pueden activar o desactivar, por cambios de temperatura, tensión de alimentación o frecuencia de trabajo (variaciones PVT). Los modelos propuestos para estos fallos son *intermittent stuck-at*, *intermittent pulse*, *intermittent short*, *intermittent open*, *intermittent delay*, *intermittent speed-up* e *intermittent indetermination*.

Para finalizar, es preciso comentar que no se ha intentado hacer un estudio exhaustivo de los distintos mecanismos de fallos, nuevas tecnologías, etc. Simplemente, se ha querido hacer un resumen de modelos de fallos deducidos para los distintos tipos de fallo, haciendo énfasis en los nuevos modelos propuestos para fallos intermitentes.

3.2.5 Estudio de los efectos de los fallos intermitentes

Los modelos de fallos intermitentes deducidos en el capítulo 4 del trabajo se han aplicado sobre el modelo en VHDL del microcontrolador 8051 ejecutando el algoritmo *bubblesort*, y se han inyectado fallos en los elementos de almacenamiento y en los buses del sistema.

El objetivo de los experimentos ha sido estudiar la influencia de distintos parámetros que afectan a los fallos intermitentes. Inicialmente se han estudiado los parámetros que afectan al comportamiento de una ráfaga: los tiempos de actividad e inactividad y el número de activaciones. Posteriormente se han estudiado otros factores como la multiplicidad espacial, el lugar de la inyección y la frecuencia de trabajo del sistema. Finalmente, se han comparado los efectos de los fallos intermitentes con los de los fallos transitorios y permanentes.

Como conclusión final, se puede afirmar 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. Por el contrario, el tiempo de inactividad parece no tener una gran influencia en los resultados.

3.3 Trabajo futuro

De cara al futuro, quedan varias líneas de investigación abiertas. La primera es seguir estudiando las causas y mecanismos físicos que provocan los fallos intermitentes, para profundizar en su comprensión. Dado que el autor no es experto en física ni en electrónica de bajo nivel, esta parte del trabajo será de estudio bibliográfico.

Con el estudio de las causas físicas hay que proponer nuevos modelos de fallo, o bien determinar cómo aplicar los ya deducidos en otros puntos del sistema, especialmente en la lógica combinacional. La evolución de la importancia de los fallos intermitentes hace pensar que éstos también pueden afectar a los circuitos combinacionales.

Como se ha podido observar en los experimentos realizados, los resultados pueden ser muy dependientes de la carga de trabajo a la que se somete al sistema. Por tanto, es fundamental aplicar distintas

cargas de trabajo en los experimentos, con el fin de poder generalizar las conclusiones. Para ello se van a utilizar nuevas cargas.

Se ha comentado la posible dependencia de los resultados del sistema concreto bajo estudio. Por tanto, para generalizar los resultados y las conclusiones obtenidas hay que realizar inyecciones en distintos sistemas. También hay que planificar experimentos en sistemas tolerantes a fallos, para estudiar si sus mecanismos de tolerancia son válidos. El mayor problema en esta cuestión es la poca disponibilidad de modelos en VHDL de sistemas tolerantes a fallos.

Una vez estudiados los problemas y sus causas, hay que buscar las soluciones. En primer lugar se estudiarán las técnicas de mitigación actuales, para ver cuáles podrían resultar útiles para tolerar los errores provocados por los fallos intermitentes. Finalmente, el objetivo es proponer nuevas técnicas de mitigación para una rápida detección y recuperación, y aplicarlas en el modelo en VHDL de un sistema para comprobar la validez del esquema propuesto.

4

Publicaciones realizadas

4.1 Introducción

Desde la incorporación al GSTF, el autor ha publicado diversos trabajos en las actas de distintos congresos, entre los que se cuentan los más importantes a nivel internacional en el campo de la confiabilidad y la tolerancia a fallos. A continuación se enumeran las publicaciones, cuyo texto completo se incluye en el anexo I.

4.2 Fast and Early Validation of VLSI Systems

Este primer artículo se presentó, con el autor como único firmante, al *student forum* del *6th European Dependable Computing Conference* (EDCC-6), celebrado en Coimbra (Portugal), en octubre de 2006, y publicado en el *Supplemental Volume* de dicho congreso. En este artículo se iniciaba una línea de investigación en la que posteriormente no se ha profundizado: preparar un entorno de validación basado en inyección de fallos sobre modelos, combinando la simulación por *software* y la emulación sobre FPGA. El objetivo era combinar las ventajas de ambos entornos, principalmente la velocidad de la emulación y la controlabilidad y observabilidad de la simulación.

Esta línea de investigación no se ha profundizado debido principalmente a dos motivos: el aumento de prestaciones de los computadores ha reducido enormemente los tiempos necesarios para las simulaciones; además, es difícil disponer de modelos de sistemas lo suficientemente complejos como para que valga la pena disponer de un entorno de validación tan complejo como el descrito. No obstante, la línea de investigación queda abierta por su interés, como demuestra el hecho de que el artículo haya sido referenciado en "Contribución a la validación experimental no intrusiva de la confiabilidad en los sistemas empotrados basados en componentes COTS", tesis doctoral presentada por Juan Pardo Albiach en 2007.

En cuanto al congreso, comentar que el EDCC es el más importante a nivel europeo en el campo de la confiabilidad, y está recomendado por el IFIP (*International Federation for Information Processing*) *Working Group 10.4 on Dependable Computing and Fault Tolerance*. Su nivel se refleja en su inclusión

en la clasificación de congresos elaborada por ERA (*Excelence in Research for Australia*), que sustituye a la clasificación CORE (*Computing Research and Education Association of Australasia*). En la versión de diciembre de 2009 aparece con categoría B.

4.3 Analysis of the influence of intermittent faults in a microcontroller

Esta comunicación se presentó al *11th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems* (DDECS'08), celebrado en Bratislava (Eslovaquia) en abril de 2008. Los firmantes de este artículo fueron Joaquín Gracia Morán, Luis José Saiz Adalid, Juan Carlos Baraza Calvo, Daniel Gil Tomás y Pedro Gil Vicente. Con este artículo se iniciaba una línea de investigación en el estudio de las causas y mecanismos de los fallos intermitentes, para proponer modelos de fallos y utilizar esos modelos para inyectar fallos en el modelo en VHDL de un microcontrolador.

Este artículo ha sido referenciado por Layali Rashid, Karthik Pattabiraman y Sathish Gopalakrishnan en su trabajo "*Towards Understanding the Effects of Intermittent Hardware Faults on Programs*", presentado al *4th Workshop on Dependable and Secure Nanocomputing* (WDSN) cuyas actas se publican en el *Proceedings Supplemental Volume of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2010).

4.4 Applying Fault Injection to Study the Effects of Intermittent Faults

Este *fast abstract* se presentó en el *7th European Dependable Computing Conference* (EDCC-7), que tuvo lugar en Kaunas (Lituania) en mayo de 2008. Los autores, por orden de firma, fueron Luis José Saiz Adalid, Joaquín Gracia Morán, Juan Carlos Baraza Calvo, Daniel Gil Tomás y Pedro Gil Vicente. Este artículo supone la continuación de la línea de investigación abierta con el artículo anterior.

4.5 Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers

Un artículo que se presentó en el *IEEE International High Level Design Validation and Test Workshop 2008* (HLDVT'08), que se celebró en Incline Village (Nevada-EE.UU.) en noviembre de 2008. Los firmantes fueron Daniel Gil Tomás, Luis José Saiz Adalid, Joaquín Gracia Morán, Juan Carlos Baraza Calvo y Pedro Gil Vicente. Supuso un paso más en la línea de investigación en curso. Además, este artículo lo presenté personalmente en este congreso, y fue mi primera defensa de un artículo completo ante la audiencia de un congreso.

El congreso HLDVT está catalogado con categoría C en la clasificación de congresos elaborada por ERA (versión de diciembre de 2009).

4.6 Intermittent Faults: Analysis of Causes and Effects, New Fault Models, and Mitigation Techniques

Con todo lo presentado en los tres artículos previos, e incluyendo una propuesta para continuar la línea de investigación, en lo que puede llegar a ser la tesis doctoral del autor, se escribió este trabajo para el *student forum* del *39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2009), celebrado en Estoril (Portugal) en junio de 2009, con el autor como único firmante. Además, la organización del congreso dio la oportunidad de presentar el trabajo como póster, además de como presentación oral. Esto permitió un intercambio de impresiones más directo con quienes observaban los pósters.

Este artículo, junto con los tres anteriores, es la producción relacionada con el trabajo de investigación descrito en el capítulo precedente. Queda fuera de ese trabajo el primer artículo, por pertenecer a una línea de investigación distinta, y los que se describen a continuación, por ser posteriores en el tiempo, aunque suponen el desarrollo de propuestas ya planteadas en el trabajo.

En cuanto al congreso, hay que destacar que el DSN es el más importante a nivel mundial en cuanto a confiabilidad se refiere. Su nivel se refleja en su inclusión en la clasificación de congresos elaborada por ERA (versión de diciembre de 2009), con categoría A.

4.7 A Proposal of a Fault-Tolerant Mechanism for Microprocessor Buses

Este *fast abstract* se ha presentado recientemente en el *8th European Dependable Computing Conference* (EDCC-8), celebrado en Valencia (España) y organizado por nuestro grupo de investigación en abril de 2010. Los firmantes han sido Luis José Saiz Adalid, Juan Carlos Baraza Calvo, Joaquín Gracia Morán y Daniel Gil Tomás. Supone un primer acercamiento hacia las técnicas de tolerancia a fallos intermitentes aplicadas a los buses del sistema. En concreto, se empieza a investigar en el uso de códigos correctores de errores. En este caso también hice yo la defensa del artículo.

La oportunidad de vivir la organización de un evento de este tipo desde dentro (aunque sin cargo de responsabilidad) resultó una experiencia interesante.

4.8 Experimental Validation of a Fault Tolerant Microcomputer System against Intermittent Faults

Este artículo ha sido aceptado para su presentación en el *40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN 2010), que se va a celebrar en Chicago (Illinois-EE.UU.) entre el 28 de junio y el 1 de julio de 2010. Los autores, por orden de firma, son Joaquín Gracia

Morán, Daniel Gil Tomás, Luis José Saiz Adalid, Juan Carlos Baraza Calvo y Pedro Gil Vicente. En este artículo se presenta la validación de un sistema tolerante a fallos mediante simulación, para ver la respuesta del sistema al inyectarle fallos intermitentes.

Este artículo continúa la investigación relativa al modelado de fallos intermitentes y su aplicación en la inyección de fallos sobre modelos en VHDL, para estudiar sus efectos sobre distintos sistemas microprocesadores.

4.9 A Proposal to Tolerate Intermittent Faults in Microprocessor Buses

Aprovechando que el anterior artículo había sido aceptado en el DSN 2010, y dado que había trabajo en curso que, si bien no da para un artículo completo, valía la pena presentar, se decidió escribir este *fast abstract* para su envío al mismo congreso. Nuevamente, los firmantes fuimos Luis José Saiz Adalid, Juan Carlos Baraza Calvo, Joaquín Gracia Morán, Daniel Gil Tomás y Pedro Gil Vicente.

Este artículo es la continuación del trabajo presentado en el EDCC-8, en el que se proponía el uso de códigos correctores de errores para tolerar fallos intermitentes en los buses del sistema.

4.10 Resumen

Desde la incorporación del autor al GSTF su producción está formada por ocho artículos (tres artículos completos, tres *fast abstracts* y dos contribuciones a *student forums*). Entre los congresos a los que se han presentado las ponencias, destacar las distintas ediciones del DSN y del EDCC.

El texto completo de todos los artículos se incluye en el anexo I.

5

Conclusiones

5.1 Introducción

Esta memoria presenta el trabajo realizado y los conocimientos adquiridos por el autor durante el período inicial de formación para obtener el título de doctor. Desde los cursos de formación hasta el trabajo de investigación, las publicaciones realizadas y las líneas de investigación en marcha, todo se ha ido detallando a lo largo de esta memoria. En el anexo I se incluyen las publicaciones aceptadas hasta ahora, y en el anexo II se presenta el trabajo de investigación completo. En la siguiente sección se esboza el trabajo en curso y futuro, con el objetivo de la tesis doctoral en el horizonte.

5.2 Trabajo en curso y futuro

Las líneas de investigación abiertas se han apuntado en la sección 3.3. En concreto, en la actualidad tenemos en proyecto completar el estudio de los síndromes de los errores provocados por fallos intermitentes, y continuar con la validación de un sistema tolerante a fallos, al inyectarle fallos intermitentes. Por otra parte, hay que continuar con el estudio de técnicas de mitigación de fallos en buses, utilizando códigos correctores de errores. También hay que investigar técnicas de mitigación para otros elementos del sistema que sean especialmente sensibles a los fallos intermitentes, como pueden ser los elementos de almacenamiento (memoria y registros) o la lógica combinacional.

Finalmente, la conclusión es que se está llevando a cabo un estudio completo de los efectos de los fallos intermitentes en los sistemas informáticos. Para ello se investigan sus causas y mecanismos, y se proponen modelos de fallos. El objetivo es plantear técnicas de mitigación adaptadas a las características específicas de este tipo de fallo, cada vez más frecuente en las tecnologías actuales.

Anexo I: Publicaciones

Este anexo incluye las publicaciones realizadas hasta la fecha:

[Saiz06] "Fast and Early Validation of VLSI Systems"; Luis J. Saiz; Procs. Supplemental Volume of the 6th European Dependable Computing Conference (EDCC-6), Coimbra (Portugal), Octubre 2006.

[Gracia08] "Analysis of the influence of intermittent faults in a microcontroller"; J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil; Procs. 11 th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'08), Bratislava (Eslovaquia), Abril 2008.

[Saiz08] "Applying Fault Injection to Study the Effects of Intermittent Faults"; L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil; Procs. 7th European Dependable Computing Conference (EDCC-7), Kaunas (Lituania), Mayo 2008.

[DGil08] "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", D. Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil; Procs. IEEE International High Level Design Validation and Test Workshop 2008 (HLDVT'08), Incline Village (Nevada-EE.UU.), Noviembre 2008.

[Saiz09] "Intermittent Faults: Analysis of Causes and Effects, New Fault Models, and Mitigation Techniques"; Luis-J. Saiz-Adalid; Procs. Supplemental Volume of the 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"; Luis-J. Saiz-Adalid, J.-Carlos Baraza-Calvo, Joaquín Gracia-Morán, Daniel Gil-Tomas; Procs. Supplemental Volume of the 8th European Dependable Computing Conference (EDCC-8), Valencia (España), Abril 2010.

[Gracia10] "Experimental Validation of a Fault Tolerant Microcomputer System against Intermittent Faults"; Joaquín Gracia-Morán, Daniel Gil-Tomás, Luis-J. Saiz-Adalid, J.-Carlos Baraza-Calvo, Pedro-J. Gil-Vicente; pendiente de publicación en actas del 40th Conference on Dependable Systems and Networks (DSN 2010), Chicago (Illinois-EE.UU.), Junio 2010.

[Saiz10b] "A Proposal to Tolerate Intermittent Faults in Microprocessor Buses"; Luis-J. Saiz-Adalid, J.-Carlos Baraza-Calvo, Joaquín Gracia-Morán, Daniel Gil-Tomás, Pedro-J. Gil-Vicente; pendiente de publicación en actas (Supplemental Volume) del 40th Conference on Dependable Systems and Networks (DSN 2010), Chicago (Illinois-EE.UU.), Junio 2010.

NOTA: las referencias en negrita corresponden a los artículos incluidos en el trabajo de investigación. Su texto está duplicado, ya que se incluyen tanto en este anexo como al final del trabajo.

Fast and Early Validation of VLSI Systems*

Luis J. Saiz
Fault-Tolerant Systems Research Group (GSTF), Technical University of Valencia (UPV)
E.T.S.I.Ap. Building 1G. Camino de Vera, s/n. 46022 Valencia, Spain
ljsaiz@disca.upv.es

Abstract

Fault Injection is a well known technique for systems Fault Tolerance validation. Simulation is a valuable experimental approach to get an early characterization of systems performance and dependability. Validation using Fault Simulation enables system defects to be fixed in early phases of the design cycle, saving thus time-to-market and money.

Nevertheless, the main problem of Fault Simulation is its high temporal cost. During last years, Fault Emulation using FPGA has emerged as a fast fault injection technique that can also be applied in early design cycle stages.

The combination of these two techniques can produce a Fault Simulation-Fault Emulation co-validation framework, where a Model-driven validation of VLSI systems can be carried out.

This paper shows the ideas or rules to accomplish this task using a co-validation methodology.

1. Introduction

When developing and exploiting modern dependable systems, one of the main problems is their validation. Functional aspects as well as the correct operation of the Fault Tolerance Mechanisms (FTMs) must be considered.

Fault Injection is one of the most used techniques during system validation. It enables FTMs validation and verification and system dependability characterization. This technique can be classified into three main categories [1]: physical fault injection (or Hardware Implemented Fault Injection, HWIFI), Software Implemented Fault Injection (SWIFI) and Simulated Fault Injection. Although Fault Emulation is commonly included into Simulated Fault Injection, it has a different basis (as explained later), and it could be a category on its own into Fault Injection techniques.

As one of the objectives of this work is to get an early way to validate VLSI systems, it is not feasible to wait until the real system is available. So we will focus on Model-driven Fault Injection techniques. Figure 1 shows their classification.

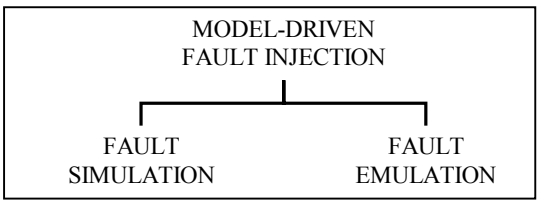


Figure 1. Model-driven Fault Injection techniques

In the design phase of a system, Fault Simulation is an important experimental approach to get an early measure of performance and dependability. In this way, when using simulated fault injection, a model of the system under test is simulated in a computer system. Faults are induced by altering the logical values of the model elements during the simulation. Another interesting advantage of this technique with respect to other injection techniques is the high observability and controllability of all modelled components.

Fault Emulation is a technique that allows us to get a hardware prototype of the system under test. With this prototype we can study the behaviour of the system in an early design stage. FPGAs provide solutions to implement hardware description language

* This work has been partially funded by the Spanish Government under the project MCYT TEC 2005-05119/MIC, "Estudio de las técnicas de inyección de fallos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida"

(HDL) models of VLSI systems, and their reconfiguration capabilities can be of interest to emulate the behaviour of the system in the presence of faults.

One of the important differencies between Fault Simulation and Fault Emulation is the way they inject faults. In Fault Simulation, the injection is performed by altering the logical values of the elements of the model (signals, variables, etc.) As it is a software simulation, pieces of code (saboteurs, mutants) or simulator commands are used to change logical values. In Fault Emulation, we have an FPGA that functionally behaves as the system under test, and we use its reconfiguration capabilities to emulate the occurrence of faults into the implemented circuit.

When designing VLSI systems, it is common the use of HDLs during the design phase. Different Model-driven Fault Injection techniques have been developed [2, 3, 4], but there is not any framework to select the most adequate Fault Injection technique according to the characteristics of the system or system component model to validate, the desired fault models to study, the places where inject faults or the observability and abstraction level required.

The objective of this work is to open a discussion about the feasibility of a co-validation methodology for VLSI systems validation that allows the system designer to select the best suitable technique for its model and test conditions.

This paper is organized as follows. Section 2 explains the characteristics of Fault Simulation. Section 3 describes Fault Emulation properties. Finally, section 4 draws the lines for future research in order to get the best of Fault Simulation and Fault Emulation techniques in a co-validation methodology.

2. Fault Simulation

When using this technique, a model of the system under test is simulated in a computer system. The injection is carried out by altering the logical values of the elements of the model during the simulation. Main approaches of this technique are based on software simulation of electrical schemas or hardware description languages models.

Nowadays it is very common to design a system by means of an HDL. VHDL [5] has taken the place of other previously popular HDL (AHDL, ABEL). System level modelling and simulation is very easy to understand and many misconceptions and type translation problems are avoided because VHDL is a very strongly typed language. Other languages as Verilog [6] are also used for validation [7].

The main advantages of Fault Simulation are:

- This technique can inject a greater set of fault models than other techniques, and the injection can also be done in more places. As an example, Table 1 summarizes the fault models that can be used when VHDL-Based Fault Injection is used [8].
- In addition, it presents a great observability and controllability of all modelled components.
- The simulation can be accomplished at different abstraction levels.
- It is possible to carry out the validation of the system during the design phase, before having the final product, saving thus money and time-to-market.

On the other hand, there exist several disadvantages with this technique:

- The temporal cost of the simulations can be excessively great, particularly when validating complex systems.
- Model accuracy influences the results.
- The overload of the simulator grows when injecting faults, depending on the injection technique used.

Table 1. VHDL-Based Fault Models [8]

	Fault models	
Injection technique	**Transient Faults**	**Permanent / Intermittent Faults**
Simulator Commands	Pulse*, Bit-flip**, Indetermination, Delay	Stuck-at (0,1), Indetermination, Open-line, Delay
Saboteurs	Pulse*, Bit-flip**, Indetermination, Delay	Stuck-at (0,1), Indetermination, Open-line, Delay, Short, Bridging, Stuck-open
Mutants	Syntactical changes	Syntactical changes
* In combinational logic		
** In storage elements (registers and memory)		

The main impairment of Fault Simulation is the high temporal cost of the simulations when the system under test is very complex [9, 10]. With these systems, a good solution is the combination of techniques [11] or the study of specific parts of the system [12]. We will base our co-validation methodology in these two ideas.

3. Fault Emulation

A Field-Programmable Gate Array (FPGA) is a programmable logic device capable of implementing complex logical functions and systems.

SRAM-based FPGAs provide solutions to implement HDL models of VLSI systems. Although FPGA are mainly used for logic prototyping and verification purposes, their reconfiguration capabilities can be also of interest to emulate the behaviour of the system in the presence of faults.

Fault Emulation is a technique that allows us to get a hardware prototype of the system under test. The main advantage of this technique is its speed: it is much faster to run a system model on an FPGA than performing Fault Simulation. The emulation can be performed on first stages of system design. It is possible to connect the system prototype to the actual operational environment ("in-system emulation"). Against HWIFI, this technique has better accessibility, a bigger set of fault models and it permits a more accurate analysis of fault pathology.

Although first approaches to this technique had very limited fault models against other simulation techniques, latest research had accomplished to emulate more fault models [13, 14], as summarized in Table 2.

Table 2. FPGA-Based Fault Models [13, 14]

Fault models	
Transient Faults	**Permanent Faults**
Bit-flip, Pulse, Delay, Indetermination	Stuck-at (0,1) , Stuck-open, Short, Open-line, Bridging, Delay, Indetermination

Disadvantages of this technique are:

- Not all HDL models can be implemented into an FPGA. As we need to use a synthesizable model, we can use only a subset of HDL capabilities.
- It is not possible to inject faults in all parts of the model, as well as it is not possible to inject so many fault models as Fault Simulation does.
- There is a limited set of commercial emulators.
- Depending on the emulator used, we may have restricted data reading (limited Input/Output ports).
- It is necessary to reconfigure the FPGA and, sometimes, to download the configuration file.
- As technology changes so fast, newer models of FPGA or boards can obsolete previous works.

The first approach to Fault Emulation was based on stopping system's work and reconfiguring the FPGA (static fault injection). The main problem was the big amount of time lost during the reconfiguration.

Nowadays, Fault Emulation can be performed using compile- and run-time reconfiguration techniques. Compile-time reconfiguration is based on the use of modified HDL models in order to make them fault injectable [15]. Run-time reconfiguration exploits the reconfiguration capabilities of FPGA [16]. In this case, internal resources of the FPGA are reconfigured on the fly to emulate the occurrence of a fault in the targeted system.

3.1. Compile-time Reconfiguration

Compile-time reconfiguration (CTR) relies on the instrumentation of HDL models. The idea is to introduce in the circuit description (before its mapping to the FPGA) some suitable logic able to support the fault injection and the observation of its behaviour. This approach does not require FPGA reconfiguration for each injection, saving reconfiguration time.

Figure 2 shows the schema for a single experiment. Different sets of experiments (for different fault models or injection points) may need different instrumented models, so synthesization and implementation must be done for each case.

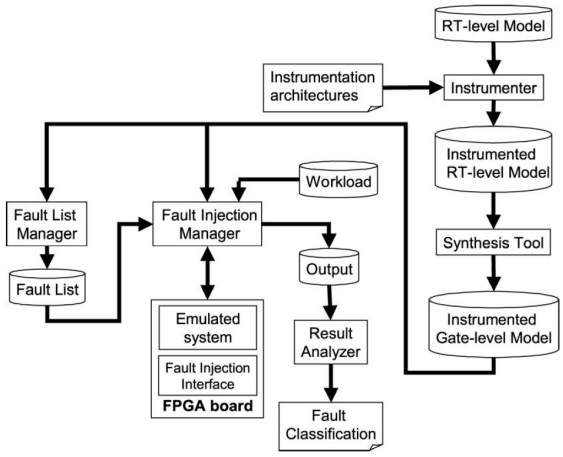


Figure 2. The CTR Fault Injection flow [15]

3.2. Run-time Reconfiguration

Run-time reconfiguration (RTR) is a methodology for the development of reconfigurable applications

[17]. It relies on the reconfiguration capabilities of programmable devices to change the behaviour of the system on-the-fly. Hence, the use of RTR techniques for fault emulation involves the reconfiguration of the system to emulate its behaviour in the presence of faults while executing a representative workload.

Figure 3 shows a high-level schema for the process. The fault is virtually injected in the HDL model, but in practice it is emulated using an FPGA [13].

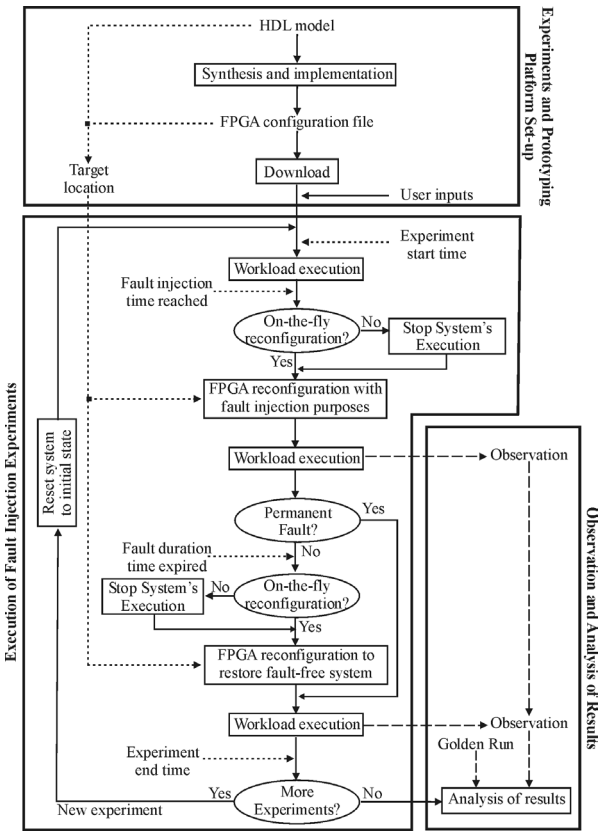


Figure 3. RTR for Fault Emulation [13]

3.3. Comparison CTR-RTR

In CTR, once the model is synthesized, mapped and placed on the FPGA fault injections are faster than RTR because system execution is not stopped. Although the time needed for the FPGA reconfiguration is very short, it increases emulation time in RTR.

If we use CTR we have to build an instrumented model. It may cause the new model not to accomplish timing requirements or to present space problems.

The main advantage of RTR against CTR is that it avoids HDL model changes. Consequently, the synthesis and implementation of the model is done only once. In RTR it can be necessary to syntesize, map and place a different modified model on the FPGA for each set of experiments, losing time.

4. Research objectives

During last years, the Fault-Tolerant Systems Research Group (GSTF) has developed several fault injection tools to carry out the study of different dependability parameters of Fault-Tolerant Systems. These tools can inject physical faults at pin-level (AFIT [11]), software implemented fault injection (INERTE [18]) and VHDL-based fault injection (VFIT [19]).

Particularly, VFIT is an automatic tool for Fault Injection in VHDL models. One of the main problems of HDL-based Fault Injection techniques is the excessive time they may require for model simulation, as stated by our experience.

The group is also working in Fault Emulation. Our research shows that not all models can be synthesized, so Fault Emulation is not always available.

With the emergence of Fault Emulation, it seems reasonable to combine Fault Emulation and Fault Simulation. The idea is to get the advantages of both methodologies, while reducing their disadvantages. This means to determine, for a given model and a given set of experiments:

- The right technique if only one is possible.
- The best if both are available.
- A combination if we are able to distribute injections of different model parts, with the best technique for each one.

The question is how to carry out this integration. Would it be possible to validate a system using a co-validation methodology?

The combination of Fault Simulation and Fault Emulation approaches in a single technique has a great potential for defining a fast model driven validation methodology for VLSI systems, which can be useful from the very early systems' design phases. Despite this potential, such integration of solutions poses a number of questions that remains unanswered. Which criteria can be used to determine which approach is more appropriate for the validation of a given system or system component? Is it feasible to apply both Fault Injection approaches to different system parts? How can experimental results be combined in order to provide a single set of dependability and performance measures? These

questions refer to what we call Fault Emulation-Fault Simulation co-validation.

As soon as a model is available, it is possible to carry out its validation. According to the model type (behavioral or structural), Fault Emulation or Fault Simulation can be selected, that is, a hardware tool or a software tool can be chosen. So, a decision about dividing a design into hardware and software has to be done, driving us to a co-validation framework.

If some rules can be defined depending on the model, the selection of one of the techniques can be done automatically. If both techniques can be used, the designer could select one. Fault Emulation can be the best option, as it is faster, but it could be interesting to use Fault Simulation in some cases (to have better accessibility, for example).

In this way, a co-validation framework can be developed. Some ideas for the methodology:

- Fault Emulation is the faster technique, so it will be proposed as the first option if it is available.
- If the HDL model can not be implemented into an FPGA, Fault Simulation should be used.
- If the HDL model is really complex, and it can not be synthesized, the solution is to synthesize some parts of the model. These parts should be the ones to validate.
- If the fault model to inject is not supported by Fault Emulation, Fault Simulation should be used.
- If the fault model to inject is not supported by Fault Emulation and the system is really complex, Fault Simulation should be used to inject the faults in the selected parts of the model, and Fault Emulation should simulate the rest of the system.
- If both techniques are proposed, the tester should select one of them.

As it can be seen, these rules are defined according to the model. In this way, it is possible to define design techniques in order to apply one of the injection techniques, something like co-validation.

A challenging question arises if we try to simulate different model parts with different techniques. We have to define inputs and outputs for each model part. The communication between the software tool for Fault Simulation and the FPGA seems to be possible, due to the good controllability of HDL simulators and the input/output capabilities of modern FPGA drivers. One of the final objectives in the implementation phase of this methodology is to make the communication possible with an affordable temporal cost.

Another interesting issue is to use web services to share this co-validation framework. Using web protocols this methodology can be available to anyone connected to the web.

To sum up, the use of Fault Emulation or Fault Simulation depends on the model, the language of implementation of the model, the fault model desired for testing, the time available to get the results, etc. The objective of my research line is to get a methodology for system models co-validation, based on all these questions. A tool based on this methodology should also be developed in the future. If we are able to get good results with this methodology, it will be a very useful tool for designers, in order to get shorter design cycles.

5. Acknowledgements

I would like to thank Joaquín Gracia for his valuable help and support supervising this paper. His work about Fault Simulation will be very important for my future research.

I wish to thank David de Andrés for his work and suggestions about Fault Emulation. He had also reviewed this paper. Thanks for your effort.

I would also like to thank Pedro J. Gil, our research director. Joaquín Gracia and him will supervise my work until I get my PhD.

Finally, I wish to thank the EDCC 6 Student Forum Program Committee for this opportunity to share my ideas with the scientific community. Special thanks to the reviewers for their valuable suggestions and efforts.

6. References

[1] J.A. Clark and D.K. Pradhan, “Fault Injection: A Method for Validating Computer-System Dependability”, IEEE Computer, 28(6):47-56, June 1995.

[2] D. Gil, J.C. Baraza, J. Gracia, P.J. Gil “VHDL Simulation-Based Fault Injection Techniques”, in Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Edited by A. Benso and P. Prinetto, Kluwer Academic Press, pp. 159-176, 2003.

[3] R. Leveugle, “Fault Injection in VHDL Descriptions and Emulation”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’00), pp. 414-419, Japan, October 2000.

[4] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, “Exploiting FPGA-based Techniques for Fault Injection Campaigns on VLSI Circuits”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’01), USA, pp. 250-258, October 2001.

[5] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993.

[6] IEEE Standard Verilog Language Reference Manual, IEEE Std. 1364-1995.

[7] H.R. Zarandi, S.G. Miremadi, A. Ejlali, “Fault Injection into Verilog Models For Dependability Evaluation of Digital Systems”, Proc. 2nd International Symposium on Parallel and Distributed Computing (ISPDC’03), pp.281-287, Slovenia, October 2003.

[8] J.C. Baraza, J. Gracia, D. Gil, P.J. Gil, "Improvement of Fault Injection Techniques Based on VHDL Code Modification", Proceedings of the IEEE International High Level Design Validation and Test Workshop 2005 (HLDVT 2005), pp. 19-26, Napa Valley Marriott Hotel & Spa, California, USA, November 2005.

[9] D. Gil, J. Gracia, J.C. Baraza, P.J. Gil, “Study, Comparison and Application of different VHDL-Based Fault Injection Techniques for the Experimental Validation of a Fault-Tolerant System”, Microelectronics Journal, 34(1):41-51, January 2003.

[10] B. Parrotta, M. Rebaudengo, M. Sonza Reorda, M. Violante, “Speeding-up Fault Injection Campaigns in VHDL models”, 19th International Conference on Computer Safety, Reliability and Security (SAFECOMP’00), pp. 27-36, The Nederlands, October 2000.

[11] S. Blanc, J. Gracia, P.J. Gil, “A Fault Hypothesis Study on the TTP/C using VHDL-based and Pin-level Fault Injection Techniques”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’02), pp. 254-262, Canada, November 2002.

[12] J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, “Using VHDL-Based Fault Injection for the Early Diagnosis of a TTP/C Controller”, IEICE Transactions on Information and Systems, Vol. E86-D(12):2634-2641, December 2003.

[13] D. de Andrés, J.C. Ruiz, D. Gil, P.J. Gil, “Run-Time Reconfiguration for Emulating Transient Faults in VLSI Systems”, to be published in Int. Conf. on Dependable Systems and Networks 2006 (DSN’06), USA, June 2006

[14] D. de Andrés, J.C. Ruiz, D. Gil, P.J. Gil, “Fast Emulation of Permanent Faults in VLSI Systems”, to be published in Procs. IEEE 16th Int. Conf. of Field Programmable Logic and Applications (FPL’06), Spain, August 2006.

[15] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, “New Techniques for efficiently assessing reliability of SOCs”, Microelectronics Journal, 34(1):53–61, 2003.

[16] L. Antoni, L. Leveugle, and B. Fehér, “Using Run-Time Reconfiguration for Fault Injection Applications”, IEEE Trans. on Instrumentation and Measurement, 52(5):1468–1473, 2003.

[17] B. L. Hutchings, M. J. Wirthlin, “Implementation Approaches for Reconfigurable Logic Applications”, Int. Workshop on Field Programmable Logic and Applications, pp. 293–302, UK, 1995.

[18] P. Yuste, D. de Andrés, L. Lemus, J.J. Serrano, P.J. Gil, “INERTE: Integrated NExus-based Real-Time fault injection tool for Embedded systems”, Procs. 2003 International Conference on Dependable Systems and Networks (DSN’03), pp. 669, USA, June 2003.

[19] J.C. Baraza, J. Gracia, D. Gil, P.J. Gil, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, Journal of Systems Architecture, 47(10):847-867, April 2002.

Analysis of the influence of intermittent faults in a microcontroller

J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) - Departamento de Informática de Sistemas y Computadores (DISCA)
Universidad Politécnica de Valencia, Spain
e-mail: {jgracia, ljsaiz, jcbaraza, dgil, pgil}@disca.upv.es

***Abstract*— Nowadays, new submicron technologies have allowed increasing processors performance while decreasing their size. However, as a side effect, their reliability has been negatively affected. Although mainly permanent and transient faults have been studied, intermittent faults are expected to be a big challenge in modern VLSI circuits. Usually, intermittent faults have been assumed to be the prelude of permanent faults. Currently, intermittent faults due to process variations and residues have grown, being necessary to study their effects.**

The objective of this work has been to analyse the impact of intermittent faults, taking advantage of the power of the simulation-based fault injection methodology. Using as background faults observed in real computer systems, we have injected intermittent faults in the VHDL model of a microcontroller. The controllability and flexibility of VHDL-based fault injection technique has allowed us to do a detailed analysis of the influence of some parameters of intermittent faults. We have also compared the results obtained with the impact of transient and permanent faults.

***Index Terms*— VHDL-based fault injection, Intermittent faults.**

I. Introduction

During last years, advances in integration techniques have allowed rising microprocessors operating frequency as well as reducing their size and power voltage (V_{DD}), achieving in this way a greater productivity. Nevertheless, these advances have a negative impact on reliability. As the new submicron technologies shrink device sizes, the rate of occurrence of faults increases [1]. The consequence is that designers have to deal with an increasing number of different fault types due to manufacturing defects.

Usually, only the effects of permanent and transient faults have been studied [2][3]. Permanent faults are provoked by irreversible physical defects due to manufacturing defects and wearout mechanisms. Transient faults are commonly generated by environmental conditions like electromagnetic interferences and cosmic radiation. The increase on integration scale has caused that transient faults normally observed in space are now common at sea level [4][5]. It is expected that transient faults will have a greater influence than permanent faults in the new VLSI circuits [6].

Intermittent faults are another type of fault not considered normally. It is foreseen that intermittent faults will have a great impact in deep submicron technologies, together with transient faults. The complexity of the manufacturing process (that provokes residuals and process variations) and some wearout mechanisms can be the origin of such faults [6][7][8].

Errors induced by transient and intermittent faults manifest in a similar way, but the last ones are activated repeatedly in the same place, and they usually are grouped in bursts [9]. On the other hand, replacement of the affected part eliminates an intermittent fault, while transients cannot be fixed by repair. Additionally, intermittent faults may be deactivated or activated by temperature, voltage and frequency changes [10].

So, intermittent faults are a big challenge in modern VLSI circuits. Our objective in this work is to study the impact of intermittent faults, and compare their consequences to those provoked by permanent and transient faults.

A common method to study experimentally the dependability parameters of a microprocessor is Fault Injection [11]. This technique allows a controlled introduction of faults in the system, not being necessary to wait a long time for logging the apparition of real faults.

Fault injection techniques can be classified in three main categories [3]: physical (or Hardware Implemented Fault Injection, HWIFI), software implemented (SWIFI) and simulation-based.

Simulation-based fault injection is a useful experimental way to evaluate the dependability of a system during the design phase. An early diagnosis allows saving costs in the design process, avoiding redesigning in case of error, and thus reducing time-to-market. We have used VHDL-based fault injection due to its flexibility, as well as the high observability and controllability of all the modelled components [12]. Moreover, VHDL is a widely utilized hardware description language in academic and industrial realms [13].

So far, very few tries have been done in order to study the effects of intermittent faults by fault injection. In most of the works issued, real systems were monitored, and faults and failures produced were observed to determine the most frequent sources of errors and their manifestation [10].

Our intention with this work is to study systematically and exhaustively the impact of the parameters of intermittent faults on the behaviour of the system, as well as to compare to the effects of permanent and transient faults, taking advantage of the power of fault injection.

Using as background the existing information about observed and logged errors in real systems, we have generated the adequate fault models to inject, according to the abstraction level of the system under study.

Intermittent faults have been injected into the VHDL model of a typical microcontroller and their effects have been compared with permanent and transient faults.

The organization of the paper is as follows. In Section II we describe the intermittent fault models injected. Section III depicts the fault injection environment. In Section IV, the fault injection experiments are explained, and the results obtained are discussed in Section V. Finally, some conclusions and a proposal of future work are provided in Section VI.

II. Intermittent Fault Models

When performing fault injection campaigns, one of the most important questions is fault representativeness, i.e., the equivalence of the supported fault model with respect to actual faults. When injecting faults in a system model, the fault models applied must be suitable to the abstraction level of the system model. Our objective in this work is to use fault models at logic and RT levels.

Hardware fault representativeness for transient and permanent faults has been extensively studied [2][14][15]. Nevertheless, and as far as we know, the causes and mechanisms of intermittent faults in new submicron technologies have not been studied so much.

One possibility is to use similar models to those for permanent faults. In fact, in a wearout process, faults initially may appear intermittently but eventually result in permanent faults [2][16].

However, the introduction of new submicron technologies makes necessary to keep in mind also the influence of manufacturing defects, like process variations and residuals [6][17]. A deeper study is necessary, being this an open line of research.

For these reasons, we have decided in this paper to select a set of intermittent faults frequently observed in real computer systems by means of fault logging. Then, we have extrapolated their effects to fault models applicable to the logic/RT abstraction levels. In this way, we can inject and simulate these faults into VHDL models.

Figure 1 shows the main causes and mechanisms of the intermittent faults considered [10], as well as the associated fault models.

Bursts of Single Bit Errors (or SBEs) have been observed in memory [18]. Failure analysis found that polymer residues led to an intermittent contact. The fault model chosen to represent a SBE in memory and register cells is an *intermittent stuck-at*, meaning that the value of the cell changes intermittently between the two logical values. The origin of the fault lies in an unstable hardware. This is not related to an external environmental effect (e.g. cosmic radiation). Thus, it is not an intermittent bit-flip, so we have called it *intermittent stuck-at*.

Bursts of SBEs have also been observed in data buses [18]. Failure analysis revealed that the source of errors were solder intermittent contacts. We have associated this mechanism to the following fault models [12]: *intermittent pulses*, *short* and *open* in bus lines.

Also, timing failures may occur due to propagation delays over the interconnection lines. For instance, barrier layer material delamination and electromigration lead to a higher resistance and, as a result, to timing violations [19]. Moreover, they can provoke *intermittent short/open* faults in metal lines. Crosstalk delays may occur when adjacent signals switch in opposite directions, and are related to capacitive effects in adjacent metallic lines. The fault models assigned to these mechanisms are *intermittent delay* and *short/open* in interconnection lines and buses [12].

Another type of causes and mechanisms of intermittent faults are due to oxide defects and wearout. Soft breakdown belongs to this class. In this case, the leakage current of the gate oxide fluctuates in time, without inducing the thermal damage [20]. This produces erratic fluctuations of the minimum supply voltage, that can be modelled with the *intermittent indetermination* fault model [12].

In the future, we want to carry out a deeper study of the fault mechanisms related with wearout processes and process variations in new submicron technologies.

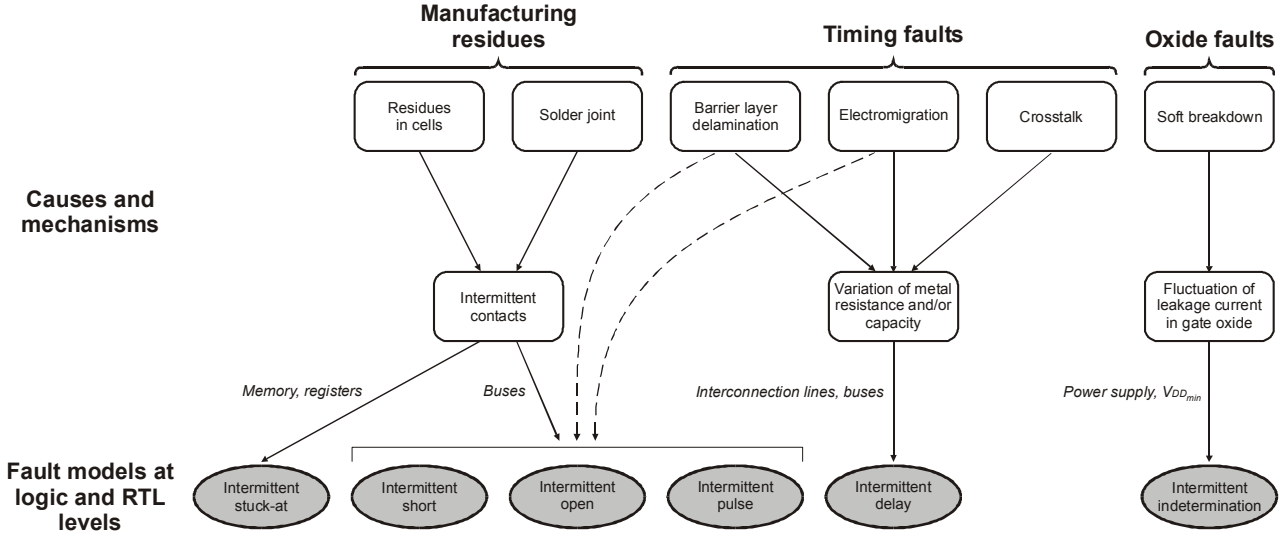


Figure 1. Some representative mechanisms and models for intermittent faults

III. The Fault Injection Environment

The *Grupo de Sistemas Tolerantes a Fallos* (GSTF) have developed a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [21], that runs on PC computers (or compatible) under Windows® and is model-independent. Although it admits VHDL models at any abstraction level, it has been mainly used on models at logic and RT levels.

With VFIT it is possible to inject faults applying simulator commands, saboteurs and mutants techniques [3][2].

VFIT can inject a great range of faults according to various aspects. With regard to the fault duration, it can inject permanent, transient and intermittent faults. In relation to the number of faults injected, it is possible to inject faults in a single location (i.e. Single Bit Faults or SBFs) or in multiple locations, (that is, Multiple Bit Faults, or MBFs), either adjacent and/or non adjacent. On the other hand, independently of the number of fault targets, these faults can be single or multiple in time domain. That is, multiple occurrences of the fault can be injected along the simulation time.

When applied to models at logic and RT levels, VFIT uses a wide set of fault models that try to be representative of deep submicron technologies, as it can be seen in Table I. This table shows models of transient, permanent and intermittent faults. As commented in Section II, intermittent fault models are not still well established, and are an open research subject. Also in Section II, we have proposed some models for frequent intermittent faults observed in real systems. We have reflected these fault models in Table I, according to the injection technique that can inject them.

The output of an injection experiment is a set of tables whose values depend on the objective of the injection experiment. In case of an error syndrome analysis, output tables contain among other values: propagation latencies, percentages of propagated errors and percentages of failures. In case of performing a dependability validation, tables show propagation, detection and recovery latencies, percentages of propagated, detected, and recovered errors, detection and recovery coverages, and failure percentages.

IV. Fault Injection Experiments

A. Notation

As mentioned above, intermittent faults manifest in bursts. That is, when a fault appears, it does not activate only once. Instead, it activates and deactivates repeatedly several times until the fault finally disappears. So, to inject intermittent faults, the following parameters must be configured: the number of activations in the burst (that we will call the *burst length*, or L_{Burst}), the duration of each activation (we will refer to it as the *activity time*, or t_A), and the separation between two consecutive activations (we will name it as the *inactivity time*, or t_I). Figure 2 illustrates this explanation.

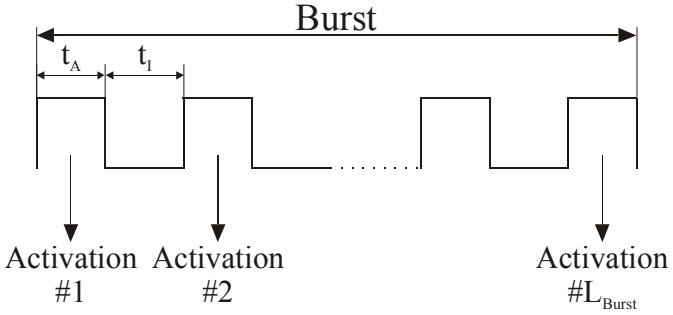


Figure 2. Description of the main elements of a burst.

The following terms and notation are used in the remaining of this paper:

Fault simulation	A simulation of the model of the system under study in presence of faults;
Injection experiment	A set of fault simulations configured with the same injection parameters;
Injection campaign	A set of injection experiments in which a number of injection parameters can vary;
t_A	Activity time;
t_I	Inactivity time;
L_{Burst}	Burst length;
t_{inj}	Injection instant;
$N_{Injected}$	Number of faults injected;
$N_{Propagated}$	Number of propagated errors, defined in this campaign as the injected faults that propagate to the microprocessor registers and memory;
$N_{Failures}$	Number of failures, defined as the number of propagated errors that provoke failures. A failure is produced when the result is erroneous at the end of the simulation time;
N_{Latent}	Number of latent errors. A latent error is a propagated error that does not provoke a failure;
$N_{Non\_effective}$	Number of non effective faults. It is the number of faults injected that have not produced either a failure or a latent error;
$P_{Failures}$	Percentage of failures;
P_{Latent}	Percentage of latent errors;
$P_{Non\_effective}$	Percentage of non effective errors.

TABLE I. FAULT MODELS INJECTED BY VFIT

Injection technique	Transient faults	Permanent faults	Intermittent faults
Simulator commands	Pulse*, Bit-flip**, Indetermination, Delay	Stuck-at (0,1), Indetermination, Open, Delay	Intermittent stuck-at (0,1), Pulse***, Indetermination, Open
Saboteurs	Pulse*, Bit-flip**, Indetermination, Delay	Stuck-at (0,1), Indetermination, Open, Delay, Short, Bridging, Stuck-open	Intermittent stuck-at (0,1), Pulse***, Indetermination, Open, Delay, Short
Mutants	Syntactical changes	Syntactical changes	Not yet defined

* In combinational logic and interconnection lines. Represents a Single Event Transient (SET)
** In storage elements (registers and memory). Represents a Single Event Upset (SEU)
*** Sequence of pulses due to intermittent fault mechanisms

B. Experiment set-up

We have scheduled two injection campaigns for intermittent faults, where the injection targets are the storage elements, that is, the register bench and the memory. The injection campaigns differ on the number of injection targets. In the first campaign we have injected SBFs (that is, only one intermittent fault in a single bit location). In the second one, we have injected MBFs in multiple adjacent and non-adjacent locations. The aim of these injection campaigns is to analyse the influence of the target size (single or multiple) and the activity time (t_A).

On the other hand, we have also injected transient and permanent faults in order to compare them to intermittent faults. The different campaigns have been carried out in a VHDL model of the 8051 microcontroller [22] running the Bubblesort sorting algorithm as a workload. Although the chosen system has not a deep submicron technology, the methodology used can be generalised to more complex microprocessors/microcontrollers.

The most relevant injection parameters are the following:

1) *Injection targets:* The register bench and the memory.
2) *Injection technique:* Simulator commands.
3) *Number of faults per experiment:* 1000 single or multiple adjacent and non adjacent faults per experiment[1].
4) *Fault models:*
   - For transient faults: Bit-flip;
   - For permanent faults: Stuck-at(0,1), Open, and Indetermination;
   - For intermittent faults: Intermittent stuck-at(0,1) (see Section II).
5) *Fault types and duration:*
   - Transient single and multiple in space[2];
   - Permanent single and multiple in space;
   - Intermittent single and multiple in space, with
     - an activity time (t_A) defined according to a Uniform distribution function in the ranges [0.01T, 0.1T], [0.1T, 1.0T], and [1.0T, 10.0T], where T is the CPU clock cycle;
     - an inactivity time (t_I) defined according to a Uniform distribution function in the ranges [0.01T, 0.1T], [0.1T, 1.0T], and [1.0T, 10.0T].
     - a burst length defined according to a Uniform distribution function in the range [2, 9].

The objective of the campaigns is to inject intermittent faults where bursts have different shapes and lengths.

In order to check the effects of the intermittent faults, we have calculated in all cases the following data:

- The percentage of failures:

$$P_{Failures} = \frac{N_{Failures}}{N_{Injected}} \times 100$$

  A failure is detected by comparing the golden run with the faulty trace, and checking disparity in the result registers.
- The percentage of latent errors:

$$P_{Latent} = \frac{N_{Latent}}{N_{Injected}} \times 100$$

  where N_{Latent} can be obtained as ($N_{Propagated} - N_{Failures}$) or as ($N_{Injected} - N_{Failures} - N_{Non\_effectivet}$).
  A latent error is detected by checking changes in the content of registers or memory at the end of simulation. Checked registers are not the result registers.

[1] This implies that, for instance, in a given fault simulation, faults might be injected simultaneously in bits 3 and 2 of the 37th memory byte, and in bits 7, 6, and 5 of register #6.

[2] A multiple fault in space are simultaneous faults produced in various targets of the model.

V. RESULTS

Table II shows the percentages of failures, latent errors and non effective faults for single intermittent faults, while Table III shows the same percentages for multiple intermittent faults. In both tables, we have varied the duration of the pulses of the burst, i.e., the activity time. When increasing the activity time, we can observe an augment of the percentage of failures. This is a logical result if we think that a longer duration of the intermittent pulses must have a bigger impact on the system.

In relation to the percentage of latent errors, a decrease is observed, because they are propagated errors that do not provoke failures. As $N_{Propagated} = N_{Failures} + N_{Latent}$, an increase in the first addend causes a drop in the second, since they are exclusive cases. We can also verify that $P_{Failures} + P_{Latent} + P_{Non\_effective} = 100\%$, as $N_{Failures} + N_{Latent} + N_{Non\_effective} = N_{Injected}$.

TABLE II. RESULTS FOR SINGLE INTERMITTENT FAULTS

% \ t_A	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	0.1 %	0.4 %	3.9 %
Latent errors	41.9 %	8.9 %	0 %
Non effective	58.0 %	90.7 %	96.1 %

TABLE III. RESULTS FOR MULTIPLE INTERMITTENT FAULTS

% \ t_A	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	1.7 %	15.5 %	53.0 %
Latent errors	43.0 %	6.4 %	0 %
Non effective	55.3 %	78.1 %	47.0 %

Figure 3 and Figure 4 show the influence of the spatial multiplicity of faults (Single vs. Multiple faults) in the percentage of failures and latent errors. We show results for different activity times. We can see in Figure 3 that multiple faults provoke a notably bigger number of failures than single faults. This is an expected behaviour, because multiple faults affect simultaneously various physical zones of the system (in this case memory and register bench). We can observe also that the difference between $P_{Failure}$ for multiple and single faults grows with the activity time.

About latent errors, Figure 4 does not reflect a substantial difference between single and multiple faults. This implies that additional areas affected by multiple faults have provoked failures in most cases, and not latent errors. Nevertheless, we think that the dependence on the workload is very strong and it is difficult to generalize the results.

Figure 5 and Figure 6 compare the impact of intermittent faults respect to permanent and transient faults. We have also represented the influence of the spatial multiplicity (single vs. multiple faults).

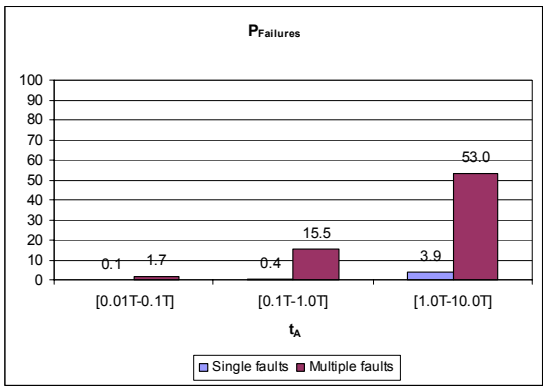


Figure 3. Influence of the spatial multiplicity of faults in the percentage of failures.

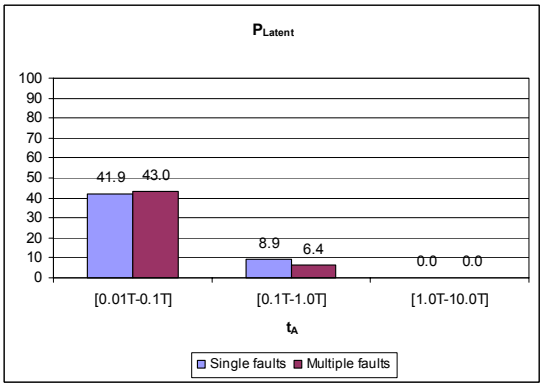


Figure 4. Influence of the spatial multiplicity of faults in the percentage of latent errors.

As expected, Figure 5 shows that permanent faults provoke a bigger percentage of failures than transient faults, because permanent faults have an indefinite duration. On the other hand, we can see that the impact of intermittent faults is lower than transient faults, even for the bigger activity time. We expected a bigger impact of intermittent faults, because they are actually a sequence of transient faults that form the burst. However, a more detailed analysis of the injected fault models and the workload allows establishing the cause of this anomalous behaviour. The characteristics of the used workload provokes a low rate of overwrite operations in the memory cells affected by transient faults (bit-flips), causing a *de facto* indefinite duration of these faults, remaining in this way stored permanently.

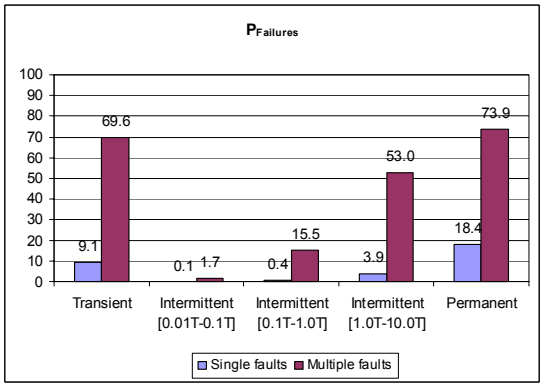


Figure 5. Comparison of the percentage of failures provoked by transient, intermittent and permanent faults.

Figure 6 shows that intermittent faults have provoked a bigger percentage of latent errors than transient and permanent faults. This type of faults propagates into memory cells or registers, but they do not provoke failures, although the system remains contaminated.

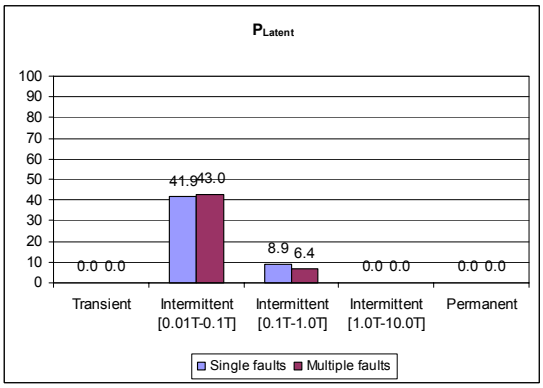


Figure 6. Comparison of the percentage of latent errors provoked by transient, intermittent and permanent faults.

Finally, Figure 7 and Figure 8 show the influence of the inactivity time, that is, the temporal separation between the pulses of the burst (see Section IV). No significant differences between single and multiple faults are observed when varying the inactivity time. The separation of the pulses of the burst does not provoke a noteworthy variation in $P_{Failure}$ or P_{Latent} (assuming that all pulses of the burst are present). However, the separation between pulses is decisive for the system to recover when detection and recovery mechanisms are introduced in the system. These mechanisms should be as fast as possible [10].

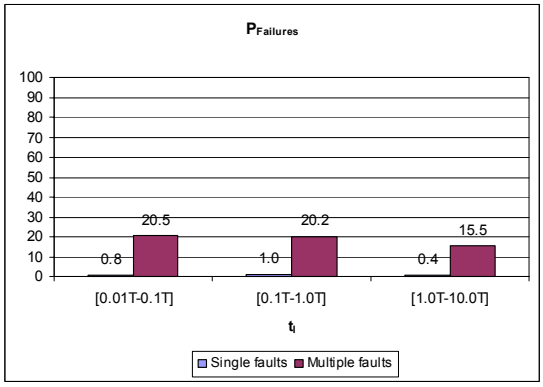


Figure 7. Percentage of failures respect to the inactivity time.

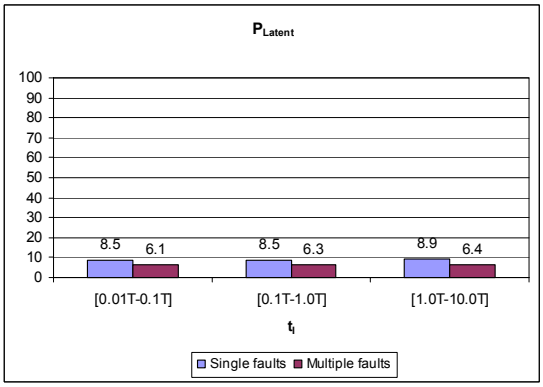


Figure 8. Percentage of latent errors respect to the inactivity time.

VI. Conclusions and Future Work

In this work, we have presented an exhaustive study of the effects of intermittent faults on the behaviour of a commercial microcontroller. The method used has been fault injection. In order to choose the intermittent fault models to inject, we have selected some faults that have been observed in real computer systems, and we have adapted their effects to a logic representative level.

We have used the VHDL-based fault injection technique to carry out the injection experiments. This technique allows a great flexibility, controllability and observability. The different injection experiments have been performed using VFIT, a VHDL-based fault injection tool developed by our research group.

In the different injection experiments, we have studied the influence of the variation of different parameters of intermittent faults. We have compared also the results obtained when injecting transient and permanent faults.

In general terms, and assuming the important influence of the workload used, it has been observed that:

- The activity time, that is, the width of the pulses is the most significant parameter of intermittent faults. We have not observed important changes respect to the separation of the burst pulses.
- The spatial multiplicity of intermittent faults presents also a notable impact in the behaviour of the system.
- Intermittent faults have provoked a lower percentage of failures than permanent and transient faults. This is caused by the specific characteristics of the used workload and the transient fault model. When using more general workloads, we expect that intermittent faults will provoke more failures than transient ones.
- Intermittent faults have provoked a bigger percentage of latent errors than transient and permanent faults. This type of faults propagates into memory cells or registers, but they do not provoke failures, although the system remains contaminated.

In the future, we want:

- To use other workloads and other microprocessors in order to try to generalize the results;
- To inject faults in other critical places, like control and data buses;
- To analyze the effect of other parameters, like the burst length and the system clock frequency;
- To inject other observed fault models in buses, like delay, short and open;
- To investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, and propose and implement new related fault models.

Acknowledgement

This work has been partially funded by the Spanish Government under the Project MCYT TEC2005-05119 “Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida”.

References

[1] C. Constantinescu, “Trends and Challenges in VLSI Circuit Reliability”, IEEE Micro, Vol. 23(4):14-19, 2003.

[2] P.J. Gil *et al.*, “Fault Representativeness”, *Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245*, 2002.

[3] A. Benso and P. Prinetto, eds., “Fault Injection Techniques and Tools for VLSI reliability evaluation”, Kluwer Academic Publishers, 2003.

[4] D. Alexandrescu, L. Anghel, and M. Nicholaidis, “Simulating SET in VDSM ICs for Ground Level Radiation”, in JETTA(20):413–421, 2004.

[5] C. Constantinescu, “Neutron SER Characterization of Microprocessors”, Procs. 2005 International Conference on Dependable Systems and Networks (DSN’05), pp. 754-759, Yokohama, Japan 2005.

[6] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[7] D. Barros Jr, F. Vargas, M.B. Santos, I.C. Teixeira, and J.P. Teixeira, “Modeling and Simulation of Time Domain Faults in Digital Systems”, Procs. 10th IEEE International On-Line Testing Symposium (IOLTS’04), pp. 5-10, Portugal, July 2004.

[8] N. Kranitis, A. Merentitis, N. Laoutaris, and G. Theodorou, “Optimal Periodic Testing of Intermittent Faults In Embedded Pipelined Processor Applications”, Design, Automation and Test in Europe (DATE’06), Vol. 1, pp. 1-6, March 2006.

[9] C. Constantinescu, “Intermittent Faults in VLSI Circuits”, 2nd Workshop on Silicon Errors in Logic - System Effects (SELSE2), April 2006.

[10] C. Constantinescu, “Impact of Intermittent Faults on Nanocomputing Devices”, in WDSN-07, Edinburgh, UK, June 2007. http://www.laas.fr/WDSN07.

[11] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell, “Fault Injection for Dependability Validation: A Methodology and Some Applications”, IEEE Transactions on Software Engineering, vol. 16(2):166–182, 1990.

[12] D. Gil, J. Gracia, J.C. Baraza, and P.J. Gil, “Study, Comparison and Application of different VHDL-Based Fault Injection Techniques for the Experimental Validation of a Fault-Tolerant System”, Microelectronics Journal, vol. 34(1):41-51, 2003.

[13] "IEEE Standard VHDL Language Reference Manual", IEEE Std 1076-1993.

[14] E.A. Amerasekera and F.N. Najm, "Failure mechanisms in semiconductor devices", John Whiley & Sons, 1997.

[15] J. Gracia, D. Gil, L. G. Lemus and P. J. Gil, “Studying Hardware Fault Representativeness with VHDL Models”, in Proc. XVII Conference on Design of Circuits and Integrated Systems (DCIS’02), pp. 33-39, Santander, Spain, November 2002.

[16] J.C. Smolens, B.T. Gold, J.C. Hoe, B. Falsafi, and K. Mai. “Detecting Emerging Wearout Faults”, 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[17] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, and V. De, “Parameter Variations and Impact on Circuits and Microarchitecture”, in Procs. 40th Conference on Design Automation (DAC 2003), pp. 338 – 342, April 2003.

[18] C. Constantinescu, “Dependability Benchmarking using Environmental Test Tools”. Procs. Reliability and Maintainability Symposium (RAMS 2005), pp. 567–571, January 2005.

[19] J.W. McPherson, “Reliability challenges for 45nm and beyond”, Procs. Conference on Design Automation (DAC 2006), pp. 176–181, July 2006.

[20] J.H. Stathis, “Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits”, IEEE Transactions On Device And Materials Reliability, Vol. 1(1):43–59, March 2001.

[21] J.C. Baraza, J. Gracia, D. Gil, and P.J. Gil, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, Journal of Systems Architecture, vol. 47(10):847–867, 2002.

[22] http://www.oregano.at.

Applying Fault Injection to Study the Effects of Intermittent Faults

L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA)
Universidad Politécnica de Valencia, Spain
e-mail: {ljsaiz, jgracia, jcbaraza, dgil, pgil}@disca.upv.es

Abstract

As new submicron technologies have allowed increasing processors performance and decreasing their size, their reliability has decreased. In this way, intermittent faults are expected to be an important challenge in modern VLSI circuits. Currently, intermittent faults due to manufacturing residuals and process variations have grown, being necessary to study their effects. Using as background faults observed in real computer systems, we have injected intermittent faults in the VHDL model of a microcontroller. The controllability and flexibility of VHDL-based fault injection technique permits to carry out a detailed analysis of the influence of some parameters of intermittent faults.

1. Introduction

The advances in integration techniques have allowed rising microprocessors operating frequency as well as reducing their size and power voltage, achieving in this way a greater performance. Nevertheless, these advances have had a negative impact on reliability. As the new submicron technologies shrink device sizes, the rate of occurrence of faults increases. Whereas the effects of permanent and transient faults have been studied in depth [1], the influence of intermittent faults is not considered very often. It is foreseen that intermittent faults (together with transient faults) will have a great impact in deep submicron technologies [2]. The complexity of the manufacturing process (that provokes residuals and process variations) and some wear out mechanisms can be the origin of such faults. Although errors induced by transient and intermittent faults manifest in a similar way, the last ones are activated repeatedly in the same place, and so they are usually grouped in bursts [3].

Our objective is to study the impact of intermittent faults, and compare their consequences to those provoked by permanent and transient faults. We have used VHDL-based fault injection due to its flexibility and the high observability and controllability of all the modelled components [4]. In addition, fault injection allows performing a systematic and exhaustive analysis of the influence of the parameters of intermittent faults on the system's behaviour.

This work is organised as follows. Section 2 describes the intermittent fault models injected. Section 3 explains the fault injection experiments, and Section 4 discusses the results obtained. Finally, Section 5 provides some conclusions and a proposal of future work.

2. Fault models

An important issue is the definition of the fault models to be injected, as they must be representative of real faults. This requires studying the fault mechanisms considering the features of the technology used. Nevertheless, and as far as we know, the causes and mechanisms of intermittent faults in new submicron technologies have not been studied so much. Intermittent faults can be related to wear out mechanisms that eventually end up in permanent faults [5]. But it is also becoming more and more necessary to keep in mind the influence of manufacturing defects, like process variations and residuals.

In this paper we have selected a set of intermittent faults frequently observed in real computer systems by means of fault logging [3]. Then, we have extrapolated their effects to fault models applicable to the logic/RT abstraction levels. In this way, we can inject and simulate these faults into VHDL models. Table 1 summarises these intermittent faults and their mechanisms and models.

Table 1. Intermittent faults mechanisms and models

Causes	Targets	Fault mechanisms	Type of fault	Fault Models
Residuals in cells	Memory and registers	Intermittent contacts	Manufacturing defect	Intermittent stuck-at
Solder joints	Buses	Intermittent contacts	Manufacturing defect	Intermittent pulses, short, open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance	Wear out-timing	Delay, intermittent short, open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing	Delay
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wear out-timing	Indetermination, delay

3. Fault injection experiments

As stated above, intermittent faults manifest in bursts. So, to inject this type of faults, the following parameters must be configured (see Figure 1): the number of activations of the fault in the burst (we will call it *burst length*, or L_{Burst}), the duration of each activation (we will refer to it as *activity time*, or t_A), and the separation between two consecutive activations (we will name it as *inactivity time*, or t_I).

The different fault injection experiments have been carried out in the VHDL model of the 8051 microcontroller running the Bubblesort sorting algorithm as workload. Fault injection experiments have been carried out using a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [6], that runs on PC computers (or compatible) under Windows®.

In order to study the effects of intermittent faults, we have calculated in all cases the percentages of *failures* provoked and *latent errors* (that is, errors that propagate to registers and memory but do not cause failures).

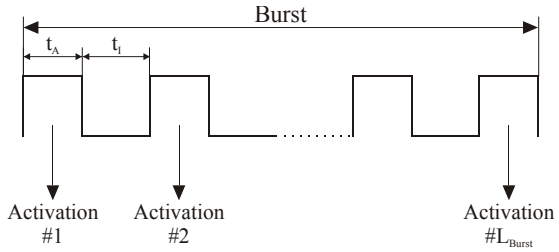


Figure 1. Description of the main elements of a burst

4. Results

Table 2 to Table 5 show the effect of intermittent faults injected in two types of targets: memory and registers (Table 2 and Table 3), and buses (Table 4 and Table 5). The fault models injected have been (see two first rows in Table 1): intermittent stuck-at (in memory and registers) and intermittent pulses (in buses). Time activity (t_A) has been generated randomly in three time intervals depending on the system clock cycle (T). Finally, single and multiple (in space domain, i.e. simultaneous faults in different targets) intermittent faults have been injected

Table 2. Results for single intermittent faults injected in memory and registers

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	0.1 %	0.4 %	3.9 %
Latent errors	41.9 %	8.9 %	0 %
Non effective	58.0 %	90.7 %	96.1 %

Table 3. Results for multiple intermittent faults injected in memory and registers

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	1.7 %	15.5 %	53.0 %
Latent errors	43.0 %	6.4 %	0 %
Non effective	55.3 %	78.1 %	47.0 %

Table 4. Results for single intermittent faults injected in buses

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	3.4%	28.2%	55.3%
Latent errors	2.5%	14.1%	16.3%
Non effective	94.1%	57.7%	28.4%

Table 5. Results for multiple intermittent faults injected in buses

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	13.4%	60.8%	89.4%
Latent errors	6.2%	19.8%	7.6%
Non effective	80.4%	19.4%	3%

The main observations that can be made are:

- As expected, the percentage of failures grows with the activity time, in both registers/memory and buses. We can also note that intermittent faults in buses have more impact that those in memory/registers.
- The percentage of latent errors does not show a clear trend with regard to the activity time. Whereas it decreases in registers/memory, it

rises slightly in buses. We think that this discrepancy is due to the workload.

- Multiple faults provoke a greater percentage of failures, which also was foreseeable. The difference is more pronounced in registers/memory than in buses.
- No significant differences in the percentage of latent errors are detected between single and multiple faults.

Finally, Figure 2 compares the impact of transient, intermittent and permanent faults injected in buses. In the case of transient faults, three different values of fault duration have been considered, using the same time intervals as for the activity time in intermittent faults. Some examples of transient and permanent fault models injected can be found in [6].

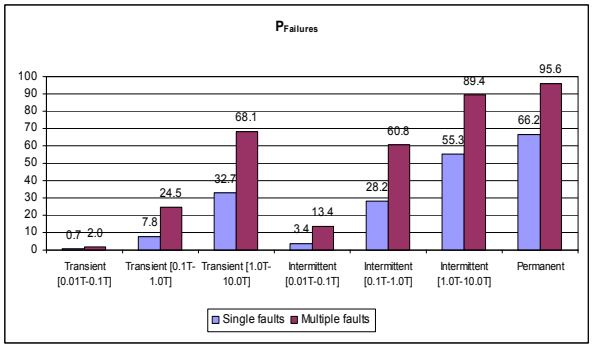


Figure 2. Comparison of the percentage of failures provoked by transient, intermittent and permanent faults in buses

From the figure, we can see that intermittent faults provoke a greater percentage of failures than transient faults. This is a logical result, as a burst of intermittent faults manifest like a sequence of transient faults in spite of having different origin. The effect of fault duration is also shown for all types of faults. Obviously, the greater impact is due to permanent faults because of their indefinite duration.

5. Conclusions and Future work

In this work, we have presented a case study of the effects of intermittent faults on the behaviour of a commercial microcontroller. The methodology used lies in VHDL-based fault injection technique, which allows a systematic and detailed analysis of the influence of different fault parameters. We have also compared the results to those obtained when injecting transient and permanent faults.

In the future, we have planned to extend this study in different aspects. First, we want to use other workloads in order to try to generalize the results. We also intend to analyse the effect of other parameters, like the burst length, the inactivity time, and the system clock frequency. Moreover, we think that it is important to inject other fault models, like delay, intermittent short and open. Finally, it is necessary to investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, in order to propose new fault models related to them.

6. Acknowledgement

This work has been partially funded by the Spanish Government under the project MCYT TEC2005-05119 “Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida”.

7. References

[1] P.J. Gil et al., “Fault Representativeness”, *Deliverable ETIE2 of Dependability Benchmarking Project*, IST-2000-25245, 2002.

[2] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[3] C. Constantinescu, “Impact of Intermittent Faults on Nanocomputing Devices”, in WDSN-07, Edinburgh, UK, June 2007, http://www.laas.fr/WDSN07.

[4] A. Benso and P. Prinetto, eds., “*Fault Injection Techniques and Tools for VLSI reliability evaluation*”, Kluwer Academic Publishers, 2003.

[5] J.C. Smolens et al, “Detecting Emerging Wearout Faults”, 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[6] J.C. Baraza et al, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, *Journal of Systems Architecture*, vol. 47(10):847–867, 2002.

Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers[†]

D.Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto ITACA
Universidad Politécnica de Valencia, Spain
e-mail: {dgil, ljsaiz, jgracia, jcbaraza, pgil}@disca.upv.es

Abstract

It is expected that intermittent faults will be a great challenge in modern VLSI circuits. In this work, we present a case study of the effects of intermittent faults on the behavior of a commercial microcontroller. The methodology used lies in VHDL-based fault injection technique, which allows a systematic and exhaustive analysis of the influence of different fault and system parameters. From the simulation traces, the occurrences of failures and latent errors have been logged. To extend the study, the results obtained have been compared to those got when injecting transient and permanent faults. The applied methodology can be generalized to more complex systems.

1. Introduction

It has been foreseen that intermittent faults will have a great impact in deep submicron technologies. The reduction of the feature size and the power voltage, together with the increase of the clock frequency, will raise the rate of transient faults. On the other hand, the complexity of the manufacturing process (that provokes residues and process variations) and special wearout mechanisms may increase the presence of intermittent faults [1].

Although errors induced by transient and intermittent faults manifest in a similar way, the last ones are activated repeatedly in the same place, and so they are usually grouped in bursts. On the other hand, whereas replacing the affected part eliminates an intermittent fault, transient faults cannot be fixed by repair. Additionally, intermittent faults may be activated or deactivated by temperature, voltage or frequency changes [2].

Transient and permanent faults have well established fault mechanisms and models [3]-[7]. Permanent faults occur due to irreversible physical changes. Transient faults are mainly generated by environmental conditions, like cosmic rays. On the other hand, the knowledge of intermittent faults is not so developed, as fewer observations of real faults and studies of their physical causes and mechanisms have been performed [2][8][9][10].

Related to intermittent faults, there are some questions difficult to answer: Where do intermittent faults happen? When do faults occur? How many times does a fault activate in a burst? How does the fault manifest at higher abstraction levels? etc. To answer these questions, it is important to understand the physical mechanisms that take place in deep submicron technologies. But this research subject is complex, it is technology dependent, and it is still in an early phase of evolution.

In order to study the impact of intermittent faults, we propose to use a methodology based on fault injection as a complement to the research on the *physics of fault.* Fault injection technique allows a controlled introduction of faults in the system, not being necessary to wait for a long time to log the apparition of real faults [11]. Particularly, VHDL-based fault injection can be a very suitable option due to its flexibility as well as the high observability and controllability of all the model components [12]. It allows both to make an exhaustive variation of the fault parameters and to analyze the effects of faults on the system behavior.

In previous works [13][14], we have generated fault models for intermittent faults at logic and RTL abstraction levels, and injected these faults in the VHDL model of a commercial microcontroller to study their impact on the system behavior. In these works, the influence of different fault and system parameters has been analyzed.

The objective of this work is to deepen those studies, analyzing the importance of other factors. Faults have been injected in new locations, and both

[†] This work has been partially funded by the Spanish Government under the project MCYT TEC2005-05119 "Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida".

the burst length and the operation frequency have been modified.

This work is organized as follows. Section 2 depicts a set of intermittent fault models that can be injected. Section 3 describes the fault injection experiments, and Section 4 discusses the results obtained. Finally, Sections 5 and 6 provide some conclusions and a proposal of future work.

2. Fault Models

As stated above, whereas the models of transient and permanent faults have been traditionally well established, intermittent fault modeling is a pending issue. To obtain representative fault models for intermittent faults, it is necessary to understand the physical mechanisms that take place in deep submicron technologies. Intermittent faults occur due to unstable or marginal hardware. Manufacturing residues, process variations and special wearout processes can lead to such faults. In addition, wearout mechanisms may provoke that intermittent faults eventually end up in permanent faults.

We have selected a set of intermittent faults observed in real computer systems by means of fault logging [9][2], as well as fault mechanisms related to process variations and wearout [2][10][15][16]. Then, we have deduced a set of fault models at logic (gate) and RTL abstraction levels which can be simulated into VHDL models [13][14]. Table 1 shows some examples of intermittent faults and the associated fault models.

Fault models applied in this work have been emphasized in bold. Bursts of Single Bit Errors (or SBEs) have been observed in the memory of a set of monitored servers [9]. Analysing the failures, it was found that polymer residues led to an intermittent contact. The fault model chosen to represent a SBE in memory and register cells is *intermittent stuck-at*, which represents that the value of the cell changes intermittently between '0' and '1'. The origin of the fault lies in an unstable hardware, and it is not related to an external environmental effect (e.g. cosmic radiation). So, it is not an *intermittent bit-flip*, and we have called this fault model *intermittent stuck-at*.

Bursts of SBEs have also been observed in data buses [9]. The analysis of the failures revealed that the source of errors were solder intermittent contacts. We have associated this mechanism to the following fault models: *intermittent pulse*, *intermittent short* and *intermittent open*, appliable to bus lines [4].

3. Fault injection experiments

The different fault injection experiments were carried out on the VHDL model of the 8051 microcontroller [17] running the Bubblesort sorting algorithm as workload. Although the chosen system has not a deep submicron technology, the methodology used can be generalized to more complex microprocessors/microcontrollers.

Fault injection experiments were performed using a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [18], that runs on PC computers (or compatible) under Windows®. VFIT can apply different fault injection techniques on VHDL models (see Figure 1) at diverse abstraction levels. In this work we used the technique based on *simulator commands*, because it is not necessary to modify the VHDL code and it is easy to apply. This technique consists on using the commands of the simulator to modify the value of the model signals and variables at simulation time. Nevertheless, it has limitations to inject some complex fault models [12].

Next we will set the main injection parameters. Some of them are closely connected with the questions posed in Section I.

Table 1. Some intermittent faults mechanisms and models [14]

Causes	Places	Fault mechanisms	Type of fault	Fault Models
Residues in cells	**Memory and registers**	Intermittent contacts	Manufacturing defect	**Intermittent stuck-at**
Solder joints	**Buses**	Intermittent contacts	Manufacturing defect	**Intermittent pulse**, intermittent short, intermittent open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance	Wearout-timing	Delay, intermittent short, intermittent open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing Voltage perturbation	Delay,intermittent pulse
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wearout-timing	Delay, intermittent indetermination

In bold, fault models injected in this work

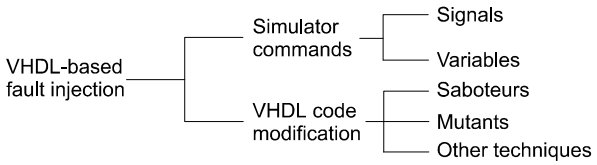


Figure 1. Fault-injection techniques for VHDL models [12]

Where are faults injected? Intermittent faults were injected in two types of places (targets) of the microcontroller: memory elements (the register file and the internal RAM), and the buses. We chose these targets because in real systems, intermittent faults have been logged in similar locations, as it was indicated in Section II. Figure 2 shows the structure and the injection targets of the microcontroller. There are other locations potentially sensitive to intermittent faults, such as the combinational logic (arithmetic and control circuits), that will be analyzed in a future work.

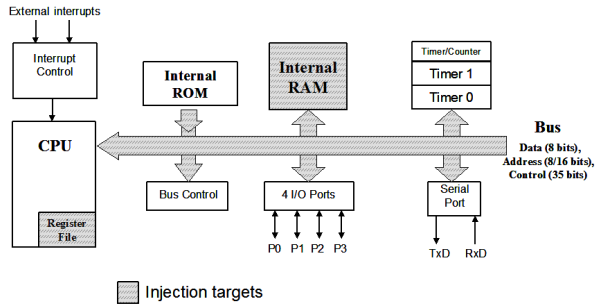


Figure 2. Injection targets

Depending on the number of places perturbed in each injection, faults can be classified as *single* (one cell or bus line) or *multiple* (various cells or bus lines). As stated in Section I, it is expected that there will be a rise of the manufacturing defects in deep submicron technologies. This may imply the presence of multiple faults in the system, and so it is interesting to take into account this issue in the injection experiments.

What type of faults are injected? The fault models to inject depend strongly on the injection place. According to Table 1, and considering the capabilities of the injection technique chosen, the intermittent fault models selected are: 1) *Intermittent stuck-at*, for memory and registers, and 2) *Intermittent pulse*, for buses. These fault models are emphasized in bold in Table 1.

When are the faults injected? The fault injection instant has been selected randomly along the workload duration, according to a Uniform distribution. In real systems, other fault distributions have been observed, such as Exponential or Weibull [8]. For instance, a Weibull distribution with increasing fault rate can be used to emulate a wearout process.

Which are the parameters of the burst? As stated above, intermittent faults manifest in bursts. So, to inject this type of faults the following parameters must be configured (see Figure 3):

- The number of activations of the fault in the burst (we will call it *burst length*, or L_{Burst}).
- The duration of each activation (we will refer to it as *activity time*, or t_A).
- The separation between two consecutive activations (we will name it as *inactivity time*, or t_I).

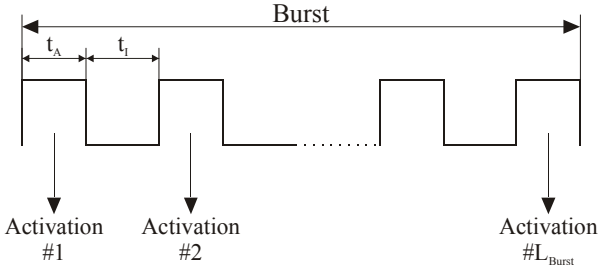


Figure 3. Main elements of a burst

The burst length (L_{Burst}) was varied randomly between 1 and 10. Activity time (t_A) and inactivity time (t_I) were generated randomly according to a Uniform distribution function in three time intervals, depending on the system clock cycle (T): [0.01T – 0.1T], [0.1T – 1.0T], and [1.0T – 10.0T].

Nevertheless, in wearout processes it is expected that during lifetime, L_{Burst} and t_A will increase and t_I will decrease. At last, the intermittent fault may become a permanent fault. Hence, we will use in the future other distribution functions, such as Weibull.

In order to measure the impact of intermittent faults, we have calculated in all cases the percentages of provoked *failures* and *latent errors*:

- Percentage of failures:

$$P_{Failures} = \frac{N_{Failures}}{N_{Injected}} \times 100$$

- Percentage of latent errors:

where:

- $N_{Injected}$ = number of faults injected (1000 per injection experiment).
- $N_{Failures}$ = Number of failures, defined as the number of propagated errors (that is, the injected faults which propagate to the memory and registers) that provoke failures. A failure is produced when the result is erroneous at the end of the simulation time. Similar definitions of failure can be found in [8] [19].
- N_{Latent} = Number of latent errors. A latent error is a propagated error that has not provoked a failure.

Figure 4 summarizes the syndrome of faults and the calculated data.

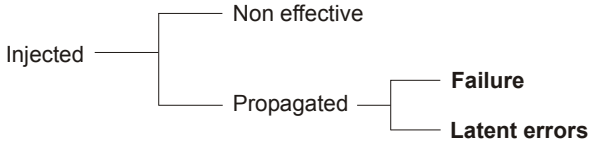


Figure 4. Syndrome of faults

4. Results

According to the injection parameters whose incidence is analyzed, we have divided the injection experiments in two groups. First, we show the study of the influence of the burst parameters. Next, we present the study of the influence of other parameters, such as the injection place and the system clock frequency.

In both studies, the importance of the fault multiplicity has been considered.

4.1. Influence of burst parameters

As explained in Section 3, the main parameters of an intermittent fault burst are the number of activations (L_{Burst}), the duration of the activations (t_A), and the separation between activations (t_I).

4.1.1. Activity time. Figures 5 and 7 show that the percentage of failures grows with the activity time (t_A in Figure 3), both in memory/registers and buses. Depending on the target and the fault multiplicity, the growth seems to fit logarithmic or linear functions (note that the t_A scale is logarithmic).

We can also note that intermittent faults have more impact on buses than on memory/registers, showing that buses are more sensitive to intermittent faults.

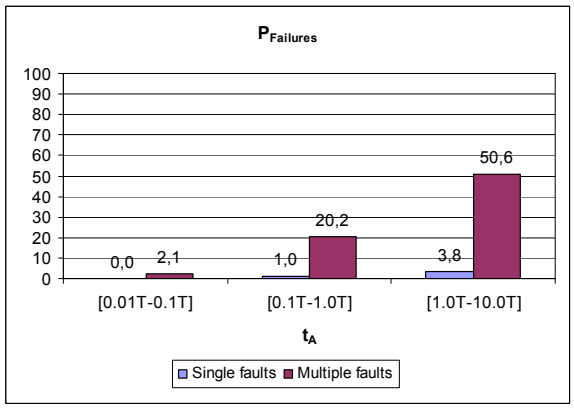


Figure 5. Influence of t_A in $P_{Failures}$ (Target: memory and registers)

Figures 6 and 8 reflect that the percentage of latent errors does not show a clear trend with regard to the activity time. Whereas it decreases in memory/registers, it rises slightly in buses. This discrepancy can be due to the workload.

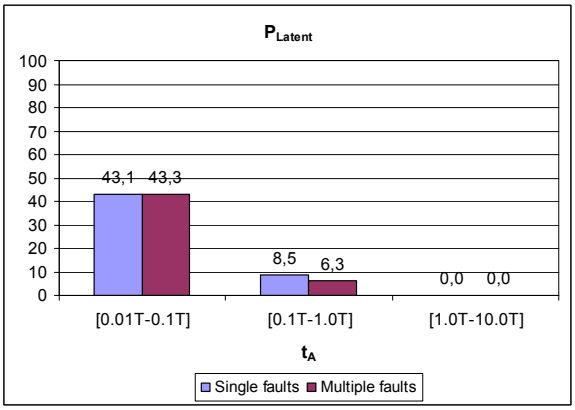


Figure 6. Influence of t_A in P_{Latent} (Target: memory and registers)

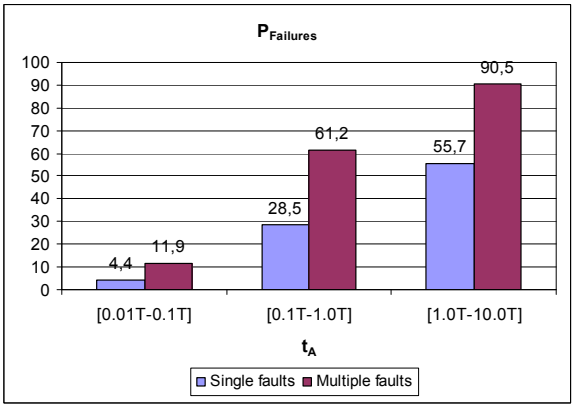


Figure 7. Influence of t_A in $P_{Failures}$ (Target: buses)

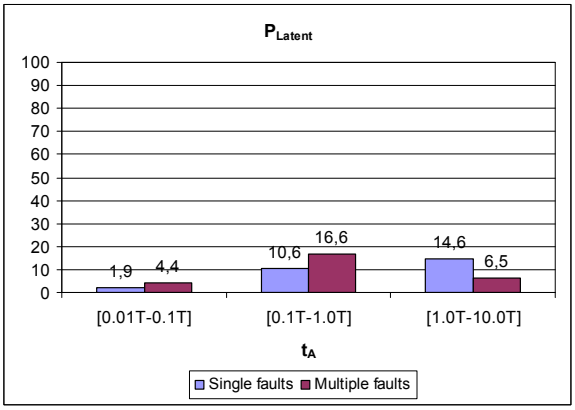


Figure 8. Influence of t_A in P_{Latent} (Target: buses)

Regarding the spatial multiplicity, from Figures 5 to 8, it can be seen that multiple faults provoke a greater percentage of failures than single faults. This is a predictable behaviour, because multiple faults affect simultaneously various physical locations of the system. Comparing the $P_{Failures(multiple)}/P_{Failures(single)}$ ratio,

values of up to 20 have been observed. It is noticeable that higher values of this ratio have been detected for lower values of t_A, and specially in registers/memory targets.

Finally, no significant differences in the percentage of latent errors are detected between single and multiple faults.

4.1.2. Separation between activations. Figures 9 and 10 show the influence of the temporal separation between the pulses of the burst (inactivity time or t_I in Figure 3) for intermittent faults injected in registers and memory. No significant differences are observed when varying the inactivity time. The separation of the activations of the burst does not provoke a noteworthy variation in $P_{Failures}$ or P_{Latent} (for a fixed value of L_{Burst}). A similar behavior has been observed when injecting in buses.

4.1.3. Burst length. The number of activations of the burst (L_{Burst} in Figure 3) has been varied from 1 to 10. In wearout processes, the trend is expected to be a raise of the length of the burst during the life time, until the fault becomes a permanent fault.

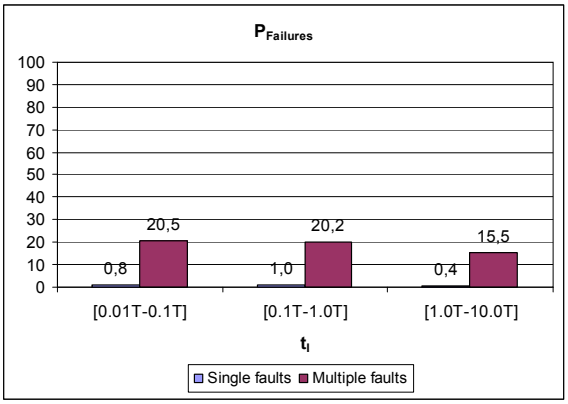


Figure 9. Influence of t_I in $P_{Failures}$ (Target: memory and registers)

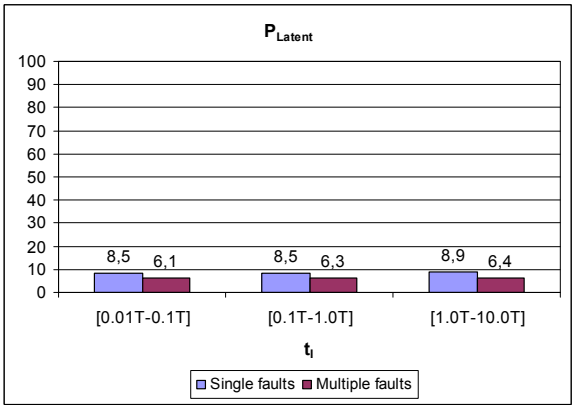


Figure 10. Influence of t_I in P_{Latent} (Target: memory and registers)

Figure 11, corresponding to faults injected in memory and registers, shows that $P_{Failures}$ rises asymptotically up to 27.5%. From $L_{Burst} \approx 9$ $P_{Failures}$ gets stable. On the other hand, P_{Latent} has small variations with the length of the burst, with values between 5% and 8%.

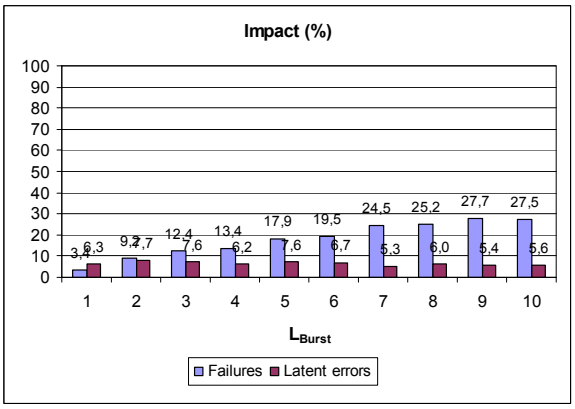


Figure 11. Influence of the L_{Burst} in $P_{Failures}$ and P_{Latent} (Target: memory and registers)

Figure 12, corresponding to faults injected in buses, shows the same trend that Figure 11, although the values of $P_{Failures}$ and P_{Latent} are notably higher. $P_{Failures}$ rises asymptotically up to 75%, and stabilizes from $L_{Burst} \approx 9$. Instead, P_{Latent} shows fewer variations, varying between 13% and 21%.

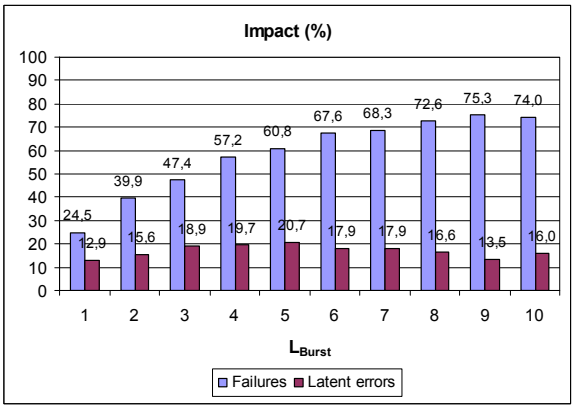


Figure 12. Influence of L_{Burst} in $P_{Failures}$ and P_{Latent} (Target: buses)

4.2. Influence of other parameters

4.2.1. Injection place. In previous results we have observed that intermittent faults in buses are more harmful than in memory/registers. One reason can be the difference in the area occupied by the two targets. The size (in bits) of the registers plus memory is quite bigger than that of the buses. This implies that the probability of sensitizing the workload is lower. In

addition, buses are a bottle-neck in the fetch and execution phases.

This suggests the need of adding mitigation techniques to detect and tolerate intermittent faults in buses. For instance, ECC implemented in hardware can deliver fast error detection and correction of errors occurring at a high rate. In general, hardware implemented error handling techniques are likely to provide the best solutions to diminish the effects of intermittent faults [2].

4.2.2. Clock frequency of the system. Figures 13 and 14 show $P_{Failures}$ and P_{Latent} respect to the frequency, for single and multiple intermittent faults in buses. In both cases, $P_{Failures}$ rises asymptotically, similarly to as with the burst length.

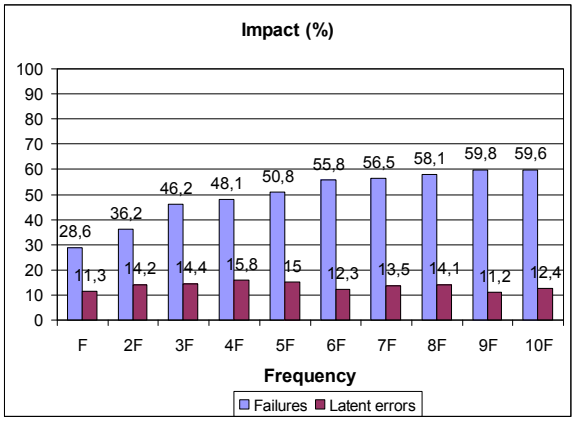


Figure 13. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: buses. Single faults)

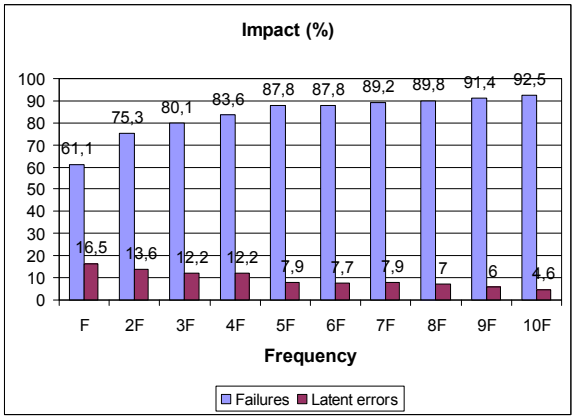


Figure 14. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: buses. Multiple faults)

When injecting single faults, $P_{Failures}$ tends to 60%, while with multiple faults it stabilizes at about 90% from 7×F (F = 10 MHz). The explanation of the strong frequency influence lies in the fact that at higher frequencies, the probability of capturing an error in active edges of synchronous components (like memory and registers) rises. It is important to remark that the frequency increase has been a trend in VLSI technologies.

On the other hand, P_{Latent} does not reflect a clear dependency. In Figure 14 it seems to show a smooth negative exponential function, whereas in Figure 13 we do not observe a defined trend. Until 4×F, P_{Latent} raises to 16%. Since then, it holds around 11%-15%.

When injecting in memory and registers, we have observed a similar asymptotic trend of $P_{Failures}$, as Figures 15 and 16 show. Nevertheless, the values are notably lower than in buses case. About P_{Latent} values, a sharp negative exponential behaviour is observed.

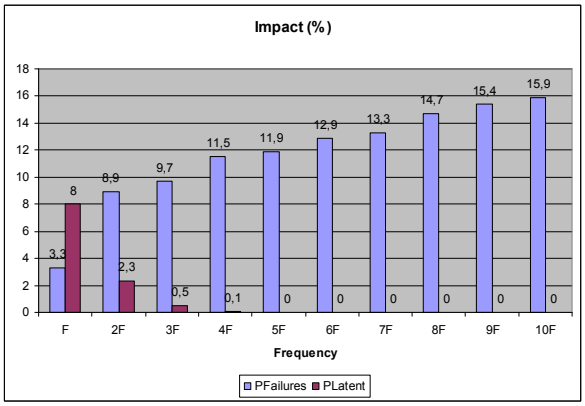


Figure 15. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: memory and registers. Single faults)

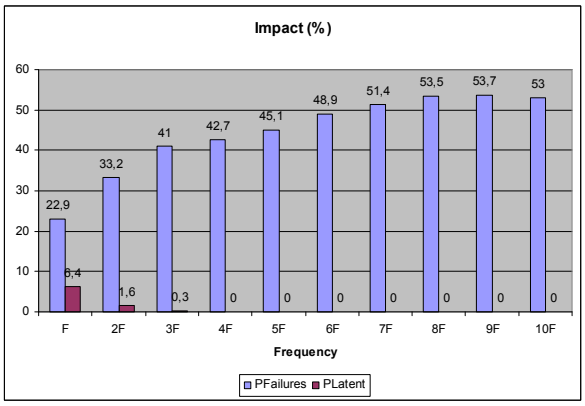


Figure 16. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: memory and registers. Multiple faults)

4.3. Comparison with transient and permanent faults

Finally, Figures 17 and 18 compare the impact of transient, intermittent and permanent faults for the two types of targets. Transient fault models injected have been *bit-flips* in memory/registers (to emulate Single Event Upsets, or SEUs) and *pulses* in buses (to emulate

Single Event Transients, or SETs). For permanent faults, the fault models injected have been *stuck-at(0,1)*, *open*, and *indetermination* [12][18]. In the case of transient faults in buses, three different values of fault duration have been considered, using the same time intervals as for the activity/inactivity times in intermittent faults. Due to their nature, when injecting bit-flips, it does not make any sense to set any duration.

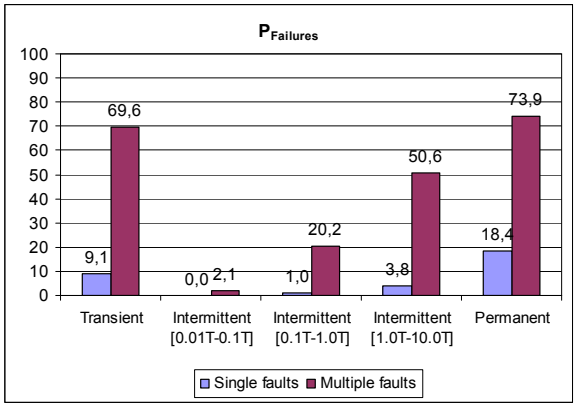


Figure 17. Influence of the fault type in $P_{Failures}$ (Target: memory and registers)

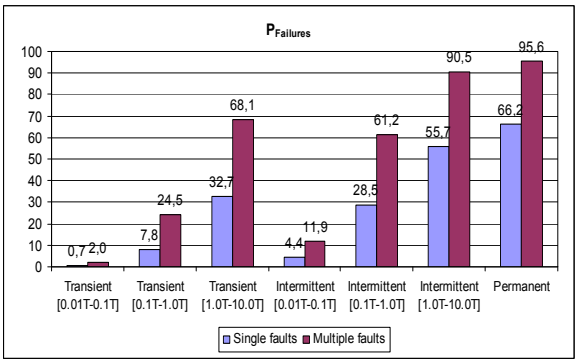


Figure 18. Influence of the fault type in $P_{Failures}$ (Target: buses)

In Figure 18 (buses) we can see that intermittent faults provoke a greater percentage of failures than transient faults. This is a logical result, as a burst of intermittent faults manifest like a sequence of transient faults in spite of having different origin. Calculating the $P_{Failures(intermittent)}/P_{Failures(transient)}$ ratio, values between 1.5 and 6 are observed. Values are higher for single faults.

However, Figure 17 (memory/registers) shows an unexpected trend. Intermittent faults have a lower impact than transient faults, even for the biggest activity time. A more detailed analysis of the injected fault models and the workload allowed establishing the cause of this anomalous behavior. The characteristics of the workload used provoke a low rate of overwrite operations in the memory cells affected by transient faults (bit-flips), causing a *de facto* infinite duration of these faults, thus remaining stored permanently.

Obviously, in both figures, the greatest impact corresponds to permanent faults because of their infinite duration, independently of the type of target.

5. Conclusions

In this work, we have presented a case study of the effects of intermittent faults on the behavior of a commercial microcontroller (8051). The methodology used lies in VHDL-based fault injection technique, which allows a systematic and exhaustive analysis of the influence of different fault parameters and factors. We have also compared the results to those obtained when injecting transient and permanent faults. This methodology can be applied to other microcontrollers and/or workloads.

In general terms, and assuming the important influence of the microcontroller and the workload used, some results can be summarized.

About the influence of burst parameters:

- The activity time is a quite critical parameter. Increasing the duration of the activations provokes the rise of the percentage of failures. Linear or logarithmic growths have been observed, depending on the target and/or the fault multiplicity.
- The burst length has also a notable impact. Nevertheless, when increasing the burst length, the percentage of failures grows towards a horizontal asymptote. From a given value of burst length, the percentage of failures stabilizes and remains constant.
- Respect to the inactivity time, no important changes have been observed. It seems that the separation between activations does not modify the percentage of failures (assuming a fixed number of activations).

During lifetime, it is expected that the trend in wearout fault mechanisms will be a raise of both the activity time and the burst length, and a decrease of the separation between activations. This must be taken into account in future fault injection experiments in order to characterize the burst in a more realistic way.

About other factors studied:

- The *spatial multiplicity* of intermittent faults presents a significant impact on the behavior of the system. Multiple faults provoke a greater percentage of failures than single faults. The complexity of the manufacturing process in deep submicron technologies may favor the presence of multiple intermittent faults.

- The injection *target (place)* leads also to important differences. We have observed that intermittent faults in buses are more harmful than in memory/registers. This fact suggests the suitability of adding mitigation techniques to deliver fast error detection and correction of intermittent errors in buses. For instance, applying ECC implemented in hardware.
- The rise of the *clock frequency* increases the impact of intermittent faults. This is due to the higher probability of capturing an error in active edges of synchronous components. The percentage of failures presents an asymptotical rising trend, similar to the burst length dependency. The increase of the clock frequency has been a trend in deep submicron technologies.

It is necessary to point out that the percentage of latent errors (errors that propagate to the memory and registers, but do not provoke failures) has not shown a definite trend with regard to most parameters. Factors related with the workload execution may be involved in their behavior and need to be more deeply studied.

Finally, the impact of the intermittent faults has been compared to that of transient and permanent faults. As a burst of intermittent faults manifest like a sequence of transient faults (in spite of having a different origin), intermittent faults provoke a greater percentage of failures than transient faults. Obviously, the greatest impact is caused by permanent faults because of their infinite duration.

6. Future work

In the future, we have planned to extend this study in different aspects. First, we intend to use other workloads in order to try to generalize the results. We also want to analyze the effect of intermittent faults on other microprocessors or microcontrollers. Moreover, we think that it is important to inject other representative fault models, like delay, intermittent short and intermittent open. It is also necessary to investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, in order to propose new fault models related to them. Particularly, it would be interesting to identify intermittent fault models for combinational circuits. Finally, we want to study mitigation techniques for fast detection and recovery of intermittent faults.

7. References

[1]. C. Constantinescu, "Impact of Deep Submicron Technology on Dependability of VLSI Circuits", in Procs. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[2]. C. Constantinescu, "Impact of Intermittent Faults on Nanocomputing Devices", in Procs. WDSN-07, Edinburgh, UK, June 2007, http://www.laas.fr/WDSN07.

[3]. E.A. Amerasekera and F.N. Najm, "Failure mechanisms in semiconductor devices", John Whiley & Sons, 1997.

[4]. P.J. Gil *et al.*, "Fault Representativeness", *Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245*, 2002.

[5]. J. Segura, A. Keshavarzi, C.F. Hawkins, "CMOS IC nanometer technology failure mechanisms", IEEE Custom Integrated Circuits Conference, 2003.

[6]. J. Segura, C.F. Hawkins, "CMOS Electronics. How it works, how it fails". 2004.

[7]. N. Weste, D. Harris, "CMOS VLSI design", Addison-Wesley, 2005.

[8]. D.P. Siewiorek, R. S. Schwarz: *Reliable computer systems: Design and evaluation* (2nd. Edition). Ed. Digital Press, 1992.

[9]. C. Constantinescu, "Dependability Benchmarking using Environmental Test Tools". Procs. Reliability and Maintainability Symposium (RAMS'05), pp. 567–571, January 2005.

[10]. J.C. Smolens, B.T. Gold, J.C. Hoe, B. Falsafi, and K. Mai. "Detecting Emerging Wearout Faults", 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[11]. J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", IEEE Transactions on Software Engineering, vol. 16(2):166–182, 1990.

[12]. D. Gil, J. C. Baraza, J. Gracia, and P. J. Gil, "VHDL simulation-based fault injection techniques", Ch. 4.1 in *Fault Injection Techniques and Tools for VLSI Reliability Evaluation*, A. Benso and P. Prinetto, Eds. Dordrecht, The Nederlands: Kluwer Academic Publishers, 2003, pp. 159–176.

[13.]. J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil, "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), pp. 80-85, Bratislava, SlovaKia, April 2008

[14]. L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Applying Fault Injection to Study the Effects of Intermittent Faults", 7th European Dependable Computing Conference (EDCC-7), Supplemental Volume, pp.67-69, Kaunas, Lithuania, May 2008.

[15]. S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitecture", in Procs. 40th Conference on Design Automation (DAC'03), pp. 338 – 342, April 2003.

[16]. J.W. McPherson, "Reliability challenges for 45nm and beyond", Procs. Conference on Design Automation (DAC'06), pp. 176–181, July 2006.

[17]. http://www.oregano.at

[18]. J.C. Baraza et al, "A Prototype of a VHDL-Based Fault Injection Tool: Description and Application", *Journal of Systems Architecture*, vol. 47(10):847–867, 2002.

[19]. J.C. Laprie, "Dependability: basic concepts and terminology", *Sringer-Verlag*, 1992.

Intermittent Faults: Analysis of Causes and Effects, New Fault Models, and Mitigation Techniques

Luis-J. Saiz-Adalid
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas – Universidad Politécnica de Valencia
Camino de Vera, s/n, 46022, Valencia, Spain
ljsaiz@disca.upv.es

Abstract

It is expected that dealing with intermittent faults will be a great challenge in modern VLSI circuits. As new submicron technologies have increased processors performance and reduced their size, their reliability has decreased. The objective of my research work is to study intermittent faults, to analyze their causes and effects and to propose new fault models and mitigation techniques. In order to check these new proposals, it is planned to develop the VHDL model of a fault tolerant system including different mitigation techniques. These techniques will be validated by injecting the intermittent fault models proposed.

1. Introduction

The advances in integration techniques have allowed rising microprocessors operating frequency, as well as reducing their size and power voltage, achieving a greater performance. Nevertheless, these advances have had a negative impact on their reliability, and the rate of occurrence of faults has increased [1]. Permanent and transient faults have been extensively studied, but the analysis of the causes and mechanisms of intermittent faults is a pending issue and an open line of research.

It is foreseen that intermittent faults will have a great impact in deep submicron technologies [2]. Usually intermittent faults have been assumed to be the prelude of permanent faults. Currently, intermittent faults due to process variations and manufacturing defects have grown.

Errors induced by transient and intermittent faults manifest in a similar way, but the last ones are activated repeatedly in the same place, and they are usually grouped in bursts. On the other hand, replacing the affected part eliminates an intermittent fault, while transient faults cannot be fixed by repair. Additionally, intermittent faults may be deactivated or activated by temperature, voltage and frequency changes [3].

Related to intermittent faults, there are some questions difficult to answer: Where do intermittent faults happen? When do faults occur? How many times does a fault activate in a burst? How does the fault manifest at higher abstraction levels? … To answer these questions, it is important to understand the physical mechanisms that take place in deep submicron technologies. But this research subject is complex, it is technology dependent, and it is still in an early phase of evolution.

In order to study the impact of intermittent faults, a methodology based on fault simulation is proposed. Fault injection allows a controlled introduction of faults in the system, not being necessary to wait for a long time to log the apparition of real faults [4]. Particularly, VHDL-based fault injection can be a very suitable option due to its flexibility and high observability and controllability of all the model components [5]. It allows to carry out an exhaustive variation of the fault parameters as well as to analyze the effects of faults on the system behavior.

My research work is enclosed in the GSTF (Fault-Tolerant Systems) research group. The group has a wide experience in fault injection techniques and dependability research. This paper introduces the research line for my PhD thesis.

Following this research line, first results and conclusions obtained by the group studying the effects of intermittent faults can be found in [6].

This work is organized as follows. Section 2 describes the intermittent fault models studied. Section 3 introduces the fault injection environment. Section 4 presents a brief analysis of mitigation techniques. Finally, Section 5 provides proposals for future work.

2. Intermittent Fault Models

Hardware fault representativeness for transient and permanent faults has been extensively studied and they are well established [7]. As far as I know, the causes and mechanisms of intermittent faults in deep submicron technologies have not been studied so much. To obtain representative fault models for intermittent faults, it is necessary to understand their physical mechanisms.

The first option is to use fault models similar to those used previously for permanent faults, changing their temporal behavior. In fact, in a wearout process faults initially may appear intermittently, but eventually end up in permanent faults [3].

However, the introduction of new submicron technologies makes necessary to keep in mind the influence of manufacturing defects. Process variations, residues and special wearout processes can lead to new intermittent faults [8]. A deeper study is necessary, being this an open line of research.

To define a set of intermittent fault models, the effects of intermittent faults observed in real computer systems by means of fault logging [2], as well as fault mechanisms related to process variations and wearout [2][8] have been extrapolated. In previous works, some intermittent fault models at logic (gate) and RT abstraction levels have already been deduced [9].

Bursts of Single Bit Errors (or SBEs) have been observed in memory [2]. The analysis of failures found that polymer residues can lead to an intermittent contact. The origin of the fault lies in an unstable hardware, and it is not related to an external environmental effect. The value of a bit storage cell changes intermittently to a fixed value. It is not a repeated bit-flip. Instead, it is similar to a stuck-at, but with intermittent temporal behavior. The fault model deduced to represent bursts of SBEs in storage elements (registers and memory) has been called *intermittent stuck-at*.

Bursts of SBEs have also been observed in data buses [10]. Failure analysis revealed that the source of errors was intermittent solder contacts. It is expected that control lines would have the same problem, although this kind of errors are more difficult to log. We have associated this mechanism to *intermittent pulse*, *intermittent short* and *intermittent open* fault models in bus and control lines.

Also, timing failures may occur due to propagation delays over the interconnection lines. For instance, barrier layer delamination and electromigration lead to a higher resistance and, as a result, to timing violations. Electromigration can also induce voids [3] that can provoke short and open faults in metal lines. Process, voltage and temperature (PVT) variations tend to amplify crosstalk delays. They may take place when adjacent signals switch in opposite directions, and are related to capacitive effects in adjacent metallic lines (Miller effect). Even more, crosstalk speed-ups may occur when adjacent signals switch in the same direction [11], provoking that signals switch faster than expected. The fault models proposed to these mechanisms are *intermittent delay*, *intermittent speed-up*, *intermittent short* and *intermittent open* in interconnection lines and buses.

Another type of causes and mechanisms of intermittent faults are due to oxide defects and wearout. Soft breakdown belongs to this class. In this case, the leakage current of the gate oxide fluctuates in time, without inducing the thermal damage, so it does not produce a permanent breakdown [12]. This provokes erratic fluctuations of the minimum supply voltage. Sensitivity to this parameter is expected to increase with scaling. This fluctuation may lead to voltage values not associated to a logic value and can be modeled with the *intermittent indetermination* fault model.

Table 1 summarizes the intermittent fault mechanisms and models described.

In the future, it is planned to carry out a deeper study of the fault mechanisms related with wearout processes and process variations in new submicron technologies.

Table 1. Some intermittent faults mechanisms and models

Causes	Targets	Fault mechanisms	Type of fault	Fault Models
Residuals in cells	Memory and registers	Intermittent contacts	Manufacturing defect	Intermittent stuck-at
Solder joints	Buses	Intermittent contacts	Manufacturing defect	Intermittent pulse, intermittent short, intermittent open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance and/or capacity Voids	Wearout/Timing	Intermittent delay, intermittent speed up, intermittent short, intermittent open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing Voltage perturbation	Intermittent delay, intermittent pulse
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wearout/Timing	Intermittent delay, intermittent indetermination

3. Fault injection environment

The GSTF have developed VFIT (VHDL-based Fault Injection Tool) [13], an automatic fault injection tool to inject faults in VHDL models. VFIT is model-independent and admits VHDL models at any abstraction level, although it has been mainly used on models at logic and RT levels. VFIT has been used for years in different research projects, and hundreds of fault injection experiments have been carried out [7][14][15].

VFIT can inject permanent, transient and intermittent faults, in both single and multiple locations. It allows to inject a wide set of fault models that try to be representative of deep submicron technologies.

The output of an injection experiment depends on the objective: it can perform an error syndrome analysis or a dependability validation campaign. In the first case, it calculates propagation latencies and percentages of propagated errors and failures. In the second one, detection and recovery coverages and latencies are obtained.

As mentioned before, intermittent faults manifest in bursts. That is, it activates and deactivates repeatedly several times in the same place. So, to inject intermittent faults, the following parameters must be configured (see Figure 1): the number of activations in the burst (burst length, or L_{Burst}), the duration of each activation (activity time, or t_A), and the separation between two consecutive activations (inactivity time, or t_I). In VFIT, the values for these parameters can be selected randomly, according to a selectable distribution function. In a burst, each activation or separation times may have different durations.

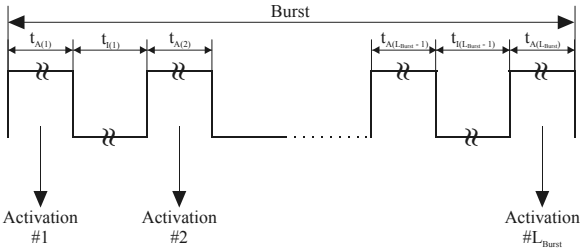


Figure 1. Parameters of a burst

Several intermittent fault injection experiments have been carried out into the VHDL model of the 8051 microcontroller [16] running different workloads: the Arithmetic Series, the Bubblesort algorithm and a matrix multiplication.

The objective of these experiments is to study the influence of different parameters of intermittent faults in the system. Due to lack of space, results are omitted here. Anyway, a detailed explanation of some of these experiments, results and conclusions can be found in [6][9][17].

4. Mitigation techniques

One objective of my research work is to study different mitigation techniques, trying to make them suitable for intermittent faults, or even finding a novel approach. Finally, in order to test the solutions proposed, the idea is to develop the VHDL model of a microprocessor or microcontroller system including them.

In this way, the most vulnerable hardware structures that may need protection should be identified. As stated previously, errors can be found in both storage elements and buses. For this reason, the information coded here must be protected. The first approach could be to use error detecting and correcting codes such as low-density parity check codes [18][19], linear codes (Hamming, Reed-Solomon, convolutional, …) [20], etc. combined with other techniques like scrubbing [21].

Elements like combinational circuits can be protected with different techniques: TMR, duplication and comparison, time redundancy, reconfiguration, etc.

The key question to tolerate intermittent faults is to detect faulty hardware, and either mask it or reconfigure the circuit. Possible techniques are:

- Detection: encoding, duplication+comparison, self-checking, etc.
- Masking: TMR, instruction-retry, etc.
- Reconfiguration when the fault/error is near a permanent fault/error.

Finally, other techniques like redundant execution, watchdog processors, software detection and recovery, etc. can be used.

5. Future work

There is a lot of pending work. Firstly, it is important a better understanding of intermittent fault causes and mechanisms, in order to get representative fault models. Secondly, and using these fault models, it will be necessary to carry out new fault injection experiments into the VHDL model of various microprocessors or microcontrollers running different workloads. Thus, it will be possible to deepen in the study of the influence of the parameters of intermittent faults. Thirdly, a complete study of mitigation techniques is needed. The objective is to get a set of techniques suitable for intermittent fault tolerance. Finally, the VHDL model has to be modified to

include the selected mitigation techniques, and the fault injection experiments should be repeated into the hardened model to check the validity of the proposed scheme.

As a conclusion, the main objective of my PhD research is to study intermittent faults: analyze their causes and mechanisms, deduce representative fault models, examine their effects using fault simulation and find solutions to mitigate them.

6. References

[1] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability", *IEEE Micro*, Vol. 23(4):14-19, 2003.

[2] C. Constantinescu, "Impact of Deep Submicron Technology on Dependability of VLSI Circuits", in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[3] C. Constantinescu, "Impact of Intermittent Faults on Nanocomputing Devices", in WDSN-07, Edinburgh, UK, June 2007. http://www.laas.fr/WDSN07.

[4] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, vol. 16(2):166–182, 1990.

[5] D. Gil, J. C. Baraza, J. Gracia, and P. J. Gil, "VHDL simulation-based fault injection techniques", Ch. 4.1 in *Fault Injection Techniques and Tools for VLSI Reliability Evaluation*, A. Benso and P. Prinetto, Eds. Dordrecht, The Nederlands: Kluwer Academic Publishers, 2003, pp. 159–176.

[6] D. Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil, "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", HLDVT'08, Incline Village, NV, USA, November, 2008, pp. 177-184.

[7] P.J. Gil et al., "Fault Representativeness", Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245, 2002.

[8] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitecture", in Procs. 40th Conference on Design Automation (DAC 2003), pp. 338 – 342, April 2003.

[9] J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil, "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), pp. 80-85, Bratislava, Slovakia, April 2008

[10] C. Constantinescu, "Dependability Benchmarking using Environmental Test Tools". Procs. Reliability and Maintainability Symposium (RAMS'05), pp. 567–571, January 2005.

[11] W. Moore, G. Gronthoud, K. Baker, M. Lousberg, "Delay-fault testing and defects in deep sub-micron ICs-does critical resistance really mean anything?", Procs. IEEE International Test Conference 2000, Atlantic City, NJ, USA, October 2000, pp. 95-104.

[12] J.H. Stathis, "Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits", IEEE Trans. on Device and Materials Reliability, Vol. 1(1):43–59, March 2001.

[13] J.C. Baraza, J. Gracia, D. Gil, and P.J. Gil, "A Prototype of a VHDL-Based Fault Injection Tool: Description and Application", Journal of Systems Architecture, vol. 47(10):847–867, 2002.

[14] J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Using VHDL-Based Fault Injection to Exercise Error Detection Mechanisms in the Time-Triggered Architecture", Procs. of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC2002), pp. 316-320, Tsukuba, Japan, December 2002.

[15] D. Gil, J. Gracia, J.C. Baraza, P.J. Gil, "Impact of Faults in Combinational Logic of Commercial Microcontrollers", Lecture Notes in Computer Science, Dependable Computing – EDCC-5, Springer-Verlag GMBH, Vol.3463/2005, pp. 379-390 April 2005.

[16] http://www.oregano.at

[17] L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Applying Fault Injection to Study the Effects of Intermittent Faults", 7th European Dependable Computing Conference (EDCC-7), Supplemental Volume, pp.67-69, Kaunas, Lithuania, May 2008.

[18] R.G. Gallager, *Low Density Parity Check Codes* Cambridge, MA: M.I.T. Press, 1963.

[19] S.K. Chilappagari, B.V. Vasic, "Fault Tolerant Memories Based on Expander Graphs", ITW'07, IEEE, Tahoe City, CA, USA, September 2007, pp. 126-131.

[20] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[21] A.M. Saleh, J.J. Serrano, J.H. Patel, "Reliability of Scrubbing Recovery-Techniques for Memory Systems", IEEE Transactions on Reliability, Vol. 39, Nº 1, pp. 114-122, April 1990.

A Proposal of a Fault-Tolerant Mechanism for Microprocessor Buses

Luis-J. Saiz-Adalid, J.-Carlos Baraza-Calvo, Joaquín Gracia-Morán, Daniel Gil-Tomás
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas – Universidad Politécnica de Valencia
Camino de Vera, s/n, 46022, Valencia, Spain
{ljsaiz, jcbaraza, jgracia, dgil}@disca.upv.es

***Abstract*—New submicron technologies have allowed increasing processors performance reducing their size, but their reliability has decreased. Faults due to manufacturing residuals and process variations have grown. Transient and intermittent faults are expected to be a big challenge in modern VLSI circuits, in addition to permanent faults. Different studies have revealed that interconnection lines and storage cells are very sensitive to these kinds of faults.**

In this paper we present a proposal of implementation of fault-tolerant coding methods to apply to system buses, lying in the use of Hamming codes. Its features and fault tolerance capabilities are also revised.

Keywords-Intermittent faults, Hamming codes.

I. INTRODUCTION

The advances in integration techniques have allowed microprocessors to raise their operating frequency and reduce their size and power voltage, achieving a greater performance. However, these advances have had a negative impact on their reliability, increasing the fault rate [1]. Whereas permanent and transient faults have been extensively studied, the analysis of intermittent faults is a pending issue.

Recent studies have focused in these questions [2][3][4] [5], but their research is still in an early phase of evolution. It is foreseen that intermittent faults will have a great impact in deep submicron technologies [6]. Currently, intermittent faults due to process variations and manufacturing defects have grown.

Errors induced by transient and intermittent faults manifest in a similar way, but the last ones are activated repeatedly in the same place, and usually appear in bursts. On the other hand, replacing the affected part eliminates an intermittent fault, while transient faults cannot be fixed by repair. Additionally, intermittent faults may be deactivated or activated by temperature, voltage and frequency changes [2].

In previous works we have generated fault models for intermittent faults at logic and RTL levels, and injected these faults in the VHDL model of a commercial microcontroller to study their impact on the system behavior [4][5]. Some parts of a system have shown to be especially sensitive to this kind of faults. For example, intermittent faults occurred in system buses provoke a great percentage of failures or latent errors.

In this work we propose the use of error correcting codes (ECCs) in order to make the system buses be fault-tolerant. We will study the characteristics and correcting capabilities of the proposed approach.

The paper is organized as follows. Section 2 describes some intermittent fault models. Section 3 reviews ECCs and their application to buses. Section 4 presents a proposal for a generic fault-tolerant bus. Finally, Section 5 provides ideas for future work.

II. INTERMITTENT FAULTS: MECHANISMS AND MODELS

Process variations, residues and special wear-out processes can lead to new intermittent faults. We have focused in those observed in interconnection lines in real computer systems, analyzing their causes and mechanisms.

Bursts of single bit errors (SBEs) have been observed in data buses (expecting the same problem in control lines) [7]. Failure analysis revealed that the source of errors was intermittent solder contacts. We have associated this mechanism to the following fault models: *intermittent stuck-at*, *intermittent short* and *intermittent open*.

Also, the propagation delays over the interconnection lines may be altered (due to PVT variations, crosstalk delays, crosstalk speed-ups, Miller effect, etc.) [8][9]. The fault models proposed to these mechanisms are *intermittent delay* and *intermittent speed-up*.

It can be assumed that intermittent faults may affect to more than one line. Multiple faults commonly may be adjacent (bad solder joints affecting some lines, crosstalk in adjacent signals, etc.) This fact must be taken into account in the design of mitigation techniques.

III. ECCS IN SYSTEM BUSES

We can consider a system bus as a binary symmetric channel. When a bus line is forced to a logic value, there is a probability that the received value had changed. ECCs allow the system to correct flipped bits. Information is encoded, sent by the channel (the bus), and decoded at destination. In this case, coding and decoding must be very fast operations, as the bus is used several times per processor instruction.

A. Linear block codes

An (n,k) block code encodes a k-bit input in an n-bit output. Assuming binary codes, they can be generated with parity-check matrices. Codes examples are: repetition, Hamming, Reed-Muller, and cyclic (Reed-Solomon and BCH). Although code generation can perform efficiently for linear block codes, the decoding problem may be difficult to solve [10]. Decoder complexity becomes major concern.

B. *Convolutional codes*

These codes have memory, that is, the current encoder output depends on the current input and on previous inputs. They are popular because they are able of achieving low bit error rates at signal-to-noise ratios close to the theoretical Shannon limit. Moreover, there exist efficient decoding algorithms [10].

C. *Other code constructions: Concatenation*

It is a method for constructing good codes by combining several simple codes. An encoder-channel-decoder system can be viewed as a super-channel with a smaller probability of error. We can create an encoder and decoder for it.

D. *Which is the best option?*

As mentioned, the main requirement is the speed, but the ability of tolerating multiple faults is also important.

Convolutional codes are unsuitable because they are not so fast. We have also rejected complex linear block codes (like Reed-Muller, Reed-Solomon, low-density parity-check codes, etc.) because they need complex decoding schemas, requiring a big amount of data to get good decoding results.

So, our proposal lies in classic and simple codes. The concatenation of a Hamming code with a triple repetition code will be studied in order to obtain its error detection and correction capabilities.

IV. OUR PROPOSAL

This approach, adapted from [11], has some interesting capabilities for error detection and correction. The studied version is for an 8-bit bus, but the design can be generalized to any bus width.

Let us consider an 8-bit word coded with a typical (12, 8) Hamming code, shortened from the perfect (15, 11) Hamming code. Bits $d_7..d_0$ are data bits, and $p_3..p_0$ are parity bits. Concatenation with a triple repetition code, repeating the whole word (not repeating bit by bit) allows us to obtain the final code word:

$$d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1\ d_0$$

$$d_7\ d_6\ d_5\ d_4\ \mathbf{p_3}\ d_3\ d_2\ d_1\ \mathbf{p_2}\ d_0\ \mathbf{p_1}\ \mathbf{p_0}$$

$$d_7 d_6 d_5 d_4 \mathbf{p_3} d_3 d_2 d_1 \mathbf{p_2} d_0 \mathbf{p_1} \mathbf{p_0} d_7 d_6 d_5 d_4 \mathbf{p_3} d_3 d_2 d_1 \mathbf{p_2} d_0 \mathbf{p_1} \mathbf{p_0} d_7 d_6 d_5 d_4 \mathbf{p_3} d_3 d_2 d_1 \mathbf{p_2} d_0 \mathbf{p_1} \mathbf{p_0}$$

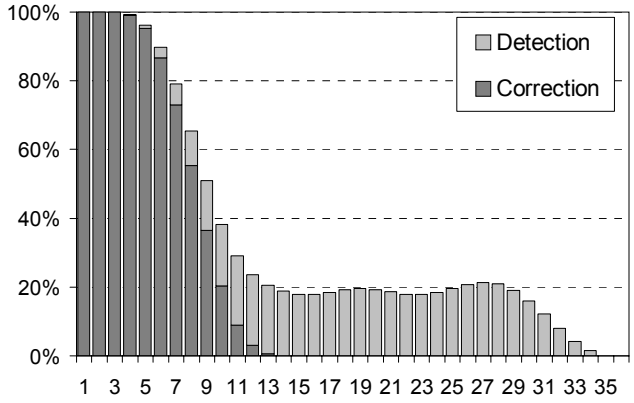


Figure 1. Coverage of error mitigation in function of flipped bits.

Coding circuit is as simple as to obtain the parity bits of 4 or 5 bits. The decoding circuit is also simple: in a first step, 12 voting circuits allow to get the 12-bit Hamming-coded word. A simple circuit calculates now the syndrome, which detects and corrects (if possible) an eventual flipped bit.

This code can correct up to 13 adjacent errors. For non adjacent errors, in the worst case 4 flipped bits may cause erroneous decoding; in the best case, 14 flipped bits allow a correct decoding. Fig. 1 shows the percentage of error detection and correction in function of the number of bits transmitted erroneously.

V. FUTURE WORK

The next step is to implement the proposed approach in the VHDL model of a microprocessor. Then, injection campaigns will be performed in order to evaluate the validity of the approach.

Trying to improve the efficiency of the code, we will review the intermittent fault mechanisms: the effect for some of them is one or more bus lines stuck at 0 or 1. That is, we may find multiple adjacent lines with the same logic value. So, better results can be obtained if we reduce the number of adjacent 1's or 0's in all correct code words. Combining even and odd parities, reverse logic and changing the order of some bits, this number can be reduced.

Lastly, mitigation techniques should be provided to other locations sensitive to intermittent faults, such as memories.

ACKNOWLEDGMENT

This work has been partially funded by the Spanish Government under the project "Sistemas empotrados seguros y confiables basados en componentes" (TIN2009-13825).

REFERENCES

[1] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability", IEEE Micro, Vol. 23(4):14-19, 2003.

[2] C. Constantinescu, "Impact of Intermittent Faults on Nanocomputing Devices", WDSN-07, Edinburgh, UK, June 2007.

[3] P.M. Wells, K. Chakraborty, G.S. Sohi, "Adapting to intermittent faults in multicore systems", ASPLOS'08, Seattle, USA, March 2008

[4] J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil, "Analysis of the influence of intermittent faults in a microcontroller", DDECS'08, pp. 80-85, Bratislava, Slovakia, April 2008.

[5] D. Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil, "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", HLDVT'08, pp. 177-184, November, 2008.

[6] C. Constantinescu, "Impact of Deep Submicron Technology on Dependability of VLSI Circuits", DSN'02, pp. 205-209, Washington D.C., USA, June 2002.

[7] C. Constantinescu, "Dependability Benchmarking using Environmental Test Tools". RAMS05, pp. 567–571, January 2005.

[8] W. Moore, G. Gronthoud, K. Baker, M. Lousberg, "Delay-fault testing and defects in deep sub-micron ICs-does critical resistance really mean anything?", ITC''00, pp. 95-104, Atlantic City, USA, October 2000.

[9] J.W. McPherson, "Reliability challenges for 45nm and beyond", DAC'06, pp. 176–181, July 2006.

[10] A. Neubauer, J. Freudenberger, V. Kühn, Coding theory: algorithms, architectures and applications, John Wiley, 2007.

[11] D.J.C. MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press, 2003

Experimental Validation of a Fault Tolerant Microcomputer System against Intermittent Faults*

J. Gracia-Moran, D.Gil-Tomas, L.J. Saiz-Adalid, J.C. Baraza, P.J. Gil-Vicente
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto ITACA
Universidad Politécnica de Valencia, Valencia, Spain
E-mail: {jgracia, dgil, ljsaiz, jcbaraza, pgil}@disca.upv.es

Abstract

As technologies shrink, new kinds of faults arise. Intermittent faults are part of these new faults. They are expected to be an increasing challenge in modern VLSI circuits. Up to now, transient and permanent faults used to be injected for the experimental validation of fault tolerance mechanisms. The main objective of this work is to improve the dependability assessment by injecting also intermittent faults. Furthermore, we have compared intermittent faults impact with the influence of transient and permanent faults. To carry out this study, we have injected bursts of intermittent faults in a fault-tolerant microcomputer system with some well known fault detection and recovery mechanisms. The methodology used lies in VHDL-Based Fault Injection technique, which allows a systematic and exhaustive analysis. Results show that intermittent faults have a notable impact on recovery mechanisms. They must be taken into account besides permanent and transient faults to implement an accurate dependability assessment.

1. Introduction

As the complexity of the manufacturing process increases, more residues and process variations are present in deep submicrometric and nanometric ICs, increasing the presence of intermittent faults. Also, this type of faults may be augmented by special wear-out mechanisms [1].

Even though transient and intermittent faults provoke errors manifested in a similar way, intermittent faults are activated repetitively in the same location, and so they are usually grouped in bursts. Alternatively, whereas replacing the affected part eliminates an intermittent fault, transient faults cannot be fixed by repair. Additionally, intermittent faults may be activated or deactivated by temperature, voltage or frequency changes [2].

On the other hand, fault mechanisms and models of transient and permanent faults, as well as their effects, have been widely studied [3][4][5], and different fault tolerance mechanisms (FTMs) have been designed to tolerate them [6][7]. Nevertheless, this type of studies has not been done very frequently for intermittent faults.

In previous works [8][9], we generated fault models for intermittent faults at logic and RTL abstraction levels, and injected these fault models in the VHDL model of a commercial microcontroller in order to study their impact on the system behavior. In these works, the influence of different fault and system parameters was analyzed.

After this analysis, next step is to study the impact of intermittent faults in Fault-Tolerant Systems (FTSs). The questions that emerge are: What happens when a FTS is affected by intermittent faults? Are current fault detection and recovery mechanisms enough to tolerate intermittent faults?

The purpose of this work is to answer these questions. To carry out this objective, we have used the VHDL model of a microcomputer system that implements different and well known FTMs. By using VHDL-based Fault Injection, we have checked if these mechanisms can tolerate intermittent faults. In this way, we intend to improve traditional fault-tolerant system validation based on injecting only transient and permanent faults. The main novelty of this paper is the injection of intermittent faults in a FTS in order to validate its FTMs.

This work is organized as follows. Section 2 depicts the fault injection environment, which includes a brief description of the fault models injected and the fault injection tool used. Section 3 describes the VHDL

* This work has been partially funded by the Spanish Government under the project "Sistemas empotrados seguros y confiables basados en componentes" (TIN2009-13825).

microcomputer model and the fault injection parameters, while Section 4 discusses the results obtained. Finally, Section 5 provides some conclusions and a proposal of future work.

2. Fault Injection Environment

We have used VHDL-based Fault Injection due to its flexibility, as well as the high observability and controllability of all the modeled components.

When injecting faults, an important issue is the definition of the fault models to be used in the injection, as they must be representative of real faults. This task requires studying the fault mechanisms considering the features of the technology used.

The intermittent fault models used in this work were deduced and presented in [8][9]. In these works, we selected a set of intermittent faults observed in real computer systems by means of fault logging [2][10], as well as fault mechanisms related to process variations and wear-out [2][11][12][13]. Then, we developed a set of fault models at logic (gate) and RTL abstraction levels which can be simulated into VHDL models.

Table 1 shows some examples of the studied intermittent faults and the associated fault models.

To inject the faults we have used a tool developed by our research group called VFIT (VHDL-based Fault Injection Tool) [14].

VFIT runs on PC computers (or compatible) under Windows®. VFIT is capable of injecting faults automatically applying the *simulator commands* technique, which consists on modifying the value or timing of the signals and/or variables of the model at simulation time. It is also feasible to inject faults using *saboteurs* and *mutants* [15], but in this case, this process needs the intervention of the user because the insertion of the saboteurs and the generation of mutants are not yet automatic. In this work we have used the technique based on *simulator commands*, as it is not necessary to modify the VHDL code and it is easy to apply.

3. Fault Injection Experiments

3.1. Microprocessor System

The different fault injection experiments were carried out on the VHDL model of a 16-bit fault-tolerant microcomputer, whose block diagram is shown in Figure 1. The system is duplex with cold stand-by sparing, parity detection and watchdog timer. Both main and spare processors are an enhanced version of the MARK2 processor [17]. It is a simple FTS, but its structure (dual with cold spare) is common in non-critical FTS such as long-life and high availability systems [7].

The structural architecture of the model is composed by the main and spare CPUs (CPUA and CPUB, respectively), the RAM memory (MEM), the output parallel port (PORTOUT), the interrupt controller (SYSINT), the clock generator (CLK), the watchdog timer (WD), the pulse generator (GENINT), two back-off cycle generators (TRGENA, TRGENB) and two AND gates (PAND2A, PAND2B).

Several FTMs have been added to increase system dependability. The error detection mechanisms (EDMs) include parity check in memory, and program control flow check performed by a watchdog timer. The error recovery mechanisms (ERMs) include the introduction of a back-off (instruction retry) cycle after parity error detection, checkpointing when errors are detected by the watchdog timer, and starting the spare processor in case of permanent errors. The number of error detections required to activate the spare CPU can be configured in the system. More details about the system and its FTMs can be found in [16].

Table 1. Some intermittent faults mechanisms and models [9]

Causes	Places	Fault mechanisms	Type of fault	Fault Models
Residues in cells	Memory and registers	Intermittent contacts	Manufacturing defect	Intermittent stuck-at
Solder joints	Buses	Intermittent contacts	Manufacturing defect	Intermittent pulse Intermittent short Intermittent open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance	Wearout-timing	Intermittent delay Intermittent short Intermittent open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing Voltage perturbation	Intermittent delay Intermittent pulse
Soft breakdown	FET transistors in SRAM cells	Leakage current fluctuation	Wearout-timing	Intermittent delay Intermittent indetermination

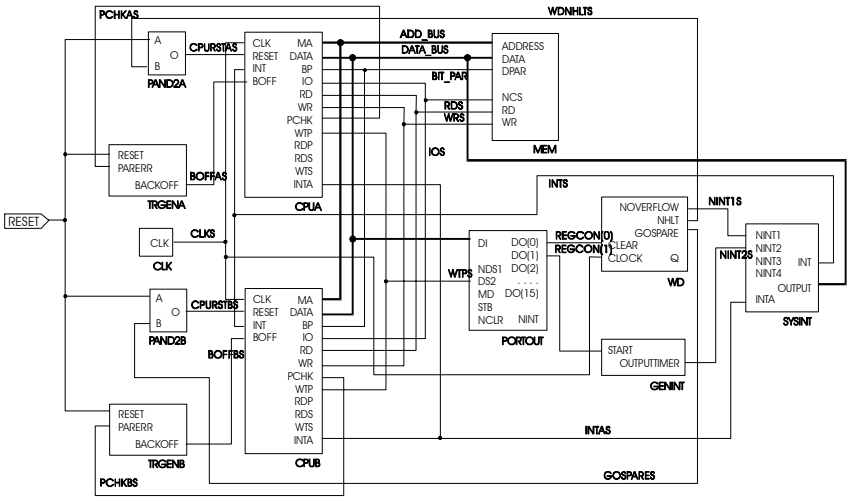


Figure 1. Block diagram of the computer system [16]

3.2. Fault Injection Parameters

A set of independent fault injection experiments have been carried out. For each experiment, the main injection parameters are:

Injection targets: buses (address, data and control) and memory (MEM component). They have been observed to be potential places for intermittent faults (see Table 1).

Number of faults injected: 1000 per experiment, being classified as single (one burst in a single place) or multiple (several bursts in different places).

Fault models injected: intermittent pulse in the buses, and intermittent stuck-at in the memory (see Table 1). Note that intermittent stuck-at is different from a burst of bit-flips, as they have different origins.

Injection instant: selected randomly along the workload duration, according to a Uniform distribution.

Workload: Arithmetic Series and Bubblesort algorithms, which are two typical and moderate-duration workloads.

Burst parameters: as stated above, intermittent faults manifest in bursts. So, to inject this type of faults the following parameters must be configured (see Figure 2):

- L_{Burst} (*burst length*): number of fault activations in the burst, selected randomly between [1, 10];
- t_A (*activity time*): duration of each activation;
- t_I (*inactivity time*): separation between two consecutive activations.

Activity time (t_A) and inactivity time (t_I) were generated randomly according to a Uniform distribution function in three time intervals: [0.01T, 0.1T], [0.1T, 1.0T] and [1.0T, 10.0T], where T is the system clock cycle.

To assess the impact of intermittent faults, we have calculated the percentages shown in Table 2.

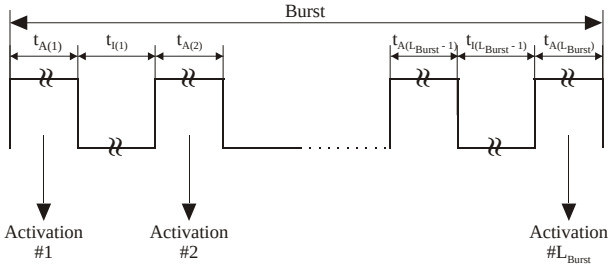


Figure 2. Main elements of a burst

Table 2. Percentages calculated

Act	Percentage of activated errors. External signals or variables have changed.
No_Effect	Percentage of non effective errors (errors which are overwritten or remain latent in the system, but do not provoke any failure).
Detect	Percentage of detected errors.
ND_Fail	Percentage of undetected errors that have provoked a failure.
D_PAR	Percentage of errors detected by the parity.
D_WD	Percentage of errors detected by the watchdog timer.
Recov	Percentage of recovered errors by the ERMs.
NR_Fail	Percentage of unrecovered errors that have provoked a failure.
R_BOFF	Percentage of errors recovered by the back-off.
R_CP	Percentage of errors recovered by the checkpointing.
R_SP	Percentage of errors recovered by the spare CPU.

Fault pathology is reflected in the graph of Figure 3. This graph shows the evolution of faults from the injection to the detection and possible recovery by the FTMs.

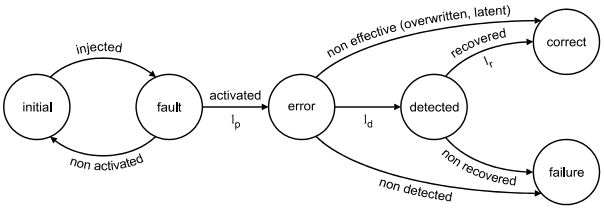


Figure 3. Fault pathology graph [14]

4. Results

4.1. Injection in the buses

Table 3 and Table 4 show the results of injecting single intermittent faults in the system buses.

Respect to error activation, the lowest percentage of activated errors (*Act* column) is greater than 93% for both workloads, showing that buses are critical elements of the system, as well as very sensitive to intermittent faults. Bubblesort workload has higher values than arithmetic series, as it uses the buses more intensively.

Respect to error detection, the percentage of detected errors (*Detect* column) shows values between 47% and 93%, depending on the activity time (t_A) and the workload. It is greater for longer values of t_A and for bubblesort workload. An increase of t_A implies a higher impact in the buses, and the bubblesort algorithm produces more traffic in the buses.

The percentage of failures provoked by undetected errors (*ND_Fail* column) is very low (less than 5% in the worst case). This means that almost all activated errors are detected (*Detect* column) or non effective (*No_Effect* column).

For both workloads, parity is the most effective EDM (*D_PAR* and *D_WD* columns). For the arithmetic series, the percentages are very similar (between 81% – 85%), while for the bubblesort algorithm the variation is bigger (between 76% – 85%). In this case, the lowest percentage of detected errors occurs when the activity time is in the range [0.01T, 0.1T]. That is, bursts with short activation times are more difficult to be managed by the parity. The watchdog timer detects fewer faults than parity. It shows values between 15% and 24%.

No significant influence of the inactivity time (t_I, or separation between the fault activations) has been observed in the percentage of detected errors.

Although the EDMs work well, the ERMs should be improved. The percentages of failures provoked by detected and non recovered errors (*NR_Fail* column) are especially large (between 23% and 34%) for both workloads. These mechanisms work better with the bubblesort algorithm. Note that (*Recov* + *NR_Fail*) may be lower than 100%, as some errors are recovered by the intrinsic redundancy of the system.

Checkpointing presents the lowest percentage of recovered errors (*R_CP* column), mainly for bubblesort algorithm. This is provoked by the watchdog timer's low percentage of detected errors. The combination of back-off and spare processor recovers most of errors. Back-off mechanism (*R_BOFF* column) works well for shorter durations of the activity time, while for longer durations, the spare (*R_SP* column) recovers the biggest part of errors. In this case, the system interprets that the fault is permanent.

Additional experiments have been performed injecting multiple intermittent faults in the system buses (that is, various bursts in different bus lines). We do not show the corresponding tables for lack of space. We have observed the same trend than in the single fault case, although the values are bigger.

4.2. Injection in the memory

Regarding faults injected in external memory, we do not show the corresponding tables for lack of space.

In this case, the percentages of activated errors are very small (lower or equal to 1%). This is because both workloads use less than a 5% of memory size, and the target selection is made randomly along all the memory. Thus, the probability for the workload to sensitize and propagate them to external signals is very low. As expected, these percentages rise with greater values of t_A. Concerning the workloads, we can observe bigger values for bubblesort algorithm.

All errors are detected or non effective. Undetected errors do not provoke any failure. Parity mechanism detects 100% of errors.

The biggest part of detected errors is recovered by back-off and spare mechanisms. Back-off works well for short durations of the activity time, while spare works better for longer durations, as observed in buses.

As expected, when injecting multiple faults, more errors are activated, as seen in buses. Respect to detection and recovery, we have seen the same tendencies than in the single fault injection case.

4.3. Comparison to transient and permanent faults

Finally, the effect of intermittent faults has been compared to the impact of transient and permanent faults. Transient faults have been injected in memory (*bit-flips*) and buses (*pulses* with a duration randomly defined in the interval [0.1T, 1.0T]). Fault models used

for permanent faults in both places has been *stuck-at* ('0', '1'), *indetermination* and *open line*. For the comparison, we have selected intermittent faults with t_A and t_I varying uniformly in the range [0.1T, 1.0T]. Table 5 shows a sample of the results obtained for multiple faults.

We have observed that when injecting faults in buses, the percentages of detected errors (*Detect* column) follows the trend: permanent > intermittent > transient. Transient faults are the most difficult to detect. The percentage of failures provoked by undetected errors (*ND_Fail* column) is very low for all fault types (lower than 2%). Parity is the most effective EDM (*D_PAR* column). Only for transient faults, the watchdog timer presents some relevance (23.66%).

The percentage of recovered errors (*Recov* column) follows the tendency: transient > intermittent >> permanent. The percentage of failures provoked by detected and unrecovered errors (*NR_Fail* column) is very high for permanent faults (97.76%). Permanent faults in buses affect the behavior of the spare CPU, preventing system recovery. Intermittent faults provoke a non negligible percentage of failures (11%). The most effective ERM is the spare, followed by the back-off. The highest activation of back-off is observed for transient and intermittent faults.

Table 3. Impact of single intermittent faults injected in the buses (arithmetic series)

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]	94.80	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

Table 4. Impact of single intermittent faults injected in the 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

Table 5. Comparison of multiple transient, intermittent and permanent faults (bubblesort)

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

The results obtained from injecting in the memory show a similar trend. Nevertheless, a very low percentage of activated errors have been obtained due to the small percentage of memory usage. This can affect the statistical accuracy of some results, especially for intermittent faults.

5. Conclusions and Future Work

In this work, we have presented a study of the effects of intermittent faults in the behavior of a FTS. This study is motivated by the increasing incidence of intermittent faults in deep submicrometric technologies. Faults have been injected in buses and memory, as they have been proved to be susceptible to intermittent faults.

We have used VHDL-based Fault Injection due to its flexibility, as well as the high observability and controllability of all the modeled components. To sum up the results:

- We have verified a good performance of the detection mechanisms implemented. Almost all the activated errors are detected or non effective. Parity presents much better results than the watchdog timer.
- Recovery mechanisms do not work as well as detection mechanisms. There are a non negligible percentage of failures provoked by errors detected, but not recovered. Retry and spare are the most effective recovery mechanisms. Retry works well with short durations of the burst activations, while spare works better for longer durations and when multiple faults are present.
- Recovery mechanisms should be improved. Some possibilities are adding ECC to memory in order to improve the recovery of parity errors, replacing cold spare with hot spare so as to reduce recovery latency and to increase recovery coverage, and using TMR at internal level of the CPU to mask transient and short intermittent faults in buses. This last technique also would reduce or even eliminate the recovery latency.
- Respect to the comparison with transient and permanent faults, detection mechanisms work well for permanent and intermittent faults. Transient faults are the most difficult to detect. It would be advisable to increase detection mechanisms in this case by means of exceptions, parity in buses, watchdog processor to monitor buses, etc. Permanent faults, especially in buses, are the more difficult faults to recover. As commented before, intermittent faults present also a notable impact on the recovery mechanisms.

In the near future, we intend to complete our research with some other works related to validate more complex and real systems with different architectures, to inject new intermittent fault models with different fault injection techniques, as *saboteurs* and *mutants*, and to inject faults in other targets different from buses and memory.

6. References

[1] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, in Procs. DSN 2002, pp. 205-209, USA, 2002.

[2] C. Constantinescu, “Impact of Intermittent Faults on Nanocomputing Devices”, in Procs. WDSN-07, UK, 2007, availabe at http://www.laas.fr/WDSN07.

[3] E.A. Amerasekera and F.N. Najm, “Failure mechanisms in semiconductor devices”, John Whiley & Sons, 1997.

[4] P.J. Gil et al., “Fault Representativeness”, Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245, 2002.

[5] C. Hawkins et al., “CMOS IC nanometer technology failure mechanisms”, in Procs. IEEE CICC 2003, pp. 605-611, USA 2003.

[6] D.P. Siewiorek and R. S. Schwarz: “Reliable computer systems: Design and evaluation”, 2nd Edition, Ed. Digital Press, 1992.

[7] D.K.Pradhan, “Fault-tolerant computing: theory and techniques”, Prentice Hall, 1992.

[8] J. Gracia et al., “Analysis of the influence of intermittent faults in a microcontroller”, in DDECS’08, pp. 80-85, Slovakia, 2008.

[9] D. Gil et al., “Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers”, in HLDVT’08, pp.177-184, USA, 2008.

[10] C. Constantinescu, “Dependability Benchmarking using Environmental Test Tools”, in RAMS’05, pp. 567–571, 2005.

[11] J.C. Smolens et al., “Detecting Emerging Wearout Faults”, 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), 2007, availabe at http://selse3.selse.org/Papers/10_Smolens_P.pdf.

[12] S. Borkar et al., “Parameter Variations and Impact on Circuits and Microarchitecture”, in DAC’03, pp. 338–342, 2003.

[13] J.W. McPherson, “Reliability challenges for 45nm and beyond”, in DAC’06, pp. 176–181, July 2006.

[14] D. Gil et al., “VHDL Simulation-Based Fault Injection Techniques” in Benso & Prinetto eds., “Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation”, Kluwer Academic, 2003.

[15] E. Jenn et al., “Fault Injection into VHDL Models: The MEFISTO Tool” in FTCS, pp. 66-75, 1994.

[16] D. Gil et al., “Fault Injection into VHDL Models: Experimental Validation of a Fault Tolerant Microcomputer System” in EDCC’03, pp. 191-208, 1999.

[17] J.R. Armstrong, “Chip-Level Modelling with VHDL”, Prentice Hall, 1989.

A Proposal to Tolerate Intermittent Faults in Microprocessor Buses

Luis-J. Saiz-Adalid, J.-Carlos Baraza-Calvo, Joaquín Gracia-Morán, Daniel Gil-Tomás, Pedro-J. Gil-Vicente
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas – Universidad Politécnica de Valencia
Camino de Vera, s/n, 46022, Valencia, Spain
{ljsaiz, jcbaraza, jgracia, dgil, pgil}@disca.upv.es

***Abstract*—A decrease in the reliability of processors is the consequence of increasing their performance and reducing their size. Faults due to manufacturing residuals and process variations have grown. Transient and intermittent faults are expected to be a big challenge in modern VLSI circuits, in addition to permanent faults. Different studies have revealed that interconnection lines and storage cells are very sensitive to these kinds of faults.**

In this paper we improve a proposal of fault-tolerant coding method presented previously, in order to apply it to system buses. The proposal lies in the use of Hamming codes. Although the use of concatenated Hamming codes is not new, the main novelty of the approach is the use of reverse logic to reduce the effects of intermittent faults in buses. Its features and fault tolerance capabilities are also revised.

Keywords-Intermittent faults, Hamming codes.

I. INTRODUCTION

The advances in integration techniques have allowed microprocessors to raise their operating frequency and reduce their size and power voltage, achieving a greater performance. However, these advances have had a negative impact on their reliability, increasing the fault rate, particularly due to transient and intermittent faults [1]. Recent studies have focused in intermittent faults [2][3][4] [5], but their research is still in an early phase of evolution. It is foreseen that intermittent faults will have a great impact in deep submicron technologies. Now, intermittent faults due to process variations and manufacturing defects have grown.

Errors induced by transient and intermittent faults show a similar behavior, but the last ones usually appear in bursts and are activated repeatedly in the same place. Replacing the affected part eliminates an intermittent fault, while transient faults cannot be fixed by repair. Additionally, intermittent faults may be deactivated or activated by temperature, voltage or frequency changes [2]. Some parts of a system have shown to be especially sensitive to this kind of faults [4][5]. Particularly, intermittent faults occurred in system buses provoke a great number of failures or latent errors.

In this work we propose the use of error correcting codes (ECCs) in order to make the system buses fault-tolerant. We will study an improvement to a previous proposition [6], its characteristics and correcting capabilities.

Section 2 reviews our previous approach for a generic fault-tolerant bus. Section 3 describes some intermittent faults causes and mechanisms, and studies the new proposal and its fault tolerance capabilities. Finally, Section 4 provides ideas for future work.

II. ECC IN SYSTEM BUSES: FIRST PROPOSAL

We can consider a system bus as a binary symmetric channel. When a bus line is forced to a logic value, there is a probability that the received value had changed. ECCs allow the system to correct flipped bits. Information is encoded, sent by the channel (the bus), and decoded at destination. In this case, coding and decoding must be very fast operations, as the bus is used several times per processor instruction.

Our first approach [6], adapted from [7], has some interesting capabilities for error detection and correction. The studied version is for an 8-bit bus, but the design can be generalized to any bus width.

Let us consider an 8-bit word (1) coded with a typical (12, 8) Hamming code (2), shortened from the perfect (15, 11) Hamming code. Bits $d_7..d_0$ are data bits, and $p_3..p_0$ are even parity bits. Concatenation with a triple repetition code, repeating the whole word (not repeating bit by bit) allows us to obtain the final code word (3):

$$d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1\ d_0 \quad (1)$$

$$d_7\ d_6\ d_5\ d_4\ \mathbf{p_3}\ d_3\ d_2\ d_1\ \mathbf{p_2}\ d_0\ \mathbf{p_1}\ \mathbf{p_0} \quad (2)$$

$$d_7d_6d_5d_4\mathbf{p_3}d_3d_2d_1\mathbf{p_2}d_0\mathbf{p_1}\mathbf{p_0}d_7d_6d_5d_4\mathbf{p_3}d_3d_2d_1\mathbf{p_2}d_0\mathbf{p_1}\mathbf{p_0}d_7d_6d_5d_4\mathbf{p_3}d_3d_2d_1\mathbf{p_2}d_0\mathbf{p_1}\mathbf{p_0} \quad (3)$$

Coding circuit is as simple as to obtain the parity bits of 4 or 5 bits. The decoding circuit is also simple: in a first step, 12 voting circuits allow to get the 12-bit Hamming-coded word. A simple circuit calculates now the syndrome, which detects and corrects (if possible) an eventual flipped bit.

This code can correct up to 13 adjacent errors. For non adjacent errors, in the worst case 4 flipped bits may cause erroneous decoding; in the best case, 14 flipped bits allow a correct decoding. Fig. 1 shows the percentage of error detection and correction with respect to the number of bits transmitted erroneously.

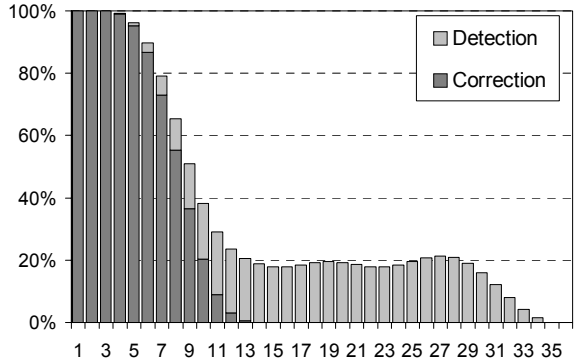


Figure 1. Coverage of error mitigation with respect to flipped bits.

III. IMPROVED ECC FOR INTERMITTENT FAULTS

Process variations, residues and special wear-out processes can lead to new intermittent faults. We have focused in those observed in interconnection lines in real computer systems, analyzing their causes and mechanisms.

Bursts of single bit errors (SBEs) have been observed in data buses (expecting the same problem in control lines) [2][8]. Failure analysis revealed that the source of errors was intermittent solder contacts. Also, the propagation delays over the interconnection lines may be altered. This can be due to crosstalk delays and the increase of the resistance provoked by wear-out mechanisms [9][10]. We have associated these fault mechanisms to intermittent pulse and intermittent delay fault models [4][5].

It can be assumed that intermittent faults may affect to more than one line. Multiple faults commonly may be adjacent (bad solder joints affecting some lines, crosstalk in adjacent signals, etc.) This fact must be taken into account in the design of mitigation techniques.

As stated above, bad solder joints are one of the typical mechanisms of intermittent faults. It may cause one or more bus lines flipped. If more than one line is affected, they are usually adjacent. That is, we may find multiple adjacent lines with the same logic value. So, better results can be obtained if we reduce the number of adjacent 1's or 0's in all correct code words. Combining even and odd parities, reverse logic and changing bits order, this number can be reduced. Even more, a big number of adjacent lines with the same logic value may affect to the same data bit more than once, causing a wrong output in the decoder voting circuit.

For example, in the previous approach, the value 0x00 is coded as 0x000000000. If the bus is affected by 14 adjacent lines with 1's the word is decoded incorrectly.

A first improvement could be the use of odd parity. In this way, the longest adjacent bit sequence with the same logical value is 11. Anyway, the above problem also occurs: 14 adjacent lines with 1's, if the bits affected twice have the 0 value (or vice versa), make the word be badly decoded.

In order to avoid this problem, we can use reverse logic for coding the central repetition of our coding approach. Although the biggest number of adjacent values is 12 bits (one bit more than the previous proposal), a bus with this coding schema supports an error of 25 adjacent bits with the same value. Fig. 2 shows the coverage of error detection and correction in function of the error length.

Combining this solution with even or odd parities, reordering and reversing some bits, we have found a coding schema with 10 bits longest sequence.

The coding circuits are groups of exclusive or gates to calculate parity bits and inverters. The decoding circuits need voters, exclusive or circuits to calculate the syndrome, a decoder and exclusive or gates to correct the error.

The advantage of these proposals is their great coverage. But there are two main disadvantages. The first is the excessive redundancy: an 8-bit word is coded in a 36-bit bus. But the most important disadvantage is the sensitivity to delay errors induced by crosstalk: when two lines switch to opposite values, capacitive effects may cause a delay in both lines, producing (possibly) two flipped bits. The situation will be the same in the three repetitions, so same bits may be affected more than once, producing a wrong decoding.

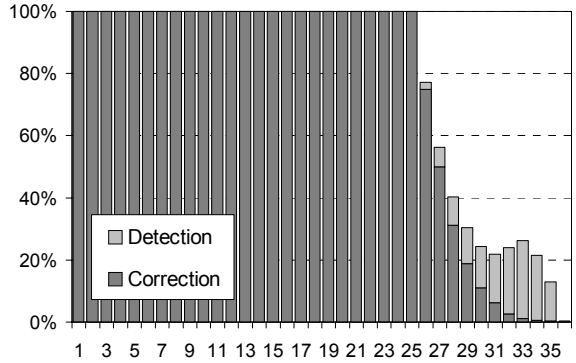


Figure 2. Coverage of error mitigation with respect to the number of adjacent bits fixed to the same value.

IV. FUTURE WORK

We also have in mind other proposals. The objective is to reduce the redundancy, or to avoid crosstalk effects, maintaining adjacent error correcting capabilities. For example, chipkill-correct ECC used in memories [11] could also be used in buses. The number of adjacent errors supported can be decided during the design phase.

The next step is to implement our proposals in the VHDL model of a microprocessor. Injection campaigns will be performed in order to evaluate the validity of the approaches.

ACKNOWLEDGMENT

This work has been partially funded by the Spanish Government under the project "Sistemas empotrados seguros y confiables basados en componentes" (TIN2009-13825).

REFERENCES

[1] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability", IEEE Micro, Vol. 23(4):14-19, 2003.

[2] C. Constantinescu, "Impact of Intermittent Faults on Nanocomputing Devices", WDSN-07, Edinburgh, UK, June 2007.

[3] P.M. Wells, K. Chakraborty, G.S. Sohi, "Adapting to intermittent faults in multicore systems", ASPLOS'08, Seattle, USA, March 2008

[4] J. Gracia et al., "Analysis of the influence of intermittent faults in a microcontroller", DDECS'08, pp. 80-85, Slovakia, April 2008.

[5] D. Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil, "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", HLDVT'08, pp. 177-184, November, 2008.

[6] L.J. Saiz, J.C. Baraza, J. Gracia, D. Gil, "A Proposal of a Fault-Tolerant Mechanism for Microprocessor Buses", EDCC'10, Supplemental Volume, Valencia, Spain, April 2010.

[7] D.J.C. MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press, 2003.

[8] C. Constantinescu, "Dependability Benchmarking using Environmental Test Tools". RAMS05, pp. 567–571, January 2005.

[9] W. Moore, G. Gronthoud, K. Baker, M. Lousberg, "Delay-fault testing and defects in deep sub-micron ICs-does critical resistance really mean anything?", ITC'00, pp. 95-104, USA, October 2000.

[10] J.W. McPherson, "Reliability challenges for 45nm and beyond", DAC'06, pp. 176–181, July 2006.

[11] T.J. Dell, "A White Paper on the Benefits of Chipkill-correct ECC for PC Server Main Memory", IBM, 1997.

Anexo II: Trabajo de investigación

A continuación se incluye el texto completo del trabajo de investigación.

UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

ANÁLISIS DE NUEVOS MODELOS DE FALLOS INTERMITENTES PARA NUEVAS TECNOLOGÍAS

Trabajo de investigación tutelado

Programa de doctorado:

Arquitectura de los Sistemas Informáticos en Red y Sistemas Empotrados

Presentado por Luis José Saiz Adalid

Dirigido por Joaquín Gracia Morán y Juan Carlos Baraza Calvo

Valencia, Diciembre de 2009

Índice

1

Introducción

1.1 Presentación

En esta monografía se presenta el trabajo realizado hasta el momento por el autor durante los estudios de doctorado realizados en el Departamento de Informática de Sistemas y Computadores (DISCA) de la Universidad Politécnica de Valencia (UPV), dentro del programa de doctorado de Arquitectura de los Sistemas Informáticos en Red y Sistemas Empotrados, principalmente en la línea de investigación seguida hasta el momento con la tesis doctoral como objetivo final.

La investigación realizada se centra en el estudio de la tolerancia a fallos en sistemas informáticos. El autor está trabajando en el Grupo de investigación de Sistemas Tolerantes a Fallos (GSTF) del Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA) de la UPV. En concreto, ha trabajado con un grupo de investigadores, con el Dr. Pedro Gil a la cabeza, cuyo interés se centra en el diseño y validación de sistemas tolerantes a fallos.

Este primer capítulo incluye, además de la presentación, los objetivos de este trabajo de investigación. En el segundo se hablará de los conceptos básicos sobre tolerancia a fallos y confiabilidad. El tercer capítulo presenta las distintas técnicas de inyección de fallos, centrándose en las técnicas basadas en simulación sobre modelos en VHDL. A continuación, en el cuarto capítulo se analiza la problemática de la representatividad de los modelos de fallos, y se proponen una serie de modelos de fallos intermitentes que se han deducido a partir del trabajo realizado en los últimos meses. Posteriormente, en el quinto capítulo se hará una breve presentación de la herramienta de inyección de fallos utilizada para llevar a cabo las inyecciones de fallos. El sexto capítulo detallará las consecuencias de los fallos intermitentes, obtenidas a través de una serie de experimentos que se han llevado a cabo utilizando la herramienta de simulación descrita en el capítulo quinto, y aplicando los modelos de fallos propuestos en el capítulo cuarto. En el séptimo capítulo se presentarán las conclusiones del trabajo, que incluyen los análisis de resultados de los experimentos realizados, así como una descripción de líneas de investigación abiertas para el trabajo futuro y la presentación de la tesis doctoral. Para finalizar se presenta la bibliografía utilizada. Se incluye además un anexo con las publicaciones realizadas en relación con este trabajo de investigación.

1.2 Objetivos

Las tecnologías submicrométricas utilizadas hoy en día para diseñar y fabricar los sistemas informáticos actuales, y los avances en las técnicas de integración, tienen como resultado una reducción del tamaño de los transistores empleados, una disminución de la tensión de alimentación utilizada y un incremento importante en la frecuencia de trabajo. Esto conlleva mayores prestaciones, pero por otro lado la confiabilidad se ve comprometida. Se prevé un aumento en la tasa de ocurrencia de fallos, y como consecuencia, los diseñadores de sistemas deben asumir que los circuitos integrados son más sensibles a distintos tipos de fallos.

Si nos centramos en la persistencia temporal de los fallos que pueden afectar a los circuitos, podremos distinguir entre **fallos permanentes** (producidos por defectos físicos irreversibles debidos a defectos de fabricación o a procesos de desgaste y envejecimiento), **fallos transitorios** (habitualmente provocados por el entorno físico externo y que raramente se vuelven a producir en el mismo lugar), y **fallos intermitentes**. Estos últimos han sido considerados habitualmente como el inicio de un proceso de desgaste, y por tanto como el preludio de un defecto físico irreversible que acaba en un fallo permanente. Sin embargo, en la actualidad se ha observado que existen fallos intermitentes que no conducen a fallos permanentes. Origen de estos fallos pueden ser la complejidad de los procesos de fabricación, que provoca residuos o variaciones en el proceso, junto con mecanismos de desgaste. La ocurrencia de determinados mecanismos físicos, donde se combina un *hardware* defectuoso con unas condiciones de trabajo concretas, puede propiciar la aparición de fallos intermitentes que no conduzcan a un fallo permanente. Por otra parte, diversos estudios destacan la importancia que van a alcanzar los fallos intermitentes en las nuevas tecnologías submicrométricas.

Los errores inducidos por fallos transitorios e intermitentes se manifiestan de forma similar. La principal característica de los fallos intermitentes está en que éstos se activan repetidamente en el mismo lugar, y habitualmente aparecen en ráfagas. Además, cambiar el componente afectado elimina el fallo intermitente, lo que no sucede con los fallos transitorios, que no pueden ser reparados. Los fallos intermitentes también se pueden activar o desactivar por cambios de temperatura, tensión de alimentación o frecuencia de trabajo (cambios conocidos como variaciones PVT, del inglés *Process, Voltage and Temperature*).

En relación con los fallos intermitentes, hay varias cuestiones difíciles de contestar: ¿dónde se producen los fallos intermitentes? ¿Cuándo se activan? ¿Cuántas veces se activa el fallo en una ráfaga? ¿Cómo se manifiesta el fallo a niveles de abstracción superiores? ... Para responder a estas cuestiones es importante entender los mecanismos físicos que tienen lugar en las tecnologías submicrométricas. Pero esta investigación es compleja, dependiente de la tecnología, y todavía está en una fase temprana de desarrollo.

El conocimiento de las causas de los fallos intermitentes no es suficientemente profundo, ya que se han publicado pocos trabajos de observación de fallos reales, o estudios de sus causas y mecanismos físicos.

Los fallos intermitentes han sido tradicionalmente olvidados a la hora de analizar las causas y mecanismos de los fallos reales que ocurren en los sistemas digitales. No ha sido hasta hace unos pocos años cuando los investigadores han prestado atención a estos tipos de fallos, y alertado de su creciente importancia. En la mayoría de los trabajos publicados se han monitorizado sistemas reales, observando los fallos y averías provocados. Tras su estudio, se han determinado las fuentes de errores y sus manifestaciones más frecuentes, y se ha descubierto que eran causados por fallos intermitentes. Sin embargo, en la literatura existente no se han encontrado trabajos en los que se haya intentado estudiar los efectos de los fallos intermitentes mediante simulación.

A la hora de utilizar modelos de fallos, una de las cuestiones fundamentales es su representatividad, es decir, la equivalencia del modelo de fallo obtenido con el fallo real, para el nivel de abstracción utilizado. El objetivo de este trabajo es determinar y utilizar modelos de fallos a nivel de puertas lógicas y de transferencia entre registros. La representatividad a nivel de *hardware* de los modelos de fallos utilizados para fallos permanentes y transitorios a este nivel de abstracción ha sido profusamente documentada, y los modelos definidos están bien establecidos por la comunidad científica. Sin embargo, para fallos intermitentes esto no ha sido así, y no han sido muchos los trabajos 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 uno permanente. Lo cierto es que con las nuevas tecnologías submicrométricas, la reducción de tamaños, el incremento de la frecuencia de funcionamiento y la necesidad de ajustar el consumo de energía, cada vez es más frecuente la aparición de fallos intermitentes que no dan lugar a fallos permanentes. Para ello, se debe tener en cuenta la influencia de los defectos de fabricación, como residuos o variaciones del proceso. El estudio de las causas, mecanismos y efectos de los fallos intermitentes es una línea de investigación abierta en la que es necesario profundizar.

En todo caso, con la investigación realizada hasta el momento, y los conocimientos actuales de las causas y mecanismos de los fallos intermitentes, ya se pueden deducir algunos modelos de fallos intermitentes. Determinar estos modelos, incluyendo su parametrización, es el objetivo de este trabajo.

Finalmente, y utilizando los modelos deducidos, se incluirán algunos experimentos llevados a cabo con la herramienta de simulación desarrollada por el grupo de investigación, con el objetivo de iniciar el estudio de los efectos de los fallos intermitentes en un microcontrolador comercial. La metodología de simulación utilizada será la inyección de fallos en modelos VHDL (del inglés *Very high speed integrated circuit Hardware Description Language*), que se caracteriza por la gran flexibilidad, observabilidad y controlabilidad de los componentes modelados, y que permitirá obtener un análisis sistemático y exhaustivo de la influencia de los distintos parámetros estudiados, sin necesidad de esperar mucho tiempo observando la aparición de fallos reales.

Es importante remarcar que, aunque el estudio se va a realizar en un ejemplo concreto de microcontrolador, la metodología utilizada se puede generalizar a sistemas más complejos.

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. En la bibliografía se utilizan estos conceptos a veces 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.

2.2 Definiciones básicas

La **confiabilidad** (en 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 da lugar a la **disponibilidad** (en inglés *availability*).
- La continuidad de un servicio correcto da lugar a la **fiabilidad** (en inglés *reliability*).
- La no ocurrencia de consecuencias catastróficas en el entorno da lugar a la **inocuidad** (en inglés *safety*).

- La no ocurrencia de revelaciones no autorizadas de información, da lugar a la **confidencialidad** (en inglés *confidentiality*).
- La ausencia de alteraciones impropias del sistema da lugar a la **integridad** (en inglés *integrity*).
- La capacidad para someterse a reparaciones y modificaciones da lugar a la **mantenibilidad** (en inglés *maintainability*).
- La asociación de la integridad y disponibilidad respecto a acciones autorizadas, junto con la confidencialidad, da lugar a la **seguridad** (en 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 no debe ser interpretada como indicación de que, por ejemplo, los desarrolladores de seguridad no tienen interés en la mantenibilidad; o que no se tiene en cuenta la confidencialidad cuando se investiga en confiabilidad. Más bien, indica dónde cae en cada caso el mayor balance de interés y actividad.

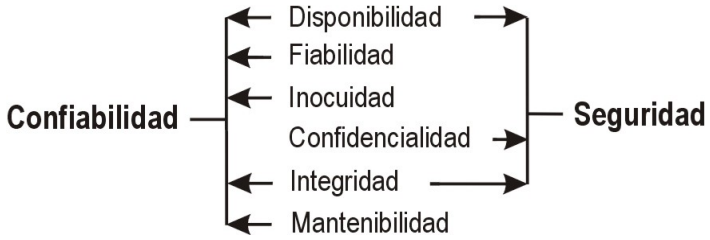


Figura 2.1: Atributos de la confiabilidad y la seguridad.

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. Puede suceder, para un sistema dado, que uno o más atributos no sean en absoluto requeridos.

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, y 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:

- Los **atributos** de la confiabilidad: disponibilidad, fiabilidad, inocuidad, confidencialidad, integridad y mantenibilidad. Permiten, en primer lugar, expresar las propiedades que se esperan de un sistema, y en segundo, valorar la calidad del servicio entregado, que es resultante de la interacción entre las amenazas y los medios que se oponen a éstos.
- Las **amenazas** de la confiabilidad: fallos, errores y averías. Son circunstancias no deseadas, pero no inesperadas en principio, que causan o tienen como resultado la no confiabilidad. Esta no confiabilidad indicará que en adelante no se puede, o no se podrá tener confianza en el servicio ofrecido por el sistema.
- Los **medios** para conseguir la confiabilidad: prevención de fallos, tolerancia a fallos, eliminación de fallos, predicción de fallos. Son métodos y técnicas para:
    1. Dar capacidad al sistema para entregar un servicio en el que se pueda confiar.
    2. Tener confianza en esa capacidad.

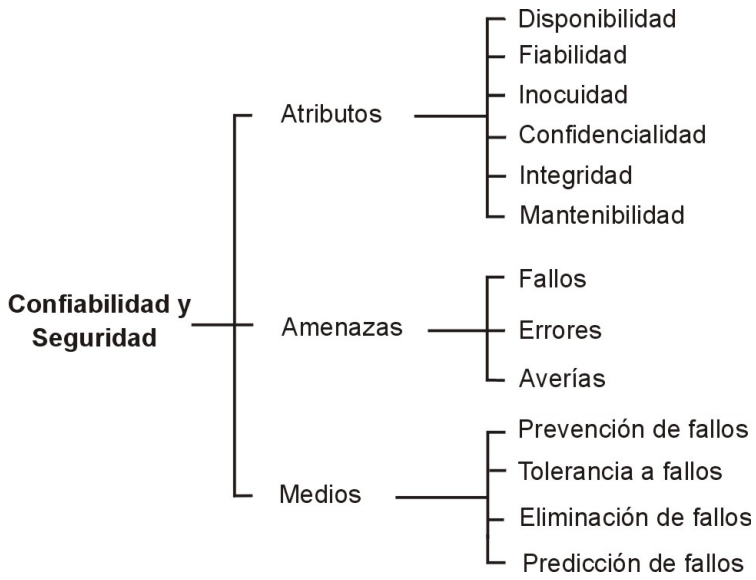


Figura 2.2: Árbol de la confiabilidad.

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, inocuidad y confidencialidad, se pueden requerir o no, dependiendo de la aplicación.

La integridad, que se puede definir como la ausencia de alteraciones indebidas de la información, generaliza las definiciones más usuales, relacionadas solamente con la noción de acciones autorizadas (por ejemplo, prevención contra la modificación o borrado no autorizado de la información); naturalmente, cuando un sistema lleva a cabo una política de autorización, "indebido" engloba a "no autorizado".

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.

La seguridad no se ha introducido como un atributo de la confiabilidad, de acuerdo con la definición que da de ella una noción compuesta [CEC91], a saber: "la combinación de confidencialidad, prevención contra la revelación no autorizada de información, integridad, prevención contra el enmendado o borrado no autorizado de información, disponibilidad y prevención contra la retención no autorizada de información".

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), y la seguridad la prevención de lo que puede ser visto como una clase específica de fallos (la prevención de acceso y/o manipulación no autorizada de información). Por tanto, la fiabilidad y la disponibilidad están mucho más próximas entre sí que respecto a, por un lado, la inocuidad, y por otro, a la seguridad. Por ello, la fiabilidad y disponibilidad se pueden agrupar conjuntamente, pudiendo definirse globalmente por medio de la minimización de las interrupciones del servicio. Sin embargo, este comentario no debería conducir a pensar que la fiabilidad y la disponibilidad no dependen del entorno del sistema: desde hace mucho tiempo se reconoce que la fiabilidad/disponibilidad de un sistema informático está sumamente correlacionada con un perfil de utilización, sea por las averías debidas a fallos físicos o por las debidas a fallos de diseño.

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, (como por ejemplo inocuidad con disponibilidad) 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.

Un sistema no se avería generalmente siempre de la misma forma. Las formas según las que un sistema puede averiarse son sus modos de avería, que pueden describirse según cuatro puntos de vista: dominio de la avería, detectabilidad de las averías, congruencia de las averías y 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 la avería. Las situaciones de salidas congeladas o de silencio dan lugar, respectivamente a los sistemas pasivos tras la avería y a los sistemas con silencio tras la 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 pero no es detectado, se producen **averías no señaladas**.

El punto de vista de la percepción de la avería lleva a distinguir, en caso de varios usuarios, entre **averías consistentes** (en las que todos los usuarios del sistema tienen la misma percepción de las averías) y **averías inconsistentes** (en las que los usuarios del sistema pueden percibir de forma diferente una avería dada). Las averías inconsistentes son a menudo clasificadas como **averías bizantinas**. Hay que resaltar que las averías de un sistema con silencio tras la avería son consistentes, mientras que pueden no serlo en un sistema pasivo tras la avería.

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 asocia 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 en ausencia de avería y las consecuencias de las averías:

- **Averías menores**, donde las consecuencias son de un orden de magnitud igual que el beneficio obtenido por el servicio entregado en ausencia de averías.
- **Averías catastróficas**, donde 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 seguro tras la avería.

La figura 2.3 resume esquemáticamente las clases de averías.

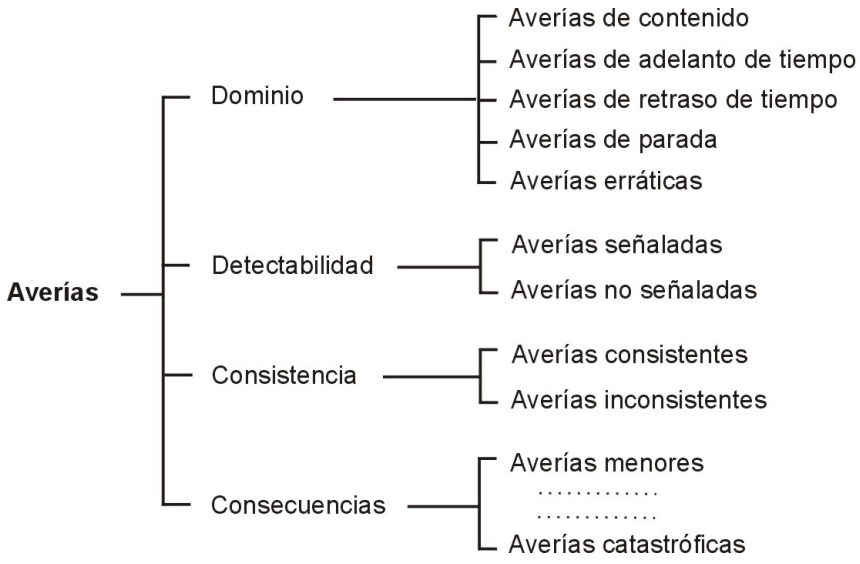


Figura 2.3: Clases de Averías [Avizienis04].

La noción de gravedad de las averías permite definir el concepto de **criticidad**: la criticidad de un sistema es la mayor gravedad de sus posibles modos de avería. La relación entre los modos de avería y la gravedad de las averías es muy dependiente de la aplicación considerada. Sin embargo, existe una amplia

clase de aplicaciones donde la inactividad se considera como una posición segura por naturaleza (por ejemplo, transportes terrestres o producción de energía), de donde se deduce la correspondencia directa que existe a menudo entre parada tras avería y seguridad en presencia de averías.

Los sistemas con parada tras avería (tanto los pasivos como con silencio tras avería) y los sistemas seguros tras avería son, sin embargo, ejemplos de sistemas controlados tras avería, es decir, sistemas diseñados e implementados con el fin de que se averíen solamente, o lo hagan en una medida aceptable, de acuerdo a modos restrictivos de avería; por ejemplo, que tengan las salidas congeladas en vez de entregar valores erráticos, que posean silencio tras la avería en lugar de enviar mensajes erráticos, o en los que aparezcan averías consistentes en vez de inconsistentes. Los sistemas controlados tras la avería pueden ser definidos, además, imponiendo alguna condición al estado interno o a la accesibilidad, al igual que los sistemas llamados **fail-stop**.

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 a veces muy difícil, si no imposible, construir un sistema sin alguna forma de redundancia.

b) De la actividad del sistema, ya que un dato con error 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 a dos o más usuarios, errores menores contra errores catastróficos.

2.4.3 Fallos

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

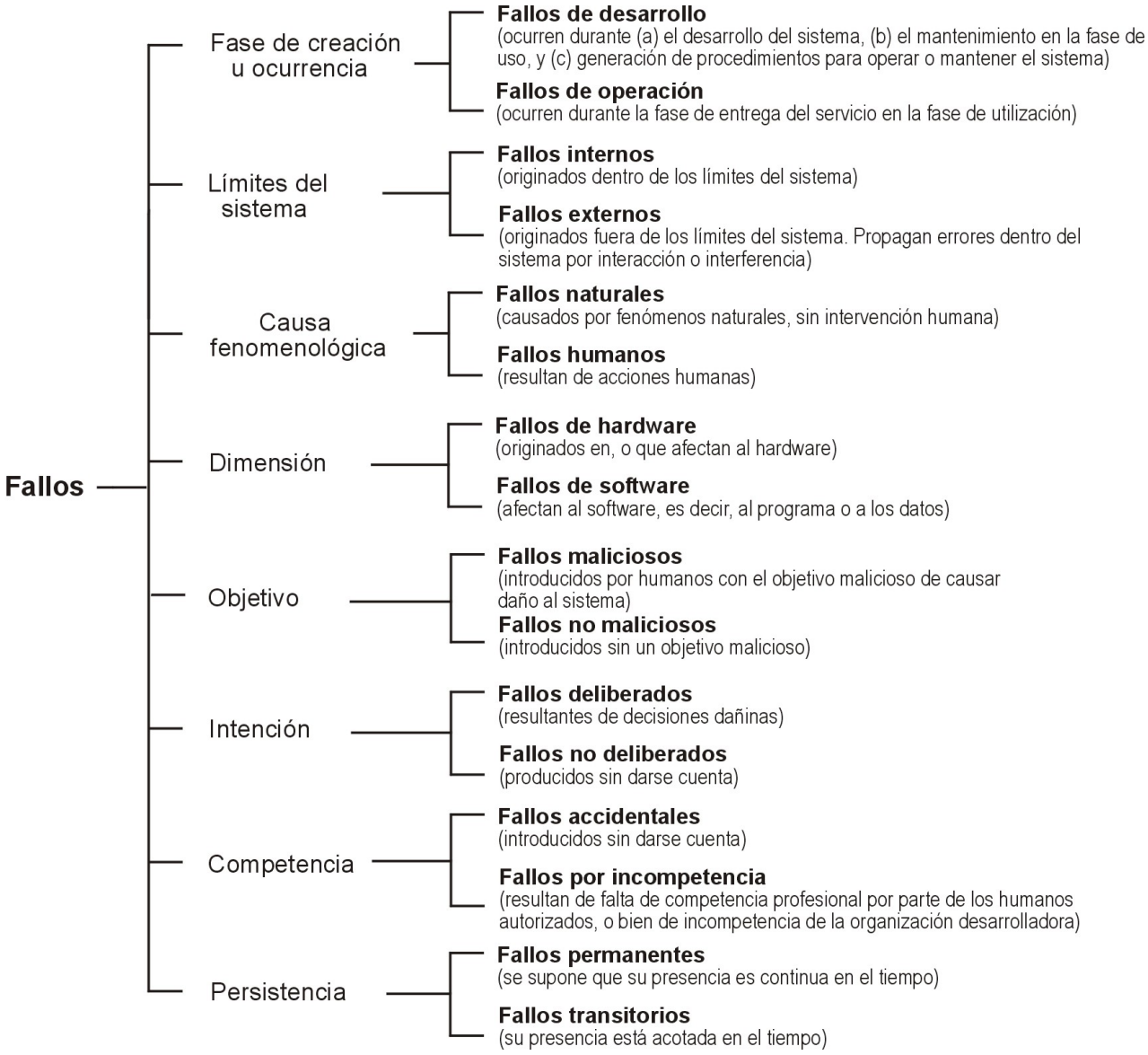


Figura 2.4: Clases de fallos elementales.

La fase de creación con respecto a la vida del sistema lleva a distinguir entre:

- **Fallos de desarrollo**, resultantes de imperfecciones originadas, bien en el desarrollo del sistema (de la especificación de las necesidades a la implementación) o durante las modificaciones posteriores, o bien durante el establecimiento de 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**, que son 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 nos llevan a distinguir entre:

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

Según la dimensión de los fallos se tiene:

- **Fallos de *hardware***: que son aquellos originados en el propio *hardware* o que afectan a éste.
- **Fallos de *software***: que afectan al *software*, es decir al programa 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 bien directamente durante su uso con el objetivo de causar algún daño al sistema.
- **Fallos no maliciosos**: cuando éstos 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**, resultantes de decisiones dañinas (con o sin intención maligna).

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 transitorios**, cuya presencia está ligada a aquellas condiciones y están, por tanto, presentes un tiempo limitado. Bajo esta denominación se incluyen tanto los fallos comúnmente denominados transitorios, que son originados por el entorno físico, y los fallos intermitentes, que son el resultado de la combinación de defectos internos con condiciones de operación internas o externas.

Si fuesen posibles todas las combinaciones de estas ocho clases de fallos, habría un total de 256 clases de fallos combinados. Sin embargo, todos los criterios no son aplicables a todas las clases de fallos; por ejemplo los fallos naturales no pueden clasificarse según su objetivo, intención o capacidad. Se han identificado 31 combinaciones probables, como puede verse en la figura 2.5. En el futuro se pueden identificar más combinaciones.

En primer lugar se pueden clasificar en tres grandes grupos:

- **Fallos de desarrollo**, que incluyen todas las clases de fallos ocurridos durante el desarrollo (combinaciones 1 a 11 en la figura 2.5).
- **Fallos físicos**, que agrupan todas las clases de fallos que afectan al *hardware* (combinaciones 6 a 23).
- **Fallos de interacción**, que contienen todos los fallos externos (combinaciones 14 a 31).

Se pueden definir más detalladamente nueve ejemplos concretos: defectos de *software*, bombas lógicas, erratas de *hardware*, defectos de producción, deterioro físico, interferencia física, intentos de intrusión, virus y gusanos, y entradas equivocadas [Avizienis04].

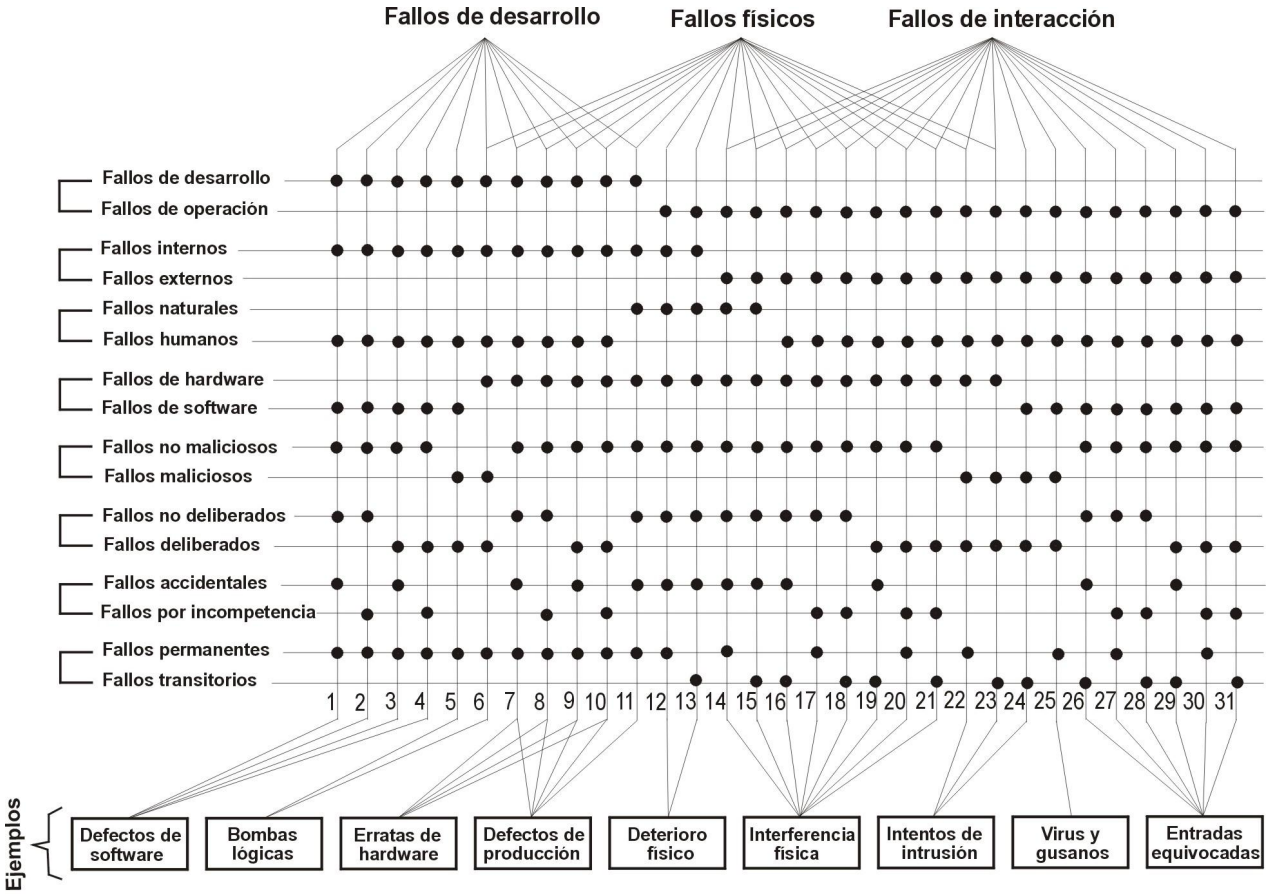


Figura 2.5: Clases de fallos combinados.

2.4.4 Patología de los fallos

Los mecanismos de creación y manifestación de los fallos, errores y averías se ilustran en la figura 2.6, y se pueden resumir como sigue:

1. Un fallo es **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 a B, propagándose el error dentro de B vía 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.

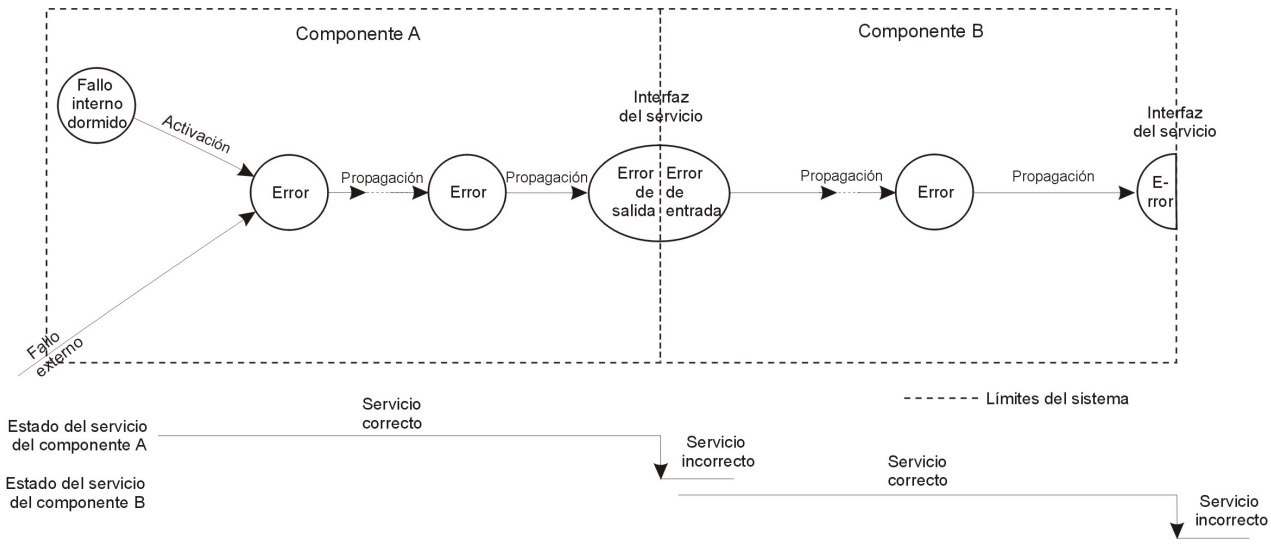


Figura 2.6: Propagación de los errores.

Estos mecanismos permiten completar la **cadena fundamental** de amenazas de la confiabilidad y la seguridad, mostrada en la figura 2.7:

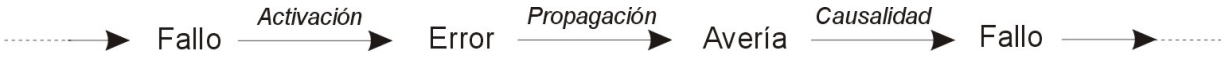


Figura 2.7: Cadena fundamental de amenazas de la confiabilidad y la seguridad.

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 generarse varios errores antes de que se produzca 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 la 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 puede suceder que estos últimos produzcan a veces 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

En este punto del capítulo se va a examinar qué son 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. La sección termina con una discusión sobre la relación entre los medios para la confiabilidad.

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**, en donde el estado erróneo se sustituye por otro ya sucedido antes de la ocurrencia del error. Este método incluye el establecimiento de puntos de recuperación, que son instantes durante la ejecución de un proceso donde se salvaguarda el estado, pudiendo éste posteriormente ser restaurado tras la ocurrencia del 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.

La figura 2.8 muestra un esquema resumido de las técnicas de tolerancia a fallos.

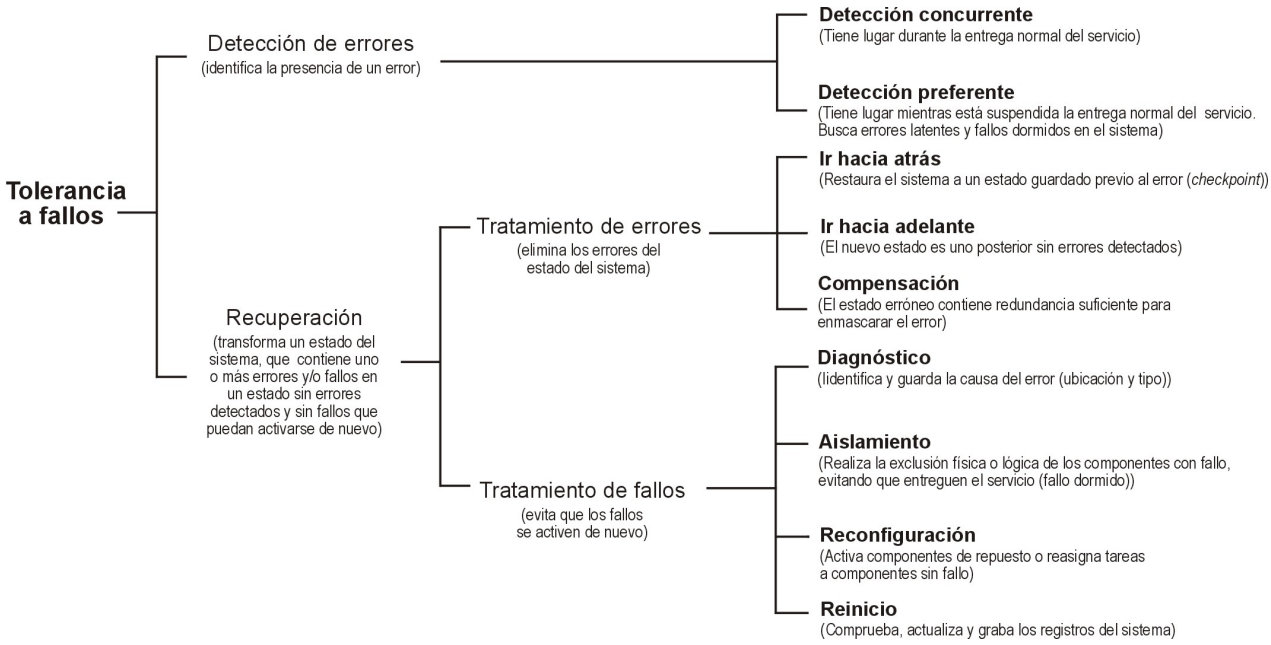


Figura 2.8: Técnicas de tolerancia a fallos.

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. Especialmente en 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 otra parte, 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, es decir, realizar una **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**. A primera vista, las nociones de fallos blandos y duros parecen sinónimas de las de fallo temporal y permanente respectivamente. Efectivamente, la tolerancia a fallos temporales 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 propagación del temporal. 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 alfa (debido 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.

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 previa activación 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 de los tests de verificación también llamados simplemente tests.

Todos estos enfoques se resumen en la figura 2.9.

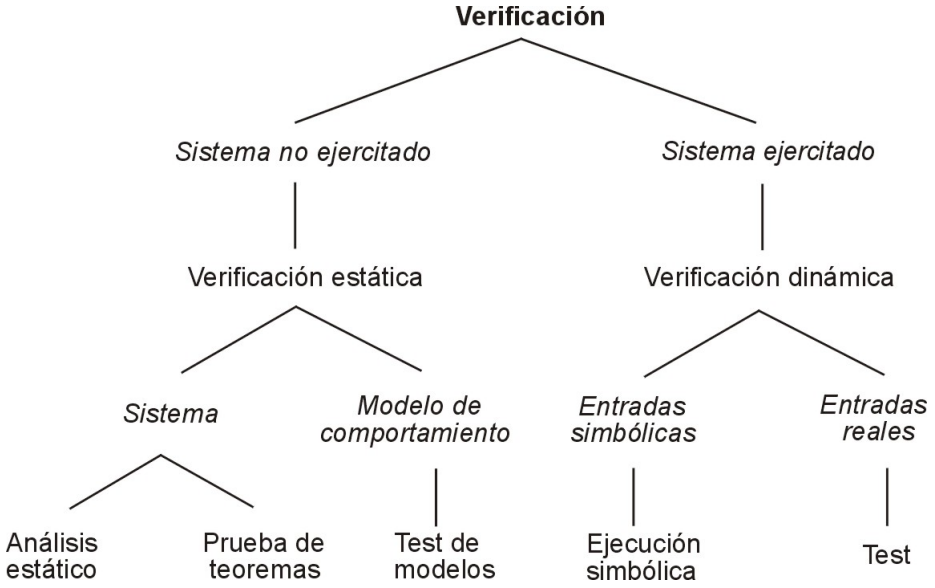


Figura 2.9: Enfoques de verificación.

El test exhaustivo de un sistema respecto de todas sus entradas posibles es generalmente impracticable. Los métodos para determinar los patrones particulares de test pueden clasificarse de acuerdo a dos puntos de vista: según los criterios de selección de las entradas de test, y según la generación de las entradas de test. En la figura 2.10 puede verse un resumen de los diferentes enfoques para la realización de tests, de acuerdo a los criterios de selección anteriores. En la parte superior de la figura se identifican los enfoques de test elementales. La parte inferior muestra las combinaciones de estos enfoques elementales, en donde se hace la distinción entre test *hardware* y *software*, ya que el test de *hardware* está orientado normalmente a los fallos de producción, y el de *software* a los fallos de desarrollo.

Es de especial importancia también, respecto a lo que el sistema no debe hacer, el verificar que el sistema no hace nada más de lo que está especificado. Este detalle está relacionado con la inocuidad y con la seguridad.

Se denomina **diseño orientado a la verificación** al diseño de sistemas de forma que sea fácil su verificación. Este planteamiento está especialmente desarrollado con respecto a los fallos físicos en el *hardware*, llamándose en este caso particular diseño para el test.

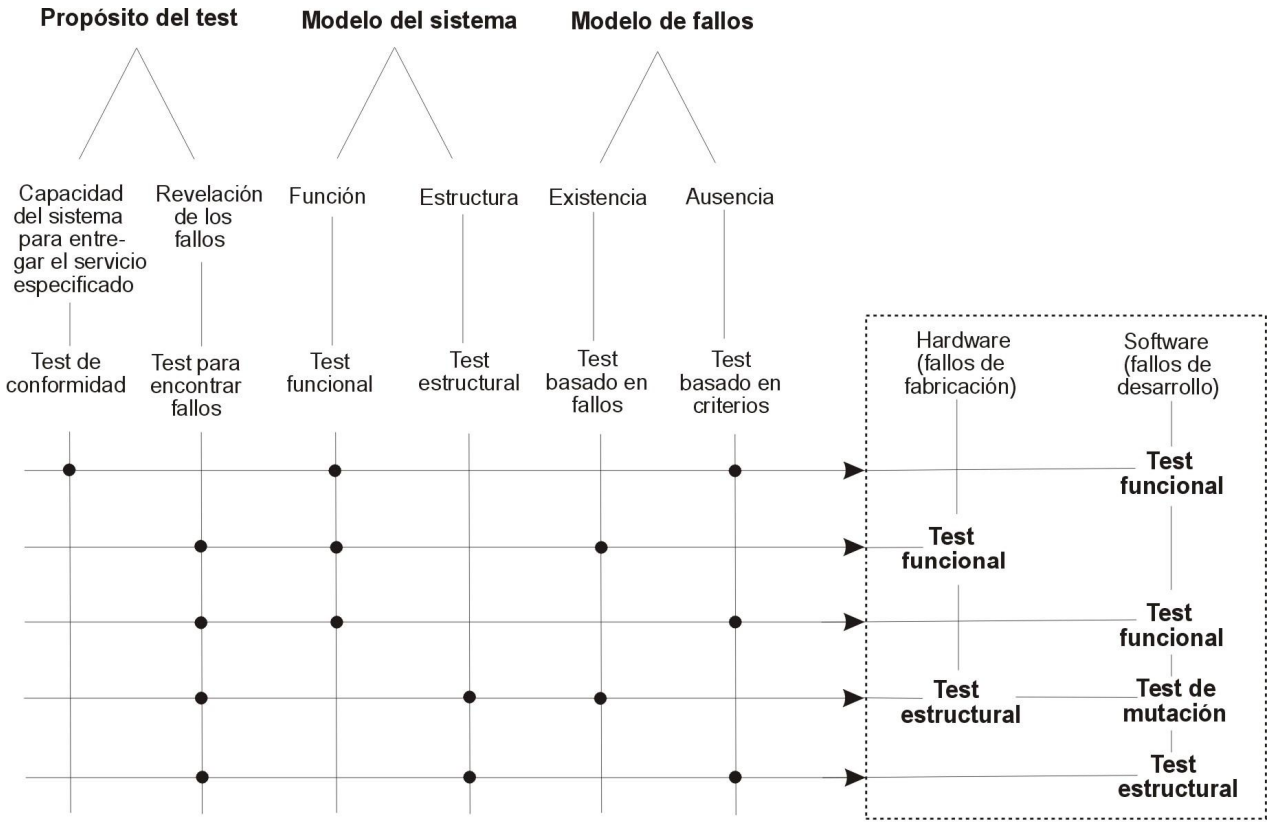


Figura 2.10: Enfoques de test según la selección de los patrones de test.

Se llama **mantenimiento correctivo** a la eliminación de fallos durante la vida operativa de un sistema. El mantenimiento correctivo puede ser de cuatro formas:

- **Mantenimiento curativo**, destinado a eliminar los fallos que han producido uno o más errores que se han detectado.
- **Mantenimiento preventivo**, destinado a eliminar los fallos antes de que provoquen errores. Estos fallos pueden ser:
  a) Fallos físicos que han aparecido después de las últimas acciones de mantenimiento preventivo.
  b) Fallos de diseño que han dado lugar a errores en otros sistemas similares.
- **Mantenimiento adaptativo**, destinado a ajustar el sistema a los cambios del entorno.
- **Mantenimiento aumentativo**, destinado a mejorar la función del sistema en respuesta a los cambios definidos por los clientes y los diseñadores, que pueden implicar la eliminación de fallos de especificación.

Estas definiciones se aplican tanto a los sistemas no tolerantes a fallos como a los tolerantes a fallos. Estos últimos se pueden mantener en línea (sin interrupción del servicio), o fuera de línea. Es importante notar, finalmente, que la frontera entre mantenimiento correctivo y tratamiento de los fallos es relativamente arbitraria. En particular, el mantenimiento curativo se puede considerar un medio de tolerancia a fallos.

Realmente, la mantenibilidad condiciona la confiabilidad del sistema a lo largo de todo su ciclo de vida, debido a las inevitables evoluciones durante su vida operativa. El término mantenimiento, tal como se ha utilizado aquí, sigue siendo el uso común, y no solamente incluye las reparaciones, sino también las modificaciones que tengan lugar en el sistema durante la fase de utilización de su ciclo de vida. En la figura 2.11 puede verse un resumen de las diversas formas de mantenimiento.

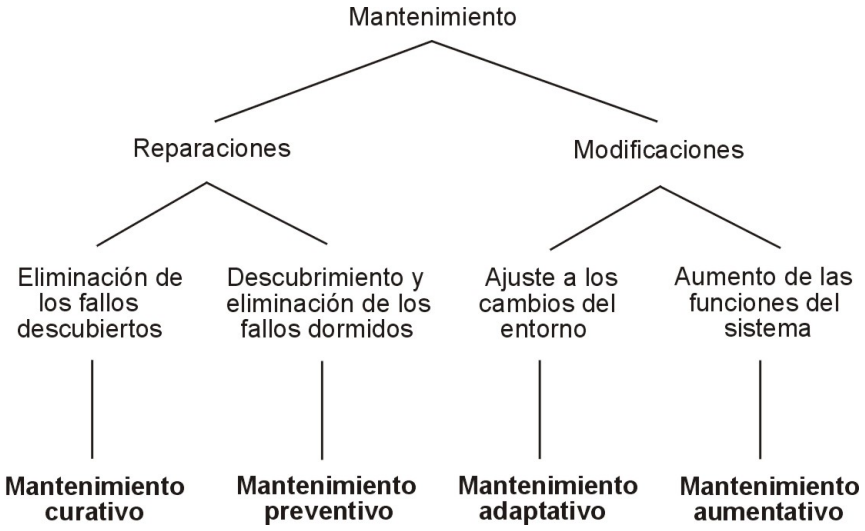


Figura 2.11: Formas distintas de mantenimiento.

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. La evaluación tiene dos facetas:

- 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.
- 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 aquel en donde el servicio entregado cumple con la función del sistema. **Servicio incorrecto** será aquel en donde el servicio entregado no cumple con la función del sistema.

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.

Habitualmente también se considera una tercera medida, la **mantenibilidad**, que se puede definir como la medida del tiempo de restauración tras la última avería, o de manera equivalente, de la entrega continua de un servicio incorrecto. La **inocuidad**, como medida, puede verse como una extensión de la fiabilidad. Si se agrupan el estado de servicio correcto con el estado de servicio incorrecto posterior a las averías benignas como un estado seguro (en el sentido de la ausencia de daños catastróficos, no de peligro), la inocuidad es entonces una medida de la continuidad del servicio inocuo, o bien del tiempo hasta la avería catastrófica. La inocuidad puede ser vista, pues, como la fiabilidad respecto a las averías catastróficas.

En caso de los sistemas con múltiples prestaciones, se pueden distinguir diversos servicios, así como diversas formas de entregar el servicio, desde la plena capacidad hasta la completa parada, que puede ser visto como entregas de servicio cada vez menos correctas. La medida combinada de prestaciones y confiabilidad se denomina habitualmente **prestabilidad**.

Los dos principales métodos de la 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 procesos elementales modelados (procesos de avería, de mantenimiento, de activación del sistema, etc.) que se pueden obtener mediante test.

Cuando se realiza una evaluación mediante modelado, los métodos a utilizar difieren significativamente según que el sistema se considere con fiabilidad estable o creciente. Estas últimas características se pueden definir de la siguiente forma:

- **Fiabilidad estable**: cuando se preserva la capacidad del sistema para entregar el servicio correcto (identidad estocástica de los tiempos sucesivos hasta la avería).
- **Fiabilidad creciente**: cuando se mejora la aptitud del sistema de entregar el servicio correcto (crecimiento estocástico de los tiempos sucesivos hasta la avería).

La evaluación de la confiabilidad en sistemas con fiabilidad estable se compone usualmente de dos fases. La primera de ellas es la fase de construcción del modelo del sistema a partir de procesos estocásticos elementales que modelan el comportamiento y las interacciones de los componentes del sistema. La segunda consiste en el procesamiento del modelo para la obtención de las expresiones y valores de las medidas de la confiabilidad del sistema.

La evaluación se puede realizar respecto a los fallos físicos, los fallos de diseño, o una combinación de ambos. La confiabilidad de un sistema es altamente dependiente de su entorno, tanto en el sentido amplio del término, como en el significado más restringido de su carga.

Los modelos de la fiabilidad creciente, sean relativos al *hardware*, al *software*, o al conjunto de los dos, están destinados a la realización de predicciones de la fiabilidad a partir de datos relativos a averías pasadas del sistema.

Cuando se evalúa 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**.

2.5.4 Dependencias entre los medios para alcanzar la confiabilidad

En las definiciones de prevención, tolerancia, eliminación y predicción de fallos de la sección 2.2, se utiliza la palabra "cómo" para especificar los objetivos que cada mecanismo pretende alcanzar. En la realidad, todos estos objetivos no son completamente alcanzables, debido a la imperfección de la naturaleza humana, que interviene en todas las actividades realizadas en aquellos mecanismos. Esto hace que la aplicación de uno de estos medios implique habitualmente la necesidad de aplicar otro u otros, por haberse introducido efectos secundarios en el sistema.

Por este motivo, para lograr un sistema de alta confiabilidad es necesaria la utilización combinada de varios de estos métodos. Las dependencias existentes entre los diferentes medios se pueden especificar así:

- A pesar de la prevención, en los diseños se generan fallos, por lo que es necesaria la eliminación de fallos: cuando se detecta un error durante la verificación, es necesario un diagnóstico para determinar sus causas y eliminarlas.
- La eliminación de fallos es imperfecta, como lo son los componentes del sistema (*hardware* y *software*). Por este motivo es necesaria la predicción de fallos.
- La importancia de los sistemas informáticos en la vida cotidiana conduce a establecer unos requisitos de tolerancia a fallos, basados en reglas constructivas. De nuevo, por la intervención humana, son necesarias la eliminación y la prevención de fallos.

Hay que hacer notar que el proceso es aún más recursivo, puesto que debido a la elevada complejidad de los sistemas actuales, son necesarias herramientas para su diseño y construcción. Para que el trabajo realizado con estas herramientas sea el esperado, éstas deben ser de alta confiabilidad, y así sucesivamente.

Los anteriores razonamientos ilustran la fuerte interacción existente entre la eliminación y la predicción de fallos. Por este motivo, ambas se incluyen en el término general **validación**.

La validación de un sistema puede ser de dos maneras:

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

2.6 Confiabilidad y tolerancia a fallos

En lo que concierne al desarrollo de sistemas de alta confiabilidad, el estado del arte consiste en efectuar una elección sistemática y equilibrada entre diferentes técnicas de tolerancia a fallos, con el fin de reforzar los métodos de prevención de fallos [Arlat90].

La prevención de fallos se centra en la utilización de componentes fiables, y pretende asegurar que el sistema desarrollado esté libre de fallos. A nivel de *hardware* son importantes la introducción de protecciones contra las perturbaciones del entorno y la utilización de componentes de alta escala de integración [Siewiorek82]. En lo que respecta al *software*, los métodos principales se concretan en una

concepción estructurada y modular, y en el empleo de lenguajes de alto nivel [Courtois92]. Debido a las limitaciones en la formalización y el control de la complejidad tecnológica actual, la tolerancia a fallos mantiene un papel preponderante en la búsqueda de un nivel significativo de confiabilidad. La introducción de la tolerancia a fallos en el proceso de desarrollo de un sistema informático hace necesario:

- Determinar los tipos de fallos susceptibles de activarse en la fase operativa.
- Diseñar el sistema de manera que utilice mecanismos de redundancia para reducir los efectos de dichos fallos sobre el servicio proporcionado en la fase operativa.

2.7 Confiabilidad y validación

La validación constituye uno de los principales problemas asociados al desarrollo y explotación de sistemas informáticos de alta confiabilidad. En efecto, además del aspecto funcional, debe ponerse el acento sobre la confianza en el comportamiento apropiado de los mecanismos que contribuyen a la confiabilidad, es decir, sobre la validación de la cobertura. Esto corresponde a una recursión del tipo validación de la validación: "¿cómo tener confianza en los métodos y mecanismos empleados para conseguir la confianza en el sistema?" [Laprie85].

La validación de la cobertura concierne principalmente a la validación del producto, es decir, del sistema desarrollado, y por tanto de los mecanismos de tolerancia a fallos integrados para asegurar la confiabilidad. Dos parámetros fundamentales permiten cuantificar la eficacia de estos mecanismos: el coeficiente de cobertura y la latencia de tratamiento. A título de ejemplo, para los mecanismos de detección, se pueden definir de la siguiente manera:

- El **coeficiente de cobertura de detección** es la probabilidad condicional de detección de errores.
- La **latencia de detección de error** es el intervalo de tiempo que separa la activación de un fallo en forma de error y su detección.

Es interesante resaltar, no obstante, la importancia de extender los métodos de validación a las diferentes fases de desarrollo (especificación, diseño e implementación), así como en la fase operativa. Como se indicó en el apartado 2.5, estos métodos pueden agruparse en dos grandes clases: la eliminación de fallos y la predicción de fallos.

La eliminación de fallos consiste en reducir (mediante verificación) la presencia de fallos y, en consecuencia, identificar las acciones más apropiadas para mejorar la concepción del sistema.

La predicción de fallos tiene por objetivo principal estimar (mediante evaluación) la influencia de la aparición, presencia y consecuencias de los fallos sobre el funcionamiento y la confiabilidad del sistema en fase operacional.

La separación entre la verificación y la evaluación es en realidad menos marcada de lo que en principio puede parecer, y su complementariedad es un hecho en numerosas ocasiones.

2.8 Tolerancia a fallos y validación experimental

En los apartados anteriores se han enumerado diferentes aspectos de los Sistemas Tolerantes a Fallos (STF) reales, como la obtención de los coeficientes (o factores) de cobertura y los tiempos de latencia en la detección y en la recuperación de errores, que indican que su validación precisa de una parte experimental. Este hecho viene motivado por 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 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.
  - 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).

2.9 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 real, 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 difíciles las observaciones en experimentos no controlados.
- La disparidad de las soluciones adoptadas para aumentar la confiabilidad en los STF, lo 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]:

> "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."

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 2.12 [Arlat90, DBench01, PGil92] 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 de manera 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.

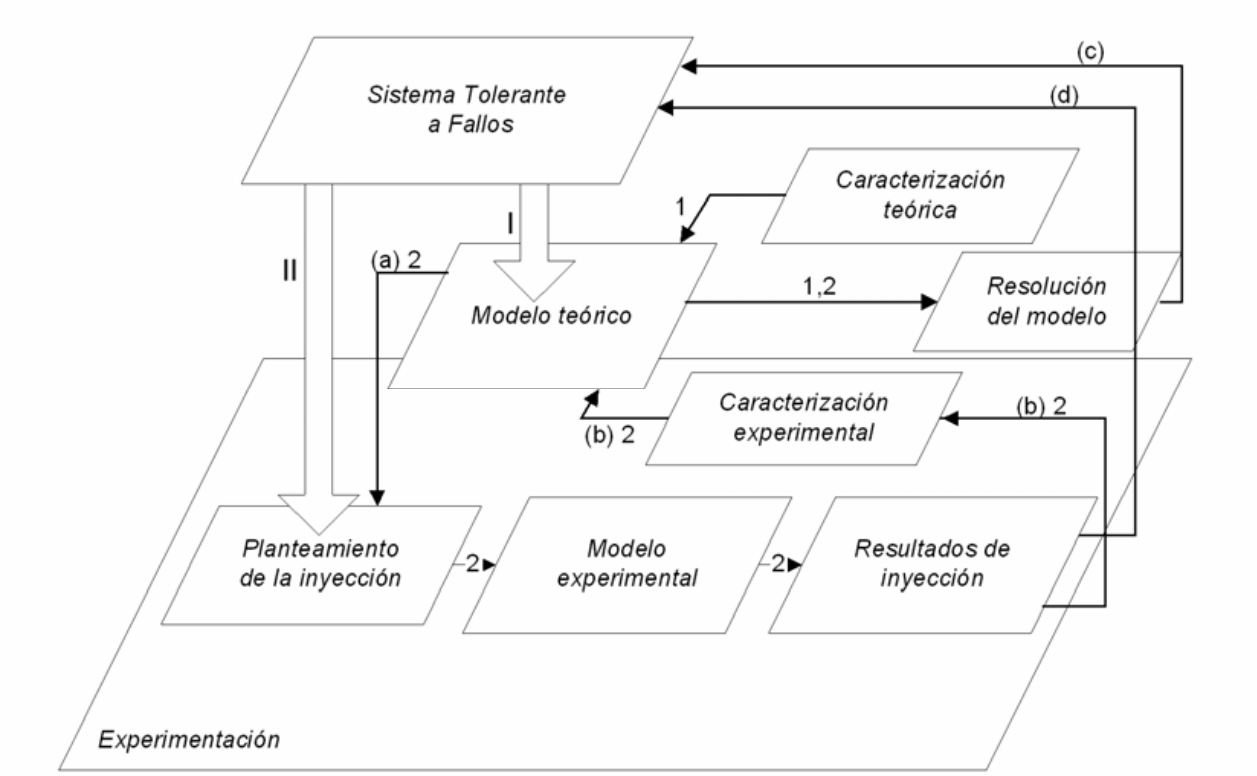


Figura 2.12: Diagrama de bloques del proceso de validación teórica y experimental mediante inyección de fallos.

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).

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. 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 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. Se puede realizar mediante inyección de fallos, esto es, introduciendo deliberadamente fallos en un modelo experimental o prototipo del sistema.

3

Técnicas de inyección de fallos

3.1 Introducción

Vista la utilidad de la inyección de fallos como herramienta de validación de la confiabilidad, a continuación se van a revisar cuáles son las técnicas de inyección de fallos que se han utilizado hasta la fecha.

La clasificación más aceptada de las técnicas de inyección de fallos [Hsueh97, Yu01] las divide en tres grandes grupos, como se puede ver en la figura 3.1:

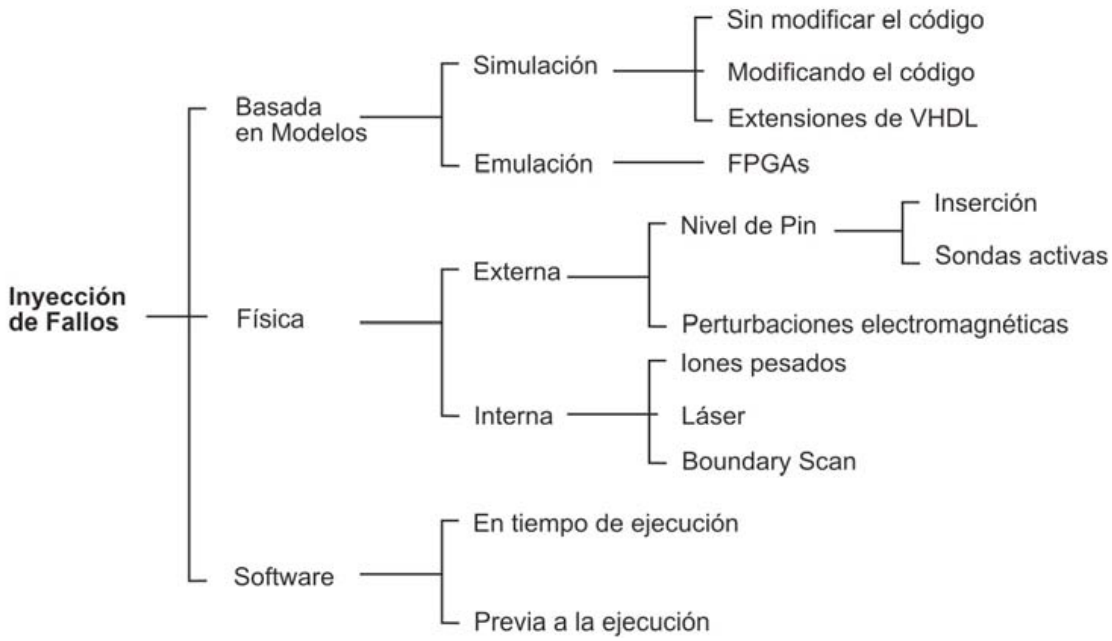


Figura 3.1: Clasificación de las técnicas de inyección de fallos.

- La inyección de fallos basada en modelos utilizando para ello la simulación, también denominada SBFI (del inglés *Simulation-Based Fault Injection*), se aplica sobre un modelo simulado del sistema. Generalmente este modelo está realizado en un lenguaje de descripción del *hardware* del que se simula su comportamiento. O mediante la emulación, donde a partir de un prototipo *hardware* del sistema bajo estudio, se puede realizar una validación de su comportamiento en fases tempranas de diseño del mismo.
- 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 ejecución.

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 las 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.

3.2 Inyección física de fallos

Con este tipo de técnica se pretende introducir en el sistema fallos de forma tan parecida a la realidad como sea posible. Todas 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 tipos de técnicas de inyección de fallos: internas y externas. En las técnicas de inyección externa los fallos se inyectan fuera de los componentes a validar, por ejemplo, en los pines de los circuitos integrados. En las técnicas internas los fallos se inyectan dentro del componente a validar, por ejemplo mediante láser, radiaciones de iones pesados o con mecanismos especiales integrados en el *hardware* del sistema.

3.2.1 Inyección física externa

Hasta la fecha se han utilizado dos técnicas de inyección externa de fallos: inyección a nivel de pin de circuito integrado [Arlat90, Madeira94, PGil97] e inyección mediante perturbaciones electromagnéticas [Damm88, Karlsson95].

Inyección física a nivel de pin

Para llevar a cabo la inyección física de fallos a nivel de pin se utiliza una herramienta especial que modifica los valores lógicos en los pines de los circuitos integrados o en las líneas de comunicación y control. Estas modificaciones de los valores lógicos deben ser representativas de los fallos que aparecen en

la fase operacional de un sistema. Los modelos de fallos físicos más utilizados son el *stuck-at* (pegado-a), *bridging* (puente) y *open-line* (línea abierta). Los fallos pueden ser permanentes, transitorios o intermitentes.

Existen dos tipos de técnicas de inyección física a nivel de pin:

- Técnica de **sondas activas**: cuando se utiliza esta técnica el fallo se inyecta directamente en los terminales de los circuitos integrados, conectores o pistas de circuito impreso, sin desconectar ningún componente. La sonda del inyector de fallos fuerza un cero o uno lógicos en los puntos elegidos.
- Técnica de **inserción**: se coloca un dispositivo especial en el lugar del componente donde se va a inyectar, o entre el mismo y su soporte (normalmente un zócalo o conector). Este dispositivo intermedio es el encargado de inyectar los fallos. Antes de inyectar el fallo se corta la conexión entre el circuito y el sistema. De esta forma la inyección se realiza sobre una línea en alta impedancia y, por lo tanto, al no haber forzado de ninguna señal no existe la posibilidad de dañar el componente sobre el que se inyecta.

Algunas herramientas de inyección física a nivel de pin son MESSALINE [Arlat90], RIFLE [Madeira94] y AFIT [PGil97].

Inyección física por interferencias electromagnéticas

El trabajo más importante es el realizado en la Technical University of Vienna (Austria) [Damm88, Karlsson95]. Su sistema de inyección de fallos utiliza los equipos de test sobre inmunidad electromagnética según el estándar EN 61000-4-4. Este equipo genera ráfagas de interferencias con las siguientes características temporales: duración 15ms, frecuencias de 1.25, 2.5, 5 ó 10 kHz, y tensiones que se pueden elegir desde 225V hasta 4400V. Este tipo de pruebas se realizan normalmente para verificar el correcto funcionamiento de un equipo completo ante interferencias electromagnéticas. En este caso se eliminan los mecanismos de protección electromagnética para aumentar la efectividad de los mismos. Los fallos resultan de las interferencias que se localizan en las pistas del circuito impreso y entran en los circuitos integrados del sistema a validar. El modelo físico de estos fallos es la perturbación electromagnética.

3.2.2 Inyección física interna

Se denomina inyección física interna a la que utiliza mecanismos físicos para introducir el fallo directamente en el interior de los componentes. Actualmente se utilizan tres técnicas de inyección física interna: mediante iones pesados, mediante técnicas láser y mediante *Boundary Scan*.

Inyección física por radiación de iones pesados

Esta técnica se desarrolló en la Chalmers University of Technology, en Göteborg (Suecia), [Gunneflo89, Karlsson95]. En esta técnica se generan fallos del tipo *bit-flip* (inversión) dentro del componente irradiado. Se utiliza una fuente radiactiva de Californio 252. En este caso lo que se intenta es simular fallos transitorios producidos por fuentes externas, como por ejemplo las interferencias radiactivas, eléctricas o electromagnéticas. Para conseguirlo se debe eliminar la tapa del encapsulado del componente a irradiar e introducir todo el conjunto (componente y fuente radiactiva) dentro de una cámara de vacío dado que las partículas del aire podrían modificar la trayectoria e incluso parar los iones.

La característica más interesante de esta técnica es la posibilidad de introducir perturbaciones en toda la superficie de silicio del circuito integrado. De esta forma se pueden inyectar fallos en zonas del chip inaccesibles para otras técnicas.

Inyección física por radiación láser

En esta técnica [Sampson98] se inyectan fallos transitorios en puntos muy bien controlados dentro de un circuito integrado sin encapsulado. La inyección se efectúa apuntando con un haz láser sobre la superficie del silicio. Este láser genera pares electrón-hueco en el semiconductor de la misma manera que los iones pesados utilizados en la técnica anterior.

Inyección mediante Boundary Scan

Esta técnica se encuentra a caballo entre las internas y las externas, ya que si bien el mecanismo que se utiliza para inyectar el fallo es interno al sistema, el fallo que se emula es una alteración del estado lógico de los pines del circuito integrado, como en las técnicas externas.

En la bibliografía se le denomina inyección basada en *Boundary Scan* o inyección basada en cadenas de exploración (*scan-chain*). Se basa en la utilización del estándar conocido con los nombres JETAG, JTAG, Boundary Scan o IEEE 1149.1 para acceder a los registros denominados BSC (*Boundary Scan Cells*). Estos registros están situados en cada entrada o salida digital del circuito integrado. En funcionamiento normal son los encargados de transferir los niveles lógicos entre el circuito y los pines, y en modo test se pueden reprogramar para transferir otros datos diferentes. En esta técnica se utiliza este modo de test para modificar artificialmente estos niveles emulando un fallo externo.

Esta técnica se desarrolló en la universidad de Chalmers [Folkesson98] con la herramienta FIMBUL (*Fault Injection and Monitoring using BUilt in Logic*), integrada más tarde en GOOFI (*Generic Object-Oriented Fault Injector*) [Aidemark01, Aidemark03]. En [Santos03] el autor estudia esta técnica para integrarla en Xception [Carreira95], que actualmente es una herramienta de inyección de fallos basada en *software*. En ese mismo trabajo se proponen cambios en el estándar *Boundary Scan* para mejorar su pobre comportamiento temporal de cara a mejorar la inyección.

3.2.3 Ventajas e inconvenientes

Las ventajas generales de la inyección de fallos implementada mediante *hardware* son [Yu03]:

- Es muy adecuada cuando se necesita gran precisión temporal, tanto en el disparo de la inyección como en la monitorización del sistema.
- La evaluación experimental mediante la inyección en el *hardware* es, en muchos casos, la única forma de calcular con precisión las coberturas y latencias del sistema.
- Esta técnica es la que mejor se adapta a los modelos de fallos en bajo nivel.
- Los experimentos de inyección son muy rápidos, y además, se pueden ejecutar casi en tiempo real, permitiendo la posibilidad de ejecutar un gran número de experimentos de inyección.

- La ejecución de experimentos de inyección de fallos en el *hardware* real, el cual está ejecutando el *software* real, permite incluir cualquier fallo de diseño que pueda estar presente en el diseño real del *hardware* y del *software*.
- Los experimentos de inyección de fallos se realizan utilizando el mismo *software* que se ejecutará en la aplicación real.
- No es necesario desarrollar y validar un modelo del sistema.

Los inconvenientes generales de la inyección de fallos implementada mediante *hardware* son:

- Existe un gran riesgo de dañar el sistema bajo prueba.
- Los altos niveles de integración, los sistemas SOC (del inglés *System On a Chip*) y las tecnologías con gran densidad de integración limitan la accesibilidad de la inyección.
- Algunos métodos *hardware* de inyección de fallos necesitan parar el procesador para inyectar el fallo y volver a ponerlo en marcha, lo que no es efectivo a la hora de medir latencias en sistemas físicos.
- Presenta una baja portabilidad entre diferentes sistemas.
- El conjunto de puntos de inyección, así como el de fallos inyectables, están limitados.
- El tiempo de configuración para cada experimento puede contrarrestar el tiempo ganado al realizar los experimentos en tiempo real o cercano al tiempo real.
- Se necesita *hardware* específico para realizar los experimentos de inyección de fallos.
- Observabilidad y controlabilidad limitadas.

3.3 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]. Sin embargo, cada vez hay más trabajos que también apuntan la posibilidad de 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*, como memoria, registros, periféricos e incluso los controladores de los diferentes relojes del sistema. Por otro lado, tiene la limitación de algunos registros internos a los que no se tiene acceso [DBench02, Iyer95].

Los modelos físicos de fallo son el *bit-flip* y el *stuck-at*. A nivel lógico puede ser corrupción de información almacenada (al inyectar sobre registros o memoria), corrupción de comunicaciones (al inyectar sobre los mensajes) y corrupción de 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.

Respecto a la portabilidad de las herramientas, un inyector puede ser un conjunto de rutinas que se ejecutan junto con la aplicación [Kanawati95, Han95, Rodríguez99], aunque también existen herramientas de inyección de fallos *software* que no ejecutan código en el sistema bajo estudio, sino que modifican el código previamente a la ejecución y observan [Kao93, Han95]. En ambos casos, una herramienta de

inyección de fallos por *software* suele estar diseñada específicamente para un sistema concreto, y su utilización se restringe al mismo.

Precisamente por este mismo motivo, la propia naturaleza de esta técnica hace que sea muy factible su implementación desde un punto de vista económico, y ha propiciado que hayan aparecido muchas herramientas basadas en inyección de fallos por *software*. Las más significativas 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] e INERTE [Yuste03].

3.3.1 Clasificación

[Folkesson99] presenta una clasificación de los métodos que se siguen para realizar la inyección mediante SWIFI:

- Previo a la ejecución (del inglés *pre-runtime fault injection*). Este tipo de técnica tiene la ventaja de minimizar la intrusión en el sistema. Consiste en modificar la carga del mismo antes de su ejecución. A su vez, existen dos variantes:
  - Modificación en tiempo de compilación, que consiste en modificar el código fuente del programa que se desea alterar, introduciendo fallos en el sistema para que se comporte de manera diferente. Esta técnica tiene el inconveniente de requerir el código fuente de la carga que ejecutará el sistema bajo estudio.
  - Modificación del código ejecutable. En este caso, se modifican las zonas de datos y/o código de la carga del sistema antes de su ejecución.
- En tiempo de ejecución (del inglés *run-time fault injection*). En este caso, la inyección se realiza por parte de otro(s) proceso(s)/programa(s) que se ejecuta(n) en el sistema, independientemente de la carga "real". Estos procesos tienen como misión modificar el entorno del programa, es decir, las zonas de programa y de datos de la memoria y los registros del procesador (evidentemente, los que son accesibles por *software*).

3.3.2 Ventajas e inconvenientes

Las ventajas generales de la inyección de fallos implementada mediante *software* son [Yu03]:

- Esta técnica puede ser aplicada tanto a aplicaciones como a sistemas operativos.
- Los experimentos de inyección se pueden ejecutar casi en tiempo real, permitiendo la posibilidad de ejecutar un gran número.
- La ejecución de los experimentos de inyección en el *hardware* real, el cual está ejecutando el *software* real, tiene la ventaja de incluir cualquier fallo de diseño que pueda estar presente en el diseño real del *hardware* y del *software*.
- No se requiere ningún *hardware* específico.
- El coste de implementación es bajo.

- No es necesario desarrollar y validar un modelo del sistema.

Los inconvenientes generales de la inyección de fallos implementada mediante *software* son:

- Conjunto limitado de instantes de inyección.
- No es posible inyectar fallos en localizaciones inaccesibles al *software*.
- Requiere la modificación del código fuente para soportar la inyección de fallos, lo que significa que el código que se ejecuta durante los experimentos de inyección no es el mismo que se ejecutará durante el funcionamiento real del sistema.
- Observabilidad y controlabilidad limitadas, ya que sólo se puede acceder a aquellas partes del sistema que sean accesibles por *software*.
- Es muy difícil modelar fallos permanentes.
- La ejecución del *software* inyector de fallos podría afectar a la planificación de las tareas del sistema, de tal manera que podría causar incumplimientos de plazos de trabajo.

3.4 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 sistema puede ser un circuito integrado, una unidad funcional o un subsistema del sistema completo. 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. La inyección se realiza mediante la alteración de los valores lógicos de los elementos del modelo durante la simulación. En el caso de la emulación, se utiliza como instrumento de simulación del sistema un soporte físico.

Las técnicas actuales de desarrollo de sistemas digitales utilizan de forma extensiva la simulación durante las primeras fases de diseño del mismo, siendo posible utilizarla para efectuar un primer análisis de las Prestaciones y la Confiabilidad. Este primer análisis permite ahorrar tiempo y dinero en el caso de detectar errores en el sistema. En EDA (*Electronic Design Automation*) es célebre la "regla de los diez" (en inglés *rule-of-tens*), que determina que el coste de encontrar y corregir defectos se multiplica por 10 en cada etapa del proceso [Kilty95].

En comparación con los modelos analíticos, la simulación presenta la capacidad de modelar sistemas complejos con un alto grado de fiabilidad, sin tener que realizar ningún tipo de restricción para conseguir que el modelo analítico sea matemáticamente tratable. Otra ventaja de la inyección de fallos mediante simulación respecto de otras técnicas de inyección es la gran observabilidad y controlabilidad de todos los componentes modelados.

La figura 3.1 muestra dos grupos de técnicas de inyección de fallos basadas en modelos: la simulación de esquemas eléctricos y de modelos en lenguajes de algún tipo (HDL, C, C++, ADA, etc.) son técnicas basadas en una simulación por *software* de un modelo, mientras que la inyección de fallos mediante emulación utiliza como instrumento de simulación del sistema un soporte físico, circuitos lógicos programables (PLD, del inglés *Programmable Logic Devices*) del tipo FPGA (*Field Programmable Gate Array*).

3.4.1 Inyección de fallos basada en simulación *software*

En el primer grupo de técnicas, la inyección de fallos que se realiza para el análisis de la Confiabilidad depende de los diferentes niveles de abstracción definidos. Por ejemplo, en [Pradhan96] se consideran tres niveles:

- Eléctrico: se corresponde con los circuitos eléctricos, inyectándose fallos que consisten en alteraciones de corriente y/o voltaje.
- Lógico: se corresponde con las puertas lógicas, y se inyectan fallos de los tipos *stuck-at* ('0' y '1'), *bit-flip, delay, open-line, indetermination, pulse*, etc.
- Funcional: correspondiente con bloques funcionales, los fallos inyectados representan cambios en los registros de la CPU, *bit-flip* en la memoria, *delay* o *pulse* en líneas de interconexión, etc.

Sin embargo, estos niveles varían según diferentes autores. En [Jenn94a] se especifican otros niveles: tecnológico, de circuito, lógico, de transferencia de registros (RT, del inglés *Register Transfer*) y no interpretado, mientras que [Siewiorek94] considera también tres niveles: de circuito, lógico y de sistema. En [DBench02] se identifican ocho niveles: físico (o de dispositivo), lógico, de transferencia de registros, algorítmico, *kernel*, *middleware*, aplicación y operación. Es decir, el establecimiento de los niveles de abstracción presenta bastante diversidad, y en ocasiones se trata de una cuestión de terminología, estando a menudo los diferentes niveles de abstracción solapados en las diferentes clasificaciones [DGil99].

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 la simulación. Por ejemplo, la simulación al nivel eléctrico no se puede usar de manera efectiva para estudiar sistemas VLSI complejos, y la simulación al nivel lógico no permite el estudio de sistemas computadores grandes. La simulación a nivel de sistema posibilita el análisis de las características de los computadores grandes y las redes. La desventaja obvia de aumentar el nivel de abstracción es que los modelos (tanto del sistema bajo estudio como de los fallos que se inyectan) se alejan de los mecanismos físicos reales.

Otro factor importante a tener en cuenta son las características temporales de los fallos. En [Iyer86, Siewiorek94] se demostraba que más del 80% de las averías de los sistemas computadores eran debidas a fallos transitorios. Estos fallos tienen diferentes causas físicas, como transitorios en la alimentación, diafonía (en inglés *crosstalk*) capacitiva o inductiva, o radiación cósmica. Pero ahora ya no es así, con la importancia creciente de los fallos intermitentes, como se ha introducido en el primer capítulo de este trabajo, y que se ampliará en el cuarto capítulo.

Existen algunos problemas comunes 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 dado, 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.

A continuación, tras comentar las principales ventajas e inconvenientes de esta técnica [Yu03], se presenta un análisis de diferentes estudios y herramientas de inyección de fallos mediante simulación, agrupadas por diferentes niveles de abstracción.

Las ventajas de la inyección de fallos basada en simulación *software* son:

- Puede soportar todos los niveles de abstracción.
- No se presenta ningún tipo de intrusión en el sistema real.
- Control total de los modelos de fallos y de los mecanismos de inyección.
- Los cambios en el sistema son muy sencillos.
- Coste bajo en la automatización.
- No requiere ningún *hardware* específico.
- Proporciona a los diseñadores del sistema una realimentación temprana.
- Máxima controlabilidad y observabilidad.
- Validación del sistema durante la fase de diseño.
- Capaz de inyectar fallos transitorios, permanentes e intermitentes.
- El coste temporal necesario para la inyección del fallo es nulo.

Los inconvenientes de la inyección de fallos basada en simulación *software* son:

- Es necesario un gran esfuerzo para desarrollar el modelo.
- Gran consumo temporal durante las simulaciones.
- Los modelos no están siempre disponibles.
- Los resultados dependen de la precisión del modelo.
- Los modelos pueden no incluir todos los fallos de diseño que pueden estar presentes en el *hardware* real.
- Los resultados dependen tanto de la bondad del modelo del sistema como de la representatividad de los modelos de fallos.

Nivel tecnológico

Los trabajos de [Choi93] presentan un modelado de los procesos de electromigración y de ruptura de la capa de óxido en los circuitos VLSI. La integración del proceso tecnológico de electromigración se realiza en dos fases principales:

1. Se simula el circuito, almacenando las informaciones concernientes a las conmutaciones que han tenido lugar.
2. Las informaciones obtenidas son utilizadas por el modelo de electromigración para evaluar las degradaciones de las conexiones y, en definitiva, la existencia de conexiones averiadas.

Para modelar la degradación de la capa de óxido se emplea un proceso similar.

Nivel de transistor

Dependiendo del autor, este nivel se denomina de diferentes formas: circuito, eléctrico o de dispositivo [Jenn94a, Siewiorek94]. La inyección de fallos a este nivel se puede usar para estudiar el impacto de las causas físicas que producen los fallos y los errores. A pesar de que los modelos de fallos más establecidos son el *stuck-at* y el *bit-flip*, utilizando este nivel algunos estudios muestran que estos modelos de fallos no son siempre representativos de los fallos físicos [Galiay80, Shen85, DBench02, Gracia02]. Además, en algunos circuitos es necesario el uso de simuladores de fallos que manejen fallos eléctricos transitorios y fallos físicos permanentes si éstos contienen elementos analógicos y digitales que no puedan ser caracterizados completamente por los modelos de fallos de tipo lógico.

Sin embargo, bajar a este nivel de detalle implica una serie de restricciones desde el punto de vista de la simulación, de manera que es difícil estudiar el comportamiento de más de un circuito integrado. Además, el tiempo de simulación necesario restringe el número de componentes que se pueden simular.

FOCUS [Choi92] modela el impacto de la penetración de una partícula ionizada mediante la inserción de una fuente de corriente a nivel de circuito. El modelo, escrito en el lenguaje SPLICE3, permite el estudio de la propagación de los errores originados a través de diferentes niveles de abstracción: eléctrico, lógico y algorítmico.

Nivel lógico

También se denomina nivel de puerta (en inglés *gate-level*) [Siewiorek94]. Los fallos y las funciones de los circuitos son simulados mediante modelos lógicos abstractos. La simulación en este nivel realiza operaciones binarias que representan el comportamiento de un dispositivo dado, obteniéndose salidas con valores discretos, así como también información temporal aproximada. Habitualmente se comparan las salidas del sistema con y sin fallos para determinar la ocurrencia de los fallos y de los errores.

Los primeros trabajos de simulación en este nivel se basan en la inyección de fallos *stuck-at* de tipo permanente. Así, en [Lomelino86] este tipo de inyección se aplica sobre una descripción esquemática basada en puertas y biestables del procesador *bit-slice* AMD 2901, utilizando un simulador desarrollado por la NASA. En [Czeck90] se lleva a cabo sobre las señales de un modelo del procesador IBM RT PC realizado en Verilog, otro lenguaje de descripción de *hardware* (HDL) cuyo uso también está muy extendido.

En [Cha93, Cha96] se describe una herramienta que hace referencia a una serie de elementos que permiten una simulación eficaz de fallos al nivel lógico, representativos de los fallos transitorios materiales (impacto de iones). En una primera fase se obtiene un modelo lógico de los fallos mediante simulación a nivel de circuito para poder utilizar, en una segunda fase, un simulador de fallos temporal. En el momento en que los errores son almacenados en un registro del sistema, se efectúa una tercera y última fase de simulación de fallos paralela con "retardo-cero".

En [Choi93] se presenta otro método para generar modelos de fallos reales al nivel lógico, denominado aproximación del diccionario de fallos (en inglés *fault-behavior dictionary approach*). En primer lugar se simula al nivel eléctrico la zona donde se inyectan los fallos (por ejemplo con PSPICE). Mientras se efectúa la inyección, sobre el subcircuito (formado por los puntos que rodean el lugar de inyección) se aplican todas las combinaciones de entrada, analizando el comportamiento de las salidas del subcircuito, y almacenándolo en un diccionario. La entrada del diccionario consiste en el vector de entradas y el lugar, instante y duración de la inyección. Después se realiza la simulación al nivel lógico a partir del modelo de fallos del diccionario. La idea es atractiva porque podría extenderse para generar fallos en otros niveles de mayor abstracción.

Nivel de transferencia entre registros

En este nivel, denominado también RTL (del inglés *Register-Transfer Level*), la simulación se encamina hacia la validación de los distintos mecanismos *hardware* que incorporan los procesadores con respecto a la alteración de alguno de los registros internos. Por ejemplo, en [Ohlsson92, Rimén92, Rimén93] se analiza el comportamiento ante fallos de un modelo en VHDL de un procesador de 32 bits con *watchdog* interno. El modelo de fallo adoptado en este estudio es el *bit-flip*, mientras que en [Karlsson90] se realiza una comparación de la técnica de inyección de iones pesados con la inyección simulada.

En [Vargas99, Vargas00a, Vargas00b] se añaden diferentes técnicas de codificación al modelo de circuito bajo prueba para aumentar la fiabilidad del mismo, mediante una herramienta CAD que denominan **FT-PRO**. Esta herramienta genera elementos de memoria tolerantes a fallos transitorios, en especial tolerantes a SEU (del inglés *Single Event Upset*). Se centran en este tipo de fallo debido a que son los más comunes en aplicaciones espaciales y de aviónica, y hoy en día son comunes incluso a nivel del mar debido al uso de tecnología submicrométrica. Una vez generado el circuito con esta herramienta, inyectan fallos modificando el código, creando una especie de mutante que inyecta el fallo (en este caso *bit-flips*) según un MTBF (*Mean Time Between Failures*, tiempo medio entre averías) determinado previamente.

Nivel de sistema

La simulación de fallos a este nivel se utiliza para estudiar sistemas computadores completos así como redes de computadores, en lugar de sus componentes individuales. Estos estudios habitualmente consideran como un todo el *hardware*, el *software*, sus interacciones y la interdependencia entre los diversos componentes del sistema. Sin embargo, existen algunos problemas en el desarrollo de modelos de simulación a este nivel, como pueden ser el esfuerzo y el tiempo requeridos para desarrollar un modelo de simulación, la dificultad para establecer los modelos de fallos adecuados, la existencia de un amplio espectro de componentes, el impacto del *software* en la Confiabilidad y la "explosión" del tiempo de simulación.

Las herramientas de simulación a nivel de sistema se diferencian de las herramientas de modelado analítico en la distinta aportación que hacen al análisis de la Confiabilidad. Las herramientas de modelado analítico sólo utilizan modelos probabilísticos para representar el funcionamiento del sistema, es decir, básicamente caracterizan el efecto de un fallo en el sistema mediante un conjunto de probabilidades y distribuciones. Por el contrario, la simulación a nivel de sistema únicamente necesita conocer la tasa de llegada y los tipos de fallos, no precisando la caracterización del efecto de los fallos. Por lo tanto, los resultados de la simulación pueden identificar los mecanismos de las averías, obtener su probabilidad y cuantificar el efecto de los fallos, lo que permite la extracción de las características que se desea modelar, y ayudan a determinar y especificar la estructura y los parámetros de los modelos analíticos. Ejemplos de herramientas de simulación a nivel de sistema son NEST (NEtwork Simulation Testbed) [Dupuy90], DEPEND [Goswami92, Goswami97], REACT [Clark92], POLIS [Lajolo00] o BOND [Baldini03].

Mención especial merece en este punto, 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, con las ventajas de la programación orientada a objetos, y que es usada para desarrollar modelos ejecutables de sistemas *hardware-software* a diferentes niveles de abstracción, desde el nivel de arquitectura del sistema hasta el nivel de transferencia de registros. También puede trabajar con el estándar TLM (del inglés *Transaction Level Modeling*) [Cai03], y es posible extender UML (del inglés *Unified Modeling Language*) para expresar conceptos de SystemC [Ying06]. Se está convirtiendo en el estándar *de facto* en el codiseño industrial [Bolchini08], donde tanto los componentes *hardware* como los componentes *software* pueden ser descritos

utilizando un lenguaje común. Puede ser integrado en los flujos de diseño de los sistemas empotrados. Facilita la simulación y la inyección de fallos en sistemas basados en *hardware* y *software*, a diferentes niveles de abstracción [Misera07], permitiendo la verificación de los mecanismos de detección y tolerancia a fallos en las etapas iniciales de diseño, que se pueden refinar en posteriores etapas [Lajolo00]. Puede ser integrado en distintos entornos de diseño, incluyendo las transformaciones de UML a SystemC [Ying06] y de SystemC a VHDL. La disponibilidad de un único entorno para el diseño, simulación e inyección de fallos simplifica el diseño completo del sistema y la evaluación de la confiabilidad, ya que puede ser usado para definir los instantes y duraciones de las inyecciones y permite activar fallos simultáneos en distintos módulos.

3.4.2 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 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* (VHDL o Verilog, principalmente) 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 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.

En las primeras aproximaciones a esta técnica [Cheng95, Li95, Burgun96, Hong96, Li97, Cheng99], se utilizaba un emulador lógico compuesto por varias placas con FPGA en paralelo, así como el *software* de control necesario para poder programar (configurar) las FPGA y hacerlas funcionar en modo traza (como un simulador *software*).

La primera técnica de inyección de fallos desarrollada, también denominada inyección de fallos estática, consistía en detener el funcionamiento del sistema en un momento determinado y reconfigurar las FPGA. La reconfiguración podía ser de todas las FPGA, de un conjunto de ellas o solamente de una. Además, la reconfiguración podía ser total (la FPGA entera) o parcial (sólo algunas partes de la FPGA). El principal inconveniente de esta implementación es la gran cantidad de tiempo perdido en la reconfiguración.

Para evitar las excesivas pérdidas de tiempo del método anterior, surgió la inyección dinámica, consistente en modificar el diseño original para hacerlo "inyectable" (en inglés *fault-injectable*). Con este método, en primer lugar, se añaden al modelo entradas adicionales de control que indicarán si se inyecta un fallo o no, y dónde. En segundo lugar, se modifica el diseño para que la FPGA pueda ejecutar tanto la función correcta del mismo como las versiones erróneas. La activación de cada "versión" se regula mediante una entrada de control. De esta manera, con un registro de desplazamiento circular se puede

seleccionar de manera sucesiva cada una de las "versiones" de la función, y estudiar el comportamiento del sistema en su conjunto.

Con el aumento de la densidad de integración y la capacidad de procesamiento de las FPGA, han aparecido en el mercado placas de prototipado (o de desarrollo) que se pueden conectar directamente a un computador "maestro" [Leveugle01]. Además, su bajo coste (comparado con el de un sistema basado en emulador) y la eliminación del *software* emulador han aumentado la capacidad de automatización de esta técnica. Sin embargo, las placas de desarrollo presentan un nuevo tipo de problema: el número de *pines* de entrada/salida disponibles para implementar funciones es bajo, ya que este tipo de tarjetas suelen disponer de dispositivos adicionales (visualizadores, señales de reloj, interruptores, pulsadores, *leds*, etc.) preconectados a la FPGA. Para solventar el problema, en [Leveugle01] se propone añadir al modelo sendos elementos adicionales para que hagan de interfaz de entrada y salida. A través de ellos se pueden multiplexar los *pines* disponibles para utilizarlos como entradas y salidas. El tamaño (en número de líneas) y conexión de estos elementos adicionales es dependiente del modelo, aunque es posible automatizar su inserción.

En cuanto al modo en que se realiza la inyección de los fallos, se pueden distinguir varios métodos. En primer lugar, en [Leveugle00, Leveugle01] se presenta una inyección de fallos basada en la generación de mutantes, mientras que en [Lima01] la inyección se basa en perturbadores, en ambos casos al nivel RT. Estas técnicas de inyección (mutantes y perturbadores) también se utilizan en la inyección de fallos mediante simulación, y se explicarán con más detalle en la sección 3.4.3.

La siguiente aproximación está basada en mutantes a nivel de puerta. En este caso, es necesario que el modelo sea estructural a nivel de puerta para tener acceso a los biestables. Posteriormente, los biestables del modelo inicial a nivel de puerta se sustituyen por otros capaces de inyectar fallos a través de unas señales de control. Sin embargo, una vez realizado este proceso, existen diferencias en los modelos y, en consecuencia, en los métodos para ejecutar la inyección de fallos. El primer método [Velazco01a, Velazco01b] presenta una circuitería de inyección puramente combinacional, mientras que en el segundo método [Civera01a, Civera01b] el mutante del biestable es más sofisticado, conteniendo además de la circuitería de inyección, un segundo biestable conectado en serie con los demás biestables, de manera que constituyen una cadena. Esta cadena se puede escribir y leer a través de dos líneas. En el momento de la inyección, en función del contenido de los biestables secundarios, se conmuta el contenido de los primarios. Es decir, se implementa un mecanismo similar al *Boundary Scan*, una técnica HWIFI descrita anteriormente. Este método es mucho más potente que el anterior, ya que permite inyectar fallos múltiples.

En [Civera01b] se presenta FIFA (*Fault Injection by means of FPGA*). En este caso, así como en [Civera01a, Civera01c] el modelo se instrumenta, es decir, el modelo VHDL original es modificado para incorporar la circuitería necesaria para la inyección. A la hora de realizar la inyección, es necesario recargar el modelo en la FPGA para cada conjunto de experimentos, con la consiguiente pérdida de tiempo. Para evitar estas pérdidas de tiempo, existe otra línea de trabajos [Antoni02, deAndrés03, Parreira03] en los que la inyección se realiza mediante una "reconfiguración dinámica" con el fin de evitar tanto la modificación del modelo como el tiempo de recarga del mismo. En este caso se utiliza la capacidad de reconfiguración en tiempo de ejecución que presentan algunas familias de FPGA para modificar dinámicamente la configuración de la FPGA en función del fallo a inyectar.

Por último, en [Alderighi03] se presenta una herramienta que permite inyectar fallos en el mecanismo de control de la configuración de FPGA de tipo *Virtex*, a diferencia de los métodos explicados anteriormente, en los que los fallos se inyectan en la configuración de celdas de memoria y en registros de usuario. La inyección se realiza mediante la modificación del *bitstream* de configuración mientras éste se está descargando en la FPGA, consiguiendo que la herramienta de inyección sea independiente de la herramienta de síntesis. Con este tipo de inyección se consigue estudiar el efecto de fallos de tipo SEU que

afectan a la correcta configuración del dispositivo. Además, se puede acceder y analizar el efecto de estos fallos en cualquier registro de configuración de la máquina de estados finitos. La herramienta permite el análisis de fallos simples y múltiples del tipo SEU que afectan al mecanismo de control de la configuración. La inyección se realiza mediante otra FPGA, siendo ésta controlada por un PC. Esta FPGA inyectora modifica el *bitstream* mientras se está cargando en el sistema bajo prueba (que es otra FPGA). La modificación consiste simplemente en el cambio del valor lógico del bit de configuración. Esta modificación implica también recalcular el CRC del *bitstream*.

Las ventajas de la inyección de fallos basada en emulación son:

- Es muy rápida.
- Puede realizarse durante las primeras etapas del diseño del sistema (cuando el modelo sintetizable esté disponible).
- Existe la posibilidad de conectar la FPGA al entorno real (esta técnica se denomina en inglés *in-system emulation*).
- Permite mayor accesibilidad, un mayor conjunto de modelos de fallos y un análisis más exacto de la patología de los fallos que la inyección de fallos implementada mediante *hardware*.

Los inconvenientes de la inyección de fallos basada en emulación son:

- Al ser circuitos sintetizables, sólo se puede utilizar un subconjunto del VHDL estándar.
- Con algunas técnicas no se puede inyectar en todas las partes del modelo.
- Existe un número reducido de emuladores comerciales.
- Se necesita modificar el código original, con lo que este código modificado puede no cumplir los requerimientos del código original, tanto temporales (la frecuencia máxima del reloj en la FPGA depende del código introducido, por lo que se debe optimizar el código escrito), como espaciales (el nuevo código tiene que "caber" en la FPGA). Es decir, se deben evitar las sobrecargas temporal y espacial.
- Lectura de datos restringida por el emulador (número máximo de E/S).
- Es necesaria la reconfiguración de la FPGA, y en algunos casos, descargar el fichero de configuración.

3.4.3 Técnicas de inyección de fallos basadas en VHDL

A la hora de realizar el modelo de un sistema, una de las posibilidades más frecuentemente utilizadas es recurrir a un lenguaje de descripción de *hardware* (HDL, del inglés *Hardware Description Language*). En la actualidad se utilizan principalmente dos: Verilog [IEEE95] y VHDL (del inglés *Very high speed integrated circuit Hardware Description Language*) [IEEE93]. Esta sección introduce algunos conceptos básicos acerca de la inyección de fallos basada en VHDL, ya que es una técnica comúnmente utilizada por las técnicas de inyección de fallos basadas tanto en simulación como en emulación. En concreto, en este trabajo se han inyectado fallos mediante simulación de modelos en VHDL, cuyos resultados se expondrán posteriormente.

Del lenguaje en sí, es interesante mencionar que se ha convertido en uno de los lenguajes más apropiados desde el punto de vista de la simulación de fallos. 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 descripciones estructurales y comportamentales en un único elemento sintáctico.
- Ofrece muy 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 gran controlabilidad y observabilidad.

Para realizar la inyección de fallos sobre modelos en VHDL, en la bibliografía consultada se proponen básicamente 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, DGil00, Baraza00], sin tener que modificar el código VHDL del mismo.

El segundo grupo se apoya en la modificación del código VHDL, mediante la inserción de componentes perturbadores (del inglés *saboteurs*) en arquitecturas estructurales [Amendola96, Boué98, Folkesson98, Gracia01, DGil03, Baraza08], o creando mutaciones (del inglés *mutants*) de componentes ya existentes [Ghosh91, Armstrong92, Gracia01, DGil03, Baraza08]. Existe otro conjunto de técnicas que se basan en la ampliación de los tipos y en la modificación de las funciones del lenguaje VHDL, a las que se denomina *otras técnicas* [DeLong96, Sieh97]. En la figura 3.2 se puede observar un resumen de las técnicas de inyección de fallos sobre modelos en VHDL.

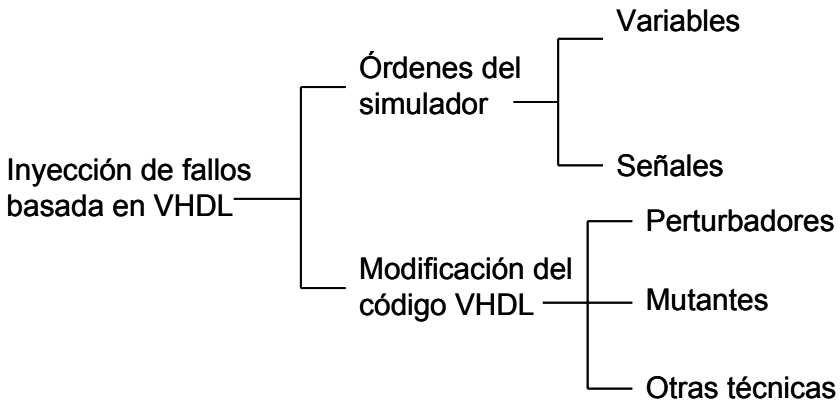


Figura 3.2: Técnicas de inyección de fallos sobre modelos en VHDL [Gracia01].

Inyección de fallos mediante órdenes del simulador

En la mayoría de los simuladores VHDL existen algunas sentencias del simulador que permiten la modificación del valor de las señales y de las variables del modelo durante la simulación del mismo. Utilizando adecuadamente estas órdenes, y asignando ciertos valores durante un tiempo determinado, se puede realizar la inyección de fallos. Dentro de esta técnica se pueden distinguir la manipulación de señales de la de variables:

- **Manipulación de señales**: mediante esta técnica, los fallos se inyectan alterando el valor o las características temporales de las señales del modelo VHDL, en modelos comportamentales. La inyección se realiza desconectando la señal de su *driver* o *drivers* para después forzar el nuevo valor. Una vez finalizada la inyección, la señal se vuelve a conectar a su(s) *driver(s)*.
- **Manipulación de variables**: en este caso, es posible alterar los valores de las variables del modelo, lo cual permite la inyección de fallos en modelos comportamentales de los componentes. A pesar de que la operación es muy similar a la manipulación de señales, en este tipo de inyección es muy importante tener en cuenta que no es posible controlar la duración de la inyección debido a la naturaleza de las variables en VHDL, por lo que no se pueden inyectar fallos permanentes.

Con esta técnica se han podido inyectar diferentes modelos de fallos, ampliando los clásicos modelos *bit-flip* para fallos transitorios y *stuck-at* para fallos permanentes. Además de estos modelos de fallos, se han implementado los modelos *open-line*, *pulse* (inversión lógica en elementos combinacionales [1]), *indetermination* (nivel de tensión que no se puede asociar a un valor lógico) y *delay* (alteración de los retardos de propagación). Además, estos fallos pueden ser inyectados con distintas duraciones: transitorios, permanentes e intermitentes.

Inyección de fallos mediante la modificación del modelo VHDL

Como ya se ha comentado previamente, existen dos técnicas que inyectan fallos mediante la manipulación y/o modificación del modelo VHDL:

- **Perturbadores**: estos elementos se añaden al modelo como componentes nuevos, y son utilizados para inyectar fallos en modelos estructurales. Tienen la función de alterar el valor o las características temporales de una o más señales a la hora de inyectar un fallo. La inyección se realiza cuando el perturbador es activado, permaneciendo inactivo durante el funcionamiento normal del sistema. Hay que tener en cuenta que las señales donde los perturbadores inyectan fallos son aquellas que conectan componentes en modelos estructurales. Los modelos de fallos que es posible inyectar con esta técnica son: *indetermination*, *delay*, *bit-flip*, *pulse*, *stuck-at* ('0', '1'), *open-line*, *stuck-open* (pegado-a '0' tras un determinado tiempo de retención), *short* y *bridging*.
- **Mutantes**: estos componentes se basan en la alteración de componentes ya existentes en el modelo VHDL para inyectar fallos en modelos comportamentales. Durante el funcionamiento normal, el componente mutado se comporta como el componente al que reemplaza. Sin embargo, cuando se inyecta el fallo, el componente mutado simula el comportamiento del componente original en presencia de fallos. La mutación puede realizarse de distintas maneras: añadiendo perturbadores a descripciones estructurales o comportamentales de los componentes, modificando descripciones estructurales mediante el reemplazo de componentes o modificando

[1] El modelo *bit-flip* es una inversión lógica en elementos de memoria.

estructuras sintácticas de descripciones comportamentales. Algunos modelos de fallos inyectables son [Armstrong92]: *stuck-then*, *stuck-else*, *assignment control*, *dead process*, *dead clause*, *micro-operations*, *local stuck-data* y *global stuck-data*.

3.5 Resumen y conclusiones

En este capítulo se ha realizado una clasificación de las técnicas de inyección de fallos utilizadas hasta la fecha. 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*).

Se han descrito las técnicas más representativas, mostrando las variantes que han aparecido junto con los trabajos más relevantes, y se han analizado las ventajas e inconvenientes de cada una de ellas. En cada una de las distintas técnicas revisadas se ha hecho hincapié en los modelos de fallos que se pueden inyectar. En el capítulo siguiente se tratará este tema con mucha más profundidad, estudiando los mecanismos físicos que pueden provocar la aparición de fallos, la representatividad de los modelos de fallos respecto de los fallos reales y los distintos niveles de abstracción a los que se puede trabajar.

En particular, se ha hecho especial énfasis en las técnicas de inyección de fallos mediante simulación de modelos en VHDL, ya que son las que se utilizarán para la realización de los experimentos y obtención de resultados.

4

Modelos de fallos

4.1 Niveles de abstracción

El primer paso a la hora de definir 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]. Estos niveles se pueden ver en la figura 4.1. Existen otras clasificaciones, como las presentadas en [Walker85, Siewiorek92, Jenn94a, Pradhan96]. En estos casos, todos los niveles están relacionados e incluso se solapan.

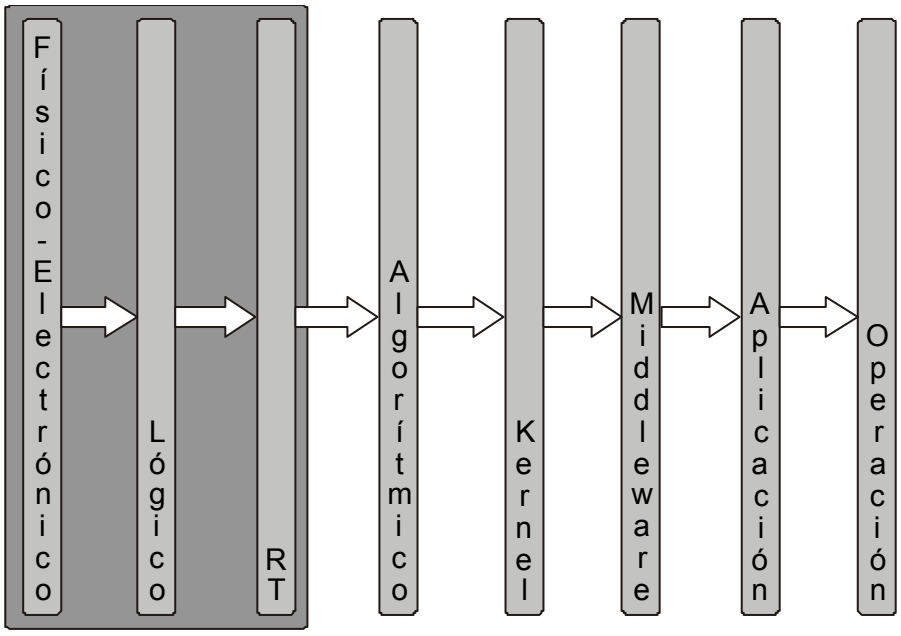


Figura 4.1: Niveles utilizados en el modelado de los fallos *hardware*.

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

En función de la figura 4.1 podemos decir que los fallos *hardware* abarcan los tres primeros niveles (físico-electrónico, lógico y RT). En este trabajo se van a estudiar modelos de fallos centrados en los niveles lógico y RT. Así pues, el resto de este capítulo presentará un resumen de los distintos modelos utilizados en estos dos niveles, y se propondrán nuevos modelos de fallos intermitentes.

4.2 Mecanismos de fallos

Antes de comenzar, se clasificarán los fallos en función de su duración:

- Fallos transitorios, con una duración corta en el tiempo.
- Fallos permanentes, que perduran de manera indefinida.
- Fallos intermitentes; son similares a los transitorios en cuanto a que tienen una duración finita, pero a diferencia de aquellos, se repiten en el tiempo sin un comportamiento periódico.

Tradicionalmente, los modelos de fallos predominantes han sido *stuck-at* ('0' y '1') para los fallos permanentes, y *bit-flip* para los fallos transitorios. Los fallos intermitentes se consideraban el preludio de fallos permanentes [DBench02, Smolens07], por lo que su estudio se incluía en el de éstos, y se utilizaba el mismo modelo de fallo.

Sin embargo, en los últimos años, el desarrollo tecnológico submicrométrico ha propiciado que el uso exclusivo de estos modelos en la inyección de fallos no sea representativo de la casuística real. Con el fin de subsanar esta deficiencia, ha sido necesario introducir nuevos modelos de fallos que sirvan tanto para los circuitos integrados actuales como para las futuras tecnologías.

A continuación se resumen los mecanismos físicos implicados en la ocurrencia de cada tipo de fallo, asociándoles una serie de modelos de fallo en los niveles de abstracción lógico y RTL [Amerasekera97, DGil99, DBench02, Baraza03].

4.2.1 Fallos permanentes

La figura 4.2 muestra la casuística de los fallos permanentes más significativos. Estos fallos se deben a defectos irreversibles en los circuitos, producidos durante el proceso de fabricación o por desgaste. Los óvalos muestran los modelos de fallos deducidos, mientras que los rectángulos muestran las causas y los mecanismos físicos que los provocan [Siewiorek92, Amerasekera97, Pradhan96, DBench02, Baraza03].

Para deducir los modelos de fallos a nivel físico-electrónico, los mecanismos han sido clasificados en dos grupos. En el primer grupo se consideran los mecanismos que provocan cortocircuitos y circuitos abiertos en las diferentes capas del transistor (metal, óxido, etc.). El efecto de estos fallos en los niveles lógico y RT es un conjunto de modelos de fallo que a veces están relacionados entre sí:

- ***Stuck-at* ('0', '1')** (pegado-a). Este es el modelo más comúnmente utilizado.
- ***Open-line*** (circuito abierto, o alta impedancia) en las líneas de conexión de circuitos lógicos.

- ***Short*** (cortocircuito) entre las líneas de conexión de circuitos lógicos.
- ***Bridging*** (puente), causado por combinaciones de cortocircuitos y circuitos abiertos. Aunque tradicionalmente el nombre de este modelo hace referencia a un cortocircuito [Pradhan86, Shaw01], algunos autores consideran como puente ciertas combinaciones entre cortocircuitos y circuitos abiertos que dan lugar a fallos más complejos [Jenn94a, DGil99], y al modelo considerado tradicionalmente como puente se le denomina cortocircuito (*short*).
- ***Stuck-open*** (pegado-a '0' tras un tiempo de retención). Este fallo se debe a nodos flotantes en alta impedancia, que mantienen el valor lógico del transistor durante un tiempo denominado tiempo de retención, que es el tiempo que tardan en descargarse las capacidades parásitas de salida a causa de las corrientes de fuga. Durante el tiempo de retención (del orden de milisegundos [Pradhan86]), el circuito presenta un comportamiento secuencial, hasta que se descarga (quedando entonces un valor lógico '0'). Este tipo de fallo es característico de los circuitos integrados MOS.
- ***Indetermination*** (indeterminación). En este caso, el fallo puede deberse tanto a cortocircuitos en las salidas del circuito lógico como a circuitos abiertos en las entradas.

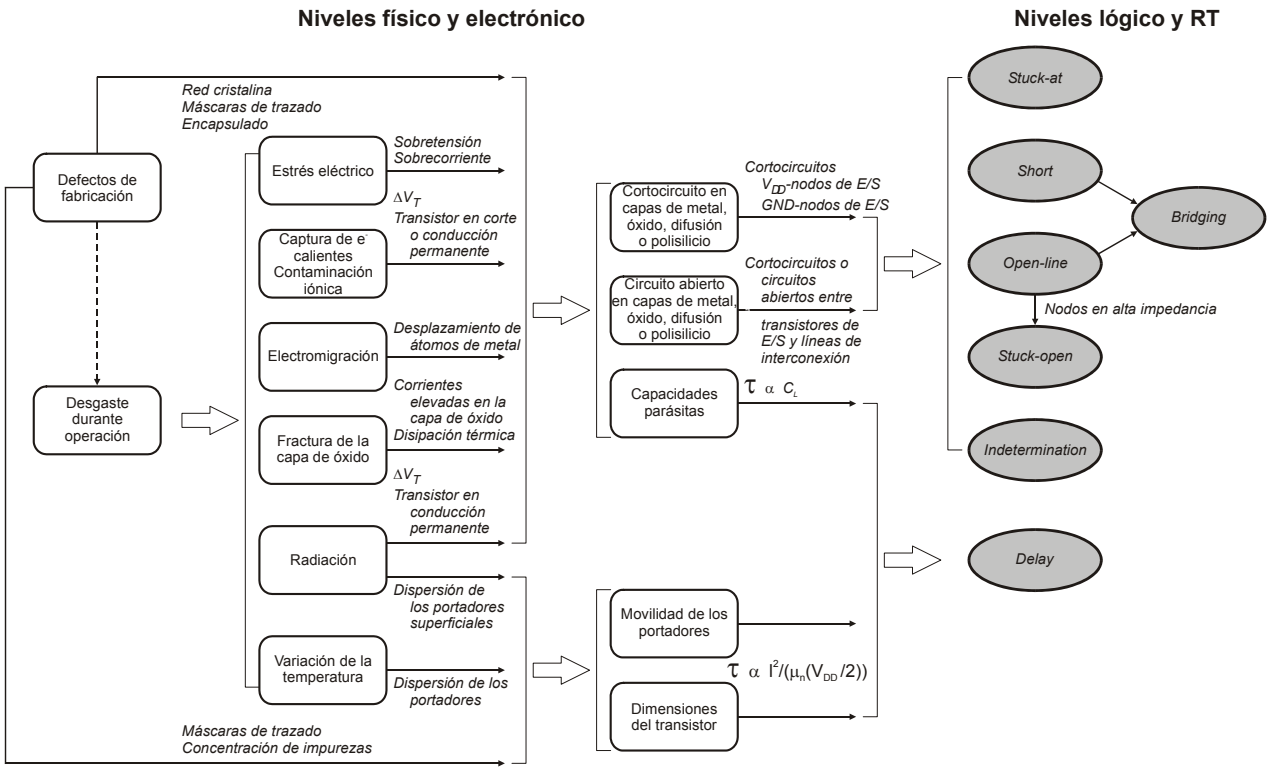


Figura 4.2: Mecanismos de fallos permanentes y modelos de fallo equivalentes [DBench02].

En el segundo grupo se incluyen algunos mecanismos de fallo que afectan al retardo de conmutación de los transistores MOS y a los tiempos de carga/descarga de las capacidades parásitas en las conexiones de entrada/salida: movilidad de portadores, modificación de las capacidades parásitas y variación de las dimensiones del transistor [DGil99]. Su efecto en los niveles lógico y RTL es una modificación permanente

de los retardos de los circuitos lógicos, por lo que se modelan con el fallo denominado ***delay*** (o alteración de los retardos).

4.2.2 Fallos transitorios

Estos fallos no se deben a ningún defecto físico en el circuito. También se denominan *soft errors* y *single event upsets* (SEU), y pueden aparecer durante el funcionamiento de un circuito por diferentes causas, tanto internas como externas. La figura 4.3 muestra los modelos de fallos deducidos para los fallos transitorios. En este caso, los modelos ***bit-flip*** (aplicado a elementos de almacenamiento: memorias y registros), ***pulse*** (pulso, aplicado a circuitos combinacionales) e ***indetermination*** permiten representar fallos físicos de diferente naturaleza: transitorios en la fuente de alimentación, diafonía (en inglés *crosstalk*), interferencias electromagnéticas (luz, radio, etc.), variaciones de temperatura, radiación (de partículas α y cósmica), etc.

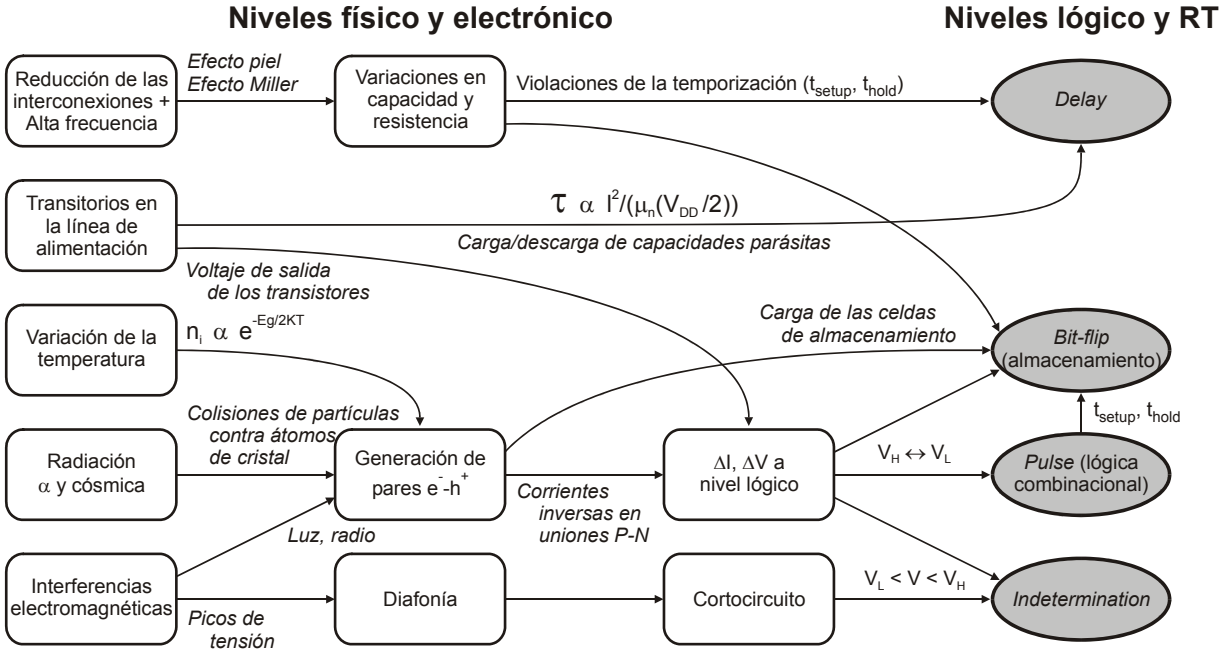


Figura 4.3: Mecanismos de fallos transitorios y modelos de fallo equivalentes [DBench02].

Estos fallos físicos pueden variar los valores de tensión y corriente de los niveles lógicos en los puntos de un circuito, como ocurre por ejemplo a causa de los transitorios en la tensión de alimentación. También pueden provocar la generación de pares electrón-hueco, que son barridos por el campo eléctrico de las zonas de deplexión de las uniones p-n de los transistores. La corriente de pares electrón-hueco generada puede modificar el valor lógico de un nodo del circuito, conmutando su valor. Si el nodo pertenece a una celda de almacenamiento (registros o memorias SRAM o DRAM) [Amerasekera97], el fallo se denomina *bit-flip*. Si pertenece a un circuito combinacional, se puede alterar el nivel lógico, bien para conmutarlo (modelo *pulse*) o para dejarlo indeterminado (modelo *indetermination*).

El modelo ***delay*** permite representar fallos físicos debidos a transitorios en la alimentación que pueden alterar el retardo de conmutación de los transistores MOS o los tiempos de carga/descarga de las

capacidades parásitas de las conexiones de entrada/salida. Además, debido al funcionamiento con frecuencias elevadas, pueden aparecer otras causas de fallos ligadas al incremento de la resistencia de las conexiones metálicas y de la capacidad parásita entre las líneas de conexión, afectando al retardo de propagación de las señales. Entre estas causas se encuentran, por ejemplo, el efecto piel (del inglés *skin effect*) [Walker00] y el efecto Miller [Sylvester99].

Uno de los problemas principales de los fallos transitorios y, por consiguiente, de sus modelos asociados, es determinar su duración. Sin embargo, se ha escrito muy poco acerca de este asunto [DGil99].

4.2.3 Fallos intermitentes

Tradicionalmente, los fallos intermitentes se han considerado 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. Se consideraban, por tanto, el preludio de un fallo permanente [DBench02, Smolens07]. Por este motivo, los modelos de fallo que se proponían eran los mismos que para los fallos permanentes.

Dada la importancia que pueden alcanzar los fallos intermitentes en las actuales tecnologías submicrométricas, y el hecho de que haya fallos intermitentes que no desemboquen en fallos permanentes, se va a profundizar en los mecanismos físicos y en los modelos de fallo asociados a los fallos intermitentes en un apartado específico, en concreto en el apartado 4.4.

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

4.3 Influencia de las tecnologías submicrométricas

A medida que aumenta el uso de los circuitos VLSI en sistemas fiables o de seguridad crítica (del inglés *critical-safety systems*), aumenta la importancia de la confiabilidad de dichos circuitos. Al reducir las geometrías y las tensiones de alimentación y aumentar las frecuencias de funcionamiento, la confiabilidad se resiente, ya que se incrementan las tasas de los fallos. A continuación se muestra un resumen del impacto de las tecnologías submicrométricas en los mecanismos físicos del fallo. De esta manera, se relacionan dichos mecanismos con modelos de fallo en los niveles lógico y RT. Un estudio más completo se puede encontrar en [Baraza03].

4.3.1 Fallos permanentes

Los grupos de mecanismos de fallo más significativos en las nuevas tecnologías son [Baraza03]:

- **Daños en la capa de óxido**
  - Rotura de la capa de óxido (*thin oxide breakdown*). Depende del espesor de la capa de óxido de la puerta. Básicamente, la rotura es un proceso que se produce en dos fases, conocidas

como desgaste y rotura [Hu99, Stathis01]. Los efectos pueden ser una excesiva corriente de fuga en los pines de entrada y salida, un incremento de la disipación de potencia en el circuito integrado y una disminución de la velocidad del circuito. Los modelos de fallos deducidos son ***short***, ***open-line***, ***indetermination***, ***delay*** y ***bit-flip*** (véase la figura 4.2).

- Inyección de portadores calientes (*hot carrier injection*). Habitualmente, la degradación de las prestaciones del transistor se atribuye a la presencia de cargas fijas en la capa de óxido a causa de la captura de electrones y huecos en dicha capa [Rodder95]. El efecto de este mecanismo de fallo es una degradación gradual de algunos parámetros del transistor, como la tensión umbral, la transconductancia y la corriente del drenador [Hawkins00]. Los modelos de fallos deducidos son ***indetermination*** y ***delay***.
- Daños por plasma. El empleo de plasma (gas ionizado) puede dañar las capas de óxido de los dispositivos MOS. El plasma incidente acumula carga en los electrodos de metal, habitualmente en las regiones donde el área o la periferia son grandes [Fang93]. Dicha carga se transfiere a la región del correspondiente electrodo, permitiendo un cierto flujo de corriente a través de la fina capa de óxido de la puerta [McVittie96]. El efecto es un incremento en la corriente de fuga de la capa de óxido. Los modelos de fallos deducidos son ***indetermination***, ***delay*** y ***bit-flip***.

- **Metalizaciones**
  - Electromigración. A medida que las conexiones metálicas se hacen más delgadas y aumenta la densidad de corriente, se incrementa el problema de la electromigración [Hawkins00]. El efecto es un incremento de la resistencia de interconexión, e incluso un circuito abierto. La electromigración también puede provocar cortocircuitos entre conexiones contiguas (en la misma capa) o situadas en capas adyacentes. A nivel lógico, los modelos que se pueden deducir son [DGil99]: ***stuck-at***, ***short***, ***open-line*** (en ocasiones ***stuck-open***), ***bridging***, ***indetermination*** y ***delay*** (véase la figura 4.2).
  - Migración inducida por estrés (*stress voiding*). Durante el proceso de fabricación de la oblea, el encogimiento posterior a la deposición de las diferentes capas puede producir una transferencia de átomos [Jones87, Tezaki90]. Este efecto se manifiesta mediante la formación de huecos (en inglés *voiding*) y de muescas (en inglés *notching*) en las líneas metálicas [Oates93], así como la formación de "rebabas" metálicas (en inglés *whiskers*) [Turner82], que pueden provocar cortocircuitos. Los modelos de fallos a nivel lógico son básicamente los mismos que en el caso anterior: ***stuck-at***, ***short***, ***open-line***, ***stuck-open***, ***bridging***, ***indetermination*** y ***delay***.
  - Otros: migración en los contactos (*contact migration*; basado en la migración de los átomos de metal y silicio en los contactos metálicos), migración en las vías (en los circuitos multicapa, en los contactos metal-metal; el mecanismo es similar a la migración en los contactos, pero en este caso sólo se desplazan átomos de metal, desde o hacia la vía) y microfracturas y defectos de cobertura de escalones (las regiones de una oblea con escalones pronunciados son susceptibles a la electromigración por el paso de elevadas densidades de corriente [Pol96], o a circuitos abiertos por la fractura de la metalización). Puesto que los efectos de todos estos mecanismos se pueden resumir como circuitos abiertos y cortocircuitos en las interconexiones metálicas [Amerasekera97], los modelos a nivel lógico son los mismos que para la electromigración y la migración por estrés: ***stuck-at***, ***short***, ***open-line***, ***stuck-open***, ***bridging***, ***indetermination*** y ***delay***.

- **Encapsulado y ensamblado**
  - Defectos de fijación del chip (*die attach*). Pueden ocasionar dos tipos de mecanismos de fallo: la aparición de huecos (*voids*) en la capa de adhesivo o la introducción de impurezas o de humedad por parte del adhesivo o la base. En ambos casos se produce la corrosión del chip, que origina otros fallos por desgaste. Los efectos de los mecanismos están relacionados con la fusión (en inglés *burnout*) del chip, variaciones paramétricas o la corrosión. Además, se puede producir la rotura del chip durante procesos con excesivo estrés mecánico o térmico.
  - Defectos de conexión. La conexión del chip con los *pines* se realiza mediante un cable y dos conexiones: una con el chip y otra con el soporte de los *pines*. Si el cable sufre tensiones (mecánicas) fuertes, la unión con el anclaje puede romperse, provocando circuitos abiertos. Existen varios puntos débiles: i) Si el cable es demasiado estrecho, tensiones fuertes pueden provocar la fractura del cable. ii) Si, por el contrario, la tensión es demasiado débil, el cable puede moverse, provocando cortocircuitos con cables adyacentes. iii) Dependiendo de las impurezas y la humedad existentes, se pueden generar aleaciones indeseables que pueden debilitar las uniones, así como poros. Todos estos problemas se traducen en la ruptura de las uniones, provocando circuitos abiertos. El efecto más común son los circuitos abiertos debidos a desplazamientos en las conexiones con el chip.
  - Deslaminación y efecto "palomita de maíz" (*popcorn effect*). 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). La degradación de los cables de conexión, la rotura del encapsulado, el desplazamiento de la metalización y la corrosión pueden ocasionar fallos eléctricos debidos al incremento de las corrientes de fuga, fallos intermitentes o circuitos abiertos.
  - Corrosión. Cuando llega humedad al chip, ésta actúa como catalizador del mecanismo de corrosión en las metalizaciones. Los principales mecanismos de fallo consisten en un incremento en la resistencia de las metalizaciones, que en ocasiones conducen a circuitos abiertos. A causa de la migración, o por la aparición de dendritas, también se puede producir un aumento en las corrientes de fuga entre líneas de metalización adyacentes, dando lugar a cortocircuitos.
  - En todos los casos, los mecanismos de fallo asociados son circuitos abiertos o cortocircuitos en las líneas, siendo los modelos propuestos básicamente los mismos que los presentados en la electromigración (***stuck-at***, ***short***, ***open-line***, ***stuck-open***, ***bridging***, ***indetermination*** y ***delay***).

4.3.2 Fallos transitorios

Como ya se ha comentado, este tipo de fallos afecta al sistema durante un intervalo de tiempo relativamente corto, sin estar causados por ningún defecto físico en el circuito. Estos fallos son muy problemáticos de tratar debido a su corta duración y a la imposibilidad de conocer la situación exacta del mismo antes de la ocurrencia del fallo. Tradicionalmente, el modelo básico utilizado ha sido el ***bit-flip***.

Algunos aspectos que más han influido en la reducción de las geometrías de los nuevos transistores submicrométricos son también la causa (directa o indirecta) del aumento de la tasa de fallos transitorios. Por ejemplo, en [Shivakumar02] se mencionan: i) la disminución de la tensión de alimentación, ii) el

aumento de la velocidad de conmutación, iii) el aumento de las etapas de segmentación (en inglés *pipelining*), y iv) la disminución de la carga crítica de las celdas. Como consecuencia, actualmente se producen más fallos transitorios, y además aparecen en lugares en los que tradicionalmente no se consideraban, como pueden ser los circuitos combinacionales.

Además, y debido a la reducción de las geometrías en los circuitos integrados actuales, aumenta la probabilidad de ocurrencia de fallos múltiples, ya que la energía necesaria para provocar un cambio de valor disminuye.

Radiación de partículas α

En los materiales utilizados en los encapsulados de los circuitos integrados existen impurezas radiactivas como torio, uranio, etc. Dichos elementos radiactivos son capaces de emitir partículas α con alta energía que pueden alterar el funcionamiento de los transistores del circuito integrado.

Normalmente, este mecanismo de fallo se ha asociado a las celdas DRAM y a los registros dinámicos, modelándose como un fallo de tipo ***bit-flip***. Sin embargo, tal como se ha comentado, los fallos ahora pueden aparecer en circuitos combinacionales, no siendo válido en este caso el modelo del *bit-flip*, ya que el efecto del fallo es distinto: cuando el efecto del impacto desaparece, el transistor recupera el estado correcto, ya que los circuitos combinacionales carecen de realimentación, por lo que no se "memorizará" el valor erróneo de tensión. Por eso, se ha definido un nuevo modelo de fallo denominado ***pulse***, que asume esta diferencia con el *bit-flip*, y que sólo es aplicable a elementos combinacionales. Esta característica es extensible a todos los fallos transitorios debidos a radiación, independientemente de su tipo.

Además, también es posible considerar otros modelos. Por ejemplo, si la carga generada es próxima a la carga crítica, el cambio en la tensión puede producir una salida indeterminada (modelo ***indetermination***) [DGil99].

Radiación de rayos cósmicos

Cuando los rayos cósmicos entran en la atmósfera, colisionan con átomos atmosféricos, produciendo partículas cósmicas: fotones, electrones, protones, neutrones, piones, etc. De entre estas partículas cósmicas, tradicionalmente se ha considerado a los neutrones de alta energía (superior a 1 MeV) como la principal fuente de fallos transitorios en dispositivos CMOS. Como en el caso anterior, el modelo más utilizado para representar los efectos de este mecanismo de fallo es el ***bit-flip***. Como ya se ha comentado, también hay que considerar el modelo ***pulse*** para elementos combinacionales, así como el modelo ***indetermination***.

Otros mecanismos

Además de la radiación, hay que tener en cuenta otras causas de fallos debidas al funcionamiento con frecuencias elevadas, ligadas sobre todo al incremento de la resistencia de las conexiones metálicas y de la capacidad parásita entre las líneas de conexión, afectando al retardo de propagación de las señales, como por ejemplo, el efecto piel (del inglés *skin effect*) [Walker00] y el efecto Miller [Sylvester99]. Dichas violaciones de los márgenes temporales de seguridad (que se modelan con el fallo ***delay***) pueden corromper los datos transferidos, provocando fallos de tipo ***bit-flip*** o ***indetermination*** en los elementos de almacenamiento.

Otros mecanismos que pueden provocar fallos transitorios, además de la radiación o los retardos en las interconexiones metálicas, son las variaciones transitorias de la tensión de alimentación y las

interferencias electromagnéticas internas (diafonía, o *crosstalk*) o externas [DGil99]. Estos mecanismos también pueden originar fallos de tipo ***bit-flip***, ***pulse***, ***delay*** e ***indetermination***, como se puede ver en la figura 4.2.

4.3.3 Fallos intermitentes

Está previsto que los fallos intermitentes, junto con los fallos transitorios, tengan un gran impacto en las nuevas tecnologías submicrométricas. Origen de estos fallos pueden ser la complejidad de los procesos de fabricación, que provoca residuos o variaciones en el proceso, junto con mecanismos de desgaste [Constantinescu02, Borkar03, Barros04, Kranitis06].

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

El estudio de las causas, mecanismos y efectos de los fallos intermitentes es una línea de investigación abierta en la que es necesario profundizar. En el siguiente apartado se hace una recopilación de causas y mecanismos físicos encontrados en la literatura científica, que servirán como base para proponer una serie de modelos de fallo específicos para fallos intermitentes. En apartados posteriores se estudiarán los efectos de los fallos intermitentes aplicando estos modelos.

4.4 Nuevos modelos de fallos intermitentes

A la hora de proponer modelos de fallo, es fundamental que sean representativos de los fallos que se producen en un sistema real en el nivel de abstracción que se esté considerando. En este sentido, para poder establecer modelos de fallo representativos de los fallos intermitentes, se puede actuar en dos sentidos: por una parte, estudiar los mecanismos y 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 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 (RTL).

Para facilitar la consulta por ambos caminos, en primer lugar se presentan en detalle los mecanismos físicos, 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.

4.4.1 Mecanismos físicos

Residuos de fabricación

En diferentes informes se han observado ráfagas de errores de un bit (SBE, del ingles *single bit errors*) en memorias [Constantinescu05]. 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.

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. Por tanto, 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 en el bus de control.

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

Como se indicó en el apartado 4.3.1, la deslaminación puede provocar el desplazamiento de la metalización o la degradación de los cables de conexión, entre otros problemas. 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 requerimientos temporales de forma intermitente, dependiendo de las condiciones del entorno (variaciones PVT). Incluso pueden aparecer cortocircuitos o circuitos abiertos intermitentes en líneas metálicas. Por tanto, los modelos de fallo propuestos son ***intermittent delay*** (aumento intermitente del retardo), ***intermittent short*** e ***intermittent open***.

Electromigración

Como se comentó en el apartado 4.3.1, a medida que las conexiones metálicas se hacen más delgadas y aumenta la densidad de corriente, se incrementa el problema de la electromigración. Al igual que para la deslaminación, su efecto es el incremento de la resistencia del material, pudiendo llegar a causar circuitos abiertos en las líneas, al producirse huecos (*voids*) [Constantinescu07]. La electromigración también puede provocar cortocircuitos entre conexiones contiguas (en la misma capa) o situadas en capas adyacentes. 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 dos 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 efectos capacitivos entre ellas, conocidos como efecto Miller.

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***.

Resumen

Evidentemente, este estudio no considera todas las causas físicas, ni pretende entrar en profundidad a ese nivel. Más bien se trata de una recopilación de efectos físicos estudiados por diferentes autores [Moore00, Stathis01, Constantinescu02, Borkar03, Constantinescu05, McPherson06], que pueden causar fallos intermitentes. Por tanto, la lista de modelos de fallo aquí deducidos no es una lista cerrada.

Como resumen de los mecanismos físicos estudiados, la figura 4.4 los representa gráficamente, relacionándolos con los modelos de fallo deducidos.

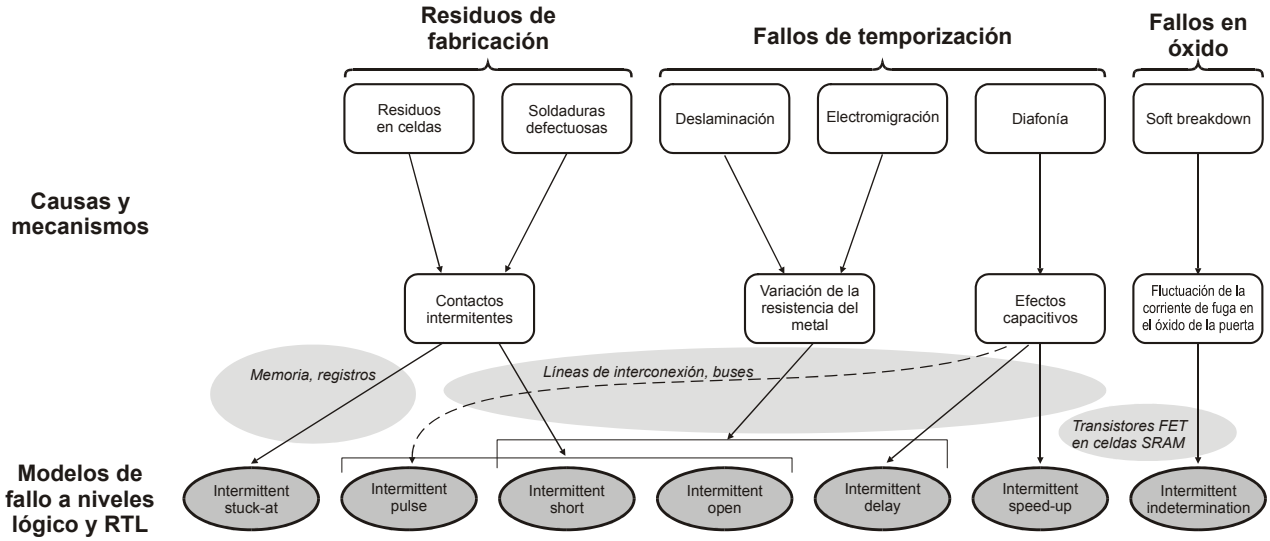


Figura 4.4: Mecanismos de fallos intermitentes y modelos de fallo equivalentes [Gracia08].

4.4.2 Modelos de fallo

En este apartado se recopilan todos los modelos de fallos intermitentes deducidos a partir de los mecanismos físicos recopilados en el apartado anterior.

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 observar en informes de errores de sistemas reales la aparición de ráfagas de fallos intermitentes en celdas de almacenamiento, que provocan errores que son detectados, reportados en *logs* y corregidos [Constantinescu05]. 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, poco se ha documentado. Posteriormente se ampliará esta cuestión.

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 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.

Se han detectado errores en los buses de datos, aunque es fácil deducir que el mismo problema puede aparecer en el bus de control.

También en líneas de interconexión se pueden producir pulsos intermitentes 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. Se ha deducido por varias vías. La primera de ellas 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. Finalmente, los estudios sobre 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].

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 anterior, distintos mecanismos físicos están 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, generalmente debidos a un incremento de la resistencia de los conductores. Es decir, el valor lógico que se envía por una línea de interconexión no llega en el tiempo esperado, lo que en caso de conmutación entre dos valores lógicos distintos puede suponer un funcionamiento inadecuado en los circuitos secuenciales.

Entre los mecanismos que pueden provocar ese incremento en la resistencia del material 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, que también puede provocar el incremento de la resistencia del material.

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.

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 ella no puede asociarse a ningún valor lógico. Este modelo de fallo pretende simular esta situación, cuando ésta se produce de forma intermitente. 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 causa fluctuaciones erráticas del voltaje de salida, y pueden aparecer tensiones no asociadas a ningún nivel lógico.

Resumen

Como resumen de los modelos de fallos intermitentes propuestos en este capítulo, se presenta la tabla 4.1 [Saiz09]. En ella aparecen brevemente los mecanismos físicos y los modelos de fallo asociados.

Tabla 4.1. Algunos mecanismos de fallos intermitentes y sus modelos [Saiz09]

Causas	Lugares	Mecanismos de fallo	Tipo 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	Desgaste/Temporización	*Intermittent delay* *Intermittent short* *Intermittent open*
Diafonía	Buses Conexiones de E/S	Interferencia electromagnética	Temporización Alteración del voltaje	*Intermittent delay* *Intermittent speed-up* *Intermittent pulse*
Daños en la capa de óxido	Transistores FET en celdas SRAM	Fluctuación de la corriente de fuga	Desgaste/Temporización	*Intermittent delay* *Intermittent indetermination*

4.4.3 Parametrización de los modelos de fallo

Una vez determinados algunos modelos de fallo aplicables a los fallos intermitentes, es preciso determinar cada uno de los parámetros que permiten modificar 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 celdas de almacenamiento o en líneas de interconexión. A la hora de configurar los fallos, se puede especificar si el fallo inyectado afectará a un solo bit (fallo simple) o a varios al mismo tiempo (fallo múltiple en el espacio). Si se inyecta un fallo múltiple, los bits afectados pueden ser adyacentes o no. El primer caso simularía bits de la misma celda de memoria o líneas contiguas de un bus, mientras que el segundo simularía, por ejemplo, líneas de interconexión que puedan estar en distintas capas del integrado.

Lugares de inyección

Los lugares donde se inyectarán los fallos dependen no sólo del modelo de fallo que se esté simulando, sino principalmente del modelo del sistema bajo estudio. No obstante, se pueden indicar unas líneas generales.

El *intermittent stuck-at* se inyectará en elementos de almacenamiento: memoria y registros. El *intermittent indetermination* afectará principalmente a los transistores de las celdas SRAM. El resto de los modelos de fallo deducidos afectan principalmente a líneas de interconexión, en especial a los buses del sistema.

Habría que estudiar en más profundidad los mecanismos físicos para analizar la posibilidad de que alguno de estos modelos de fallo pueda afectar a otros elementos del sistema, como la lógica combinacional, siendo ésta una línea de investigación abierta.

Consideraciones temporales

Como se ha comentado anteriormente, los fallos intermitentes se manifiestan habitualmente en ráfagas, es decir, cuando se produce la circunstancia física que activa 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 activó, o bien hasta que el fallo se convierte en permanente. Dado que la primera situación está cobrando mayor importancia con las tecnologías submicrométricas, este trabajo pretende iniciar el estudio de sus consecuencias.

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 fijar de distintas formas: valores fijos, seleccionadas aleatoriamente según distintas distribuciones de probabilidad, etc.

Simular la aparición de varias ráfagas requerirá una carga de trabajo suficientemente larga, lo que supera el ámbito de este trabajo, 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 un fallo intermitente real que se produce en una ráfaga, hay una serie de activaciones y desactivaciones del fallo. Esto es lo que se pretende mostrar en la figura 4.5, 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.

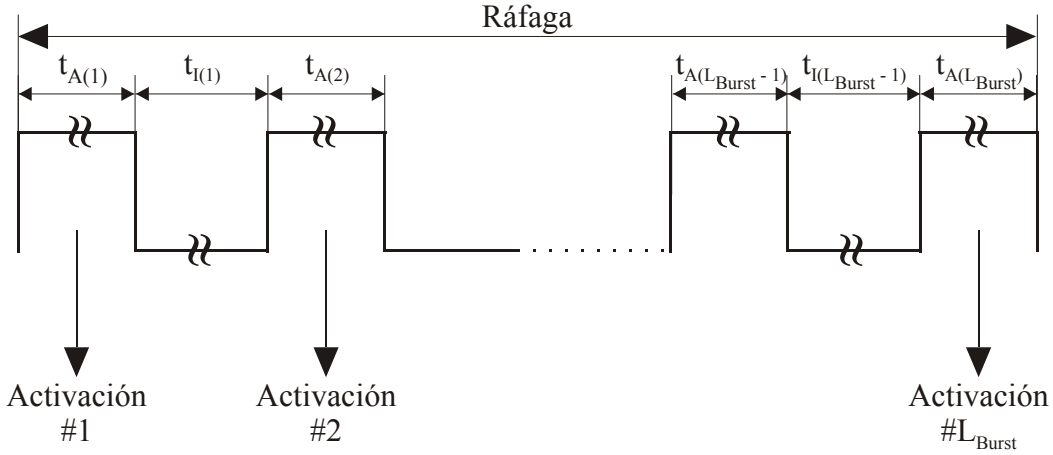


Figura 4.5: Parámetros de una ráfaga [Saiz09].

Como se puede observar, para modelar este comportamiento son precisos tres parámetros: el número de activaciones del fallo dentro de la ráfaga (tamaño de la ráfaga o L_{Burst}), tiempo de actividad (t_A) y tiempo de inactividad (t_I). Cada activación o desactivación del fallo puede tener distinta duración.

A la hora de configurar, se puede tomar un tamaño de ráfaga constante, o seleccionarlo 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 puede encontrar alguna pista acerca de los parámetros temporales, pero es muy poco lo que hay en la literatura científica al respecto. En el siguiente apartado se completa el tema 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 el valor 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 podría ser usar una distribución Uniforme tanto para el tamaño 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 puede ser válida ya que por lo general las condiciones que producen la activación y desactivación de un fallo intermitente no variarán 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; es muy flexible, modificando sus parámetros se obtienen distintas formas de distribución de probabilidad), etc.

4.5 Resumen y conclusiones

En este capítulo se ha presentado un resumen de los principales mecanismos de fallo en las tecnologías submicrométricas actuales, ampliándose el conjunto de los modelos de fallos utilizados tradicionalmente en los experimentos de inyección de fallos (*stuck-at* para los fallos permanentes y *bit-flip* para los transitorios). Estos modelos se han deducido para los niveles lógico y de transferencia de registro (RTL) y para fallos permanentes, intermitentes y transitorios.

La reducción de las geometrías ha potenciado la aparición de nuevos mecanismos de fallos permanentes e intermitentes. Estos mecanismos afectan principalmente a la capa de óxido de los transistores, las conexiones y el encapsulado de los circuitos integrados. En lo que se refiere a los fallos permanentes, aparecen defectos irreversibles en los circuitos, producidos durante el proceso de fabricación o por desgaste, y se manifiestan básicamente a nivel electrónico como cortocircuitos y circuitos abiertos, que a nivel lógico se pueden modelar como fallos de tipo *stuck-at*, *open-line*, *stuck-open*, *indetermination*, *delay*, *short* y *bridging*.

En cuanto a los fallos transitorios, la reducción de las geometrías y de las tensiones de alimentación, junto con el aumento de las frecuencias de funcionamiento, han causado una notable reducción del efecto de los mecanismos naturales de enmascaramiento de los circuitos digitales y la aparición de efectos capacitivos que afectan a la respuesta temporal de los circuitos. También son fuente de errores las radiaciones, tanto internas como externas. Los modelos de fallo establecidos son *bit-flip* (en elementos de almacenamiento), *pulse* (en lógica combinacional), *indetermination* y *delay*.

Se espera también un aumento en la tasa de fallos intermitentes, causado por las nuevas tecnologías submicrométricas, en las que aparecen nuevos mecanismos físicos que provocan fallos intermitentes que eventualmente no terminan en fallo permanente. Por ello es necesario profundizar en sus causas y mecanismos.

La complejidad de los procesos de fabricación puede provocar residuos o variaciones en el proceso, que producen contactos intermitentes en las celdas de memoria. Las soldaduras defectuosas pueden producir cortocircuitos o circuitos abiertos intermitentes en líneas de interconexión. La deslaminación de la capa de barrera y la electromigración pueden incrementar la resistencia del material, produciendo retardos, cortocircuitos o circuitos abiertos de forma intermitente. La diafonía aparece cuando hay un acople capacitivo entre dos líneas adyacentes, y puede producir picos (en una línea que debería permanecer a valor constante), retardos (conmutación más lenta de lo esperado) o aceleraciones (conmutación más rápida de lo esperado). Daños en la capa de óxido, como el mecanismo denominado *soft breakdown*, pueden producir que la corriente de fuga del óxido de la puerta fluctúe, de forma que se obtienen valores erráticos del voltaje de salida que pueden llevar a valores lógicos indeterminados.

Los fallos intermitentes se ven potenciados, y se pueden activar o desactivar, por cambios de temperatura, tensión de alimentación o frecuencia de trabajo (variaciones PVT). Los modelos propuestos para estos fallos son *intermittent stuck-at*, *intermittent pulse*, *intermittent short*, *intermittent open*, *intermittent delay*, *intermittent speed-up* e *intermittent indetermination*.

Para finalizar, es preciso comentar que no se ha intentado hacer un estudio exhaustivo de los distintos mecanismos de fallos, nuevas tecnologías, etc. Simplemente, se ha querido hacer un resumen de modelos de fallos deducidos para los distintos tipos de fallo, haciendo énfasis en los nuevos modelos propuestos para fallos intermitentes.

5

La herramienta de inyección de fallos VFIT

5.1 Introducción

Dos aspectos importantes en la realización de los experimentos de inyección son la automatización de los experimentos y la representatividad de los resultados. En el capítulo anterior se han comentado los modelos de fallos utilizados, intentando que fuesen realistas, con el fin de poder obtener unos resultados lo más aproximados posible a la realidad.

En este capítulo se va a describir **VFIT** *(**V**HDL-Based **F**ault **I**njection **T**ool)* [Baraza02, DGil03], la herramienta de inyección de fallos utilizada durante los experimentos de inyección. 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 todos los modelos de fallo vistos en el capítulo anterior, con distintos comportamientos espaciales y temporales. Otro aspecto remarcable de esta herramienta es la posibilidad de analizar automáticamente los resultados obtenidos en las distintas campañas de inyección. A partir de estos análisis, se puede estudiar el comportamiento y la respuesta del sistema ante fallos.

No es propósito de este trabajo explicar el desarrollo de VFIT. Se puede encontrar información más exhaustiva sobre este tema en [Baraza03].

5.2 Características generales de VFIT

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 campañas 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. Algunas características significativas son:

- Una única aplicación realiza automáticamente todas las fases de un experimento de inyección. Estas fases se detallan en la sección 5.3.
- Es universal y fácil de usar.
- VFIT permite inyectar un extenso conjunto de modelos de fallos, ampliando los clásicos modelos *stuck-at* y *bit-flip*. Con respecto a los modelos de fallos utilizados, éstos dependen de la técnica utilizada y del nivel de abstracción del modelo del sistema. Principalmente, se ha trabajado con circuitos modelados en los niveles de puerta, registro y chip.
- Respecto a la temporización de los fallos, se pueden inyectar fallos transitorios, permanentes e intermitentes, siendo posible elegir entre diferentes distribuciones verificadas en fallos reales (Uniforme, Exponencial, Weibull y Gaussiana) a la hora de determinar el instante de inyección y la duración del fallo.
- Permite utilizar diferentes técnicas de inyección en VHDL: órdenes del simulador, perturbadores y mutantes.
- 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 los mecanismos de detección y recuperación más apropiados con el fin de mejorar la Confiabilidad de un 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.

5.3 Fases de un experimento de inyección

Un experimento de inyección consiste en inyectar un determinado número de fallos en el modelo, según el valor de los parámetros de inyección. Para cada fallo inyectado se analiza el comportamiento del modelo, y al final del experimento, se obtiene el valor de los parámetros especificados.

Una campaña de inyección es un conjunto de experimentos de inyección diferentes, cambiando uno o varios de los parámetros de inyección (tipo y/o duración del fallo, lugar de inyección, carga de trabajo, etc.) en cada inyección. Un experimento de inyección consiste en tres fases independientes:

1. ***Configuración de la campaña.*** Con la ayuda de una interfaz gráfica, se especifican los parámetros de inyección y análisis:

   - Los parámetros de inyección más importantes son: técnica de inyección (órdenes del simulador, perturbadores, mutantes), número de inyecciones, lugar de la inyección (módulos, señales, etc.), modelos de fallos para cada lugar de inyección, distribución de los instantes de la inyección del fallo y de la duración del fallo, ciclo de reloj, carga de trabajo del sistema y duración de la simulación.

   - Por otra parte, las condiciones del análisis dependen del tipo de análisis que se realizará. Si es un estudio del síndrome de error, se deben especificar la clasificación de los fallos y establecer unas cláusulas para clasificar los errores. Si se realiza la validación de un sistema tolerante a fallos, se deben especificar unas cláusulas que indiquen la detección y recuperación de errores. En ambos casos, también se deben especificar cláusulas para determinar la terminación correcta o incorrecta de la carga de trabajo.

2. ***Simulación.*** En esta fase se realizan dos operaciones. En primer lugar, se generan automáticamente una serie de *macros* (ficheros especiales que contienen órdenes interpretables por el simulador) para realizar las inyecciones: una *macro* lleva a cabo una simulación sin fallos, mientras que el resto contienen los comandos necesarios para inyectar el número de fallos especificados. En segundo lugar, se ejecutan estas *macros* en el simulador de VHDL, obteniéndose una serie de ficheros de traza: uno con la simulación del sistema sin fallos y *n* ficheros con fallos inyectados (siendo *n* el número de inyecciones realizadas). Este es el caso más común cuando se están inyectando fallos simples. Sin embargo, hay que recordar que VFIT es capaz de inyectar fallos múltiples en una simulación.

3. ***Análisis de los resultados.*** La traza de la simulación sin fallos se compara con las *n* trazas de las simulaciones con fallos, analizando sus diferencias y extrayendo los parámetros de la confiabilidad del modelo del sistema. Dependiendo del tipo de análisis, el objetivo de la comparación es diferente, realizando el algoritmo de análisis las siguientes acciones:

   - En un *análisis del síndrome de error*, cuando se encuentra una discrepancia, se clasifica el error y el fallo en función de los tipos de error y fallo, calculándose además la latencia de propagación. Una vez que se ha clasificado el error, se comprueba el resultado de la carga de trabajo. Si el resultado es incorrecto, se actualiza el contador de averías. Los resultados típicos que se obtienen del análisis del síndrome de error son los porcentajes de errores efectivos y su latencia, así como los porcentajes de averías.

- En una *validación de un Sistema Tolerante a Fallos*, si no se encuentra ninguna discrepancia, se considera que el fallo inyectado no se ha activado. Si se encuentra una discrepancia, se busca el cumplimiento de las cláusulas de detección para determinar si el error activado ha sido detectado o no. Si no ha sido detectado, el error activado puede ser no-efectivo si el resultado de la carga de trabajo es correcto, o producir una avería si este resultado es incorrecto. Si el error ha sido detectado, se comprueba si se cumplen las cláusulas de recuperación para determinar si el error ha sido recuperado o no. Al final del análisis, se puede completar el *grafo de predicados de los mecanismos tolerantes a fallos* [Arlat93] (mostrado en la figura 5.1). Este diagrama refleja la patología de los fallos, es decir, la evolución de los fallos desde que estos son inyectados hasta que los errores son detectados y, eventualmente, recuperados. A partir del grafo, se pueden calcular las latencias medias de propagación, detección y recuperación, y las coberturas de detección y recuperación.

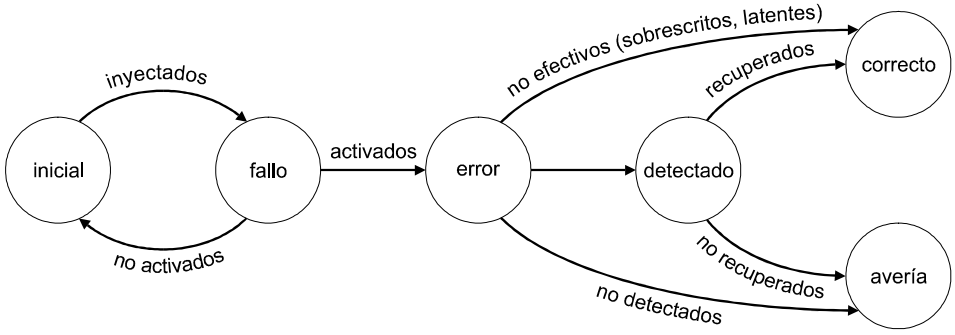


Figura 5.1: Grafo de predicados de los mecanismos tolerantes a fallos [Arlat93].

5.4 Diagrama de bloques de VFIT

La figura 5.2 muestra el diagrama de bloques detallado de VFIT. Puede verse como a partir de la interfaz con el usuario, el modelo del sistema bajo estudio y el simulador VHDL, la herramienta es capaz de proporcionar al usuario los resultados de los análisis realizados durante el experimento de inyección.

A continuación se resumen las funciones de los distintos elementos de VFIT.

Configuración

La función de este módulo es configurar VFIT, estableciendo los parámetros de la herramienta y del simulador, considerando que el simulador es parte de la herramienta. El aspecto más importante es enlazar las bibliotecas usadas.

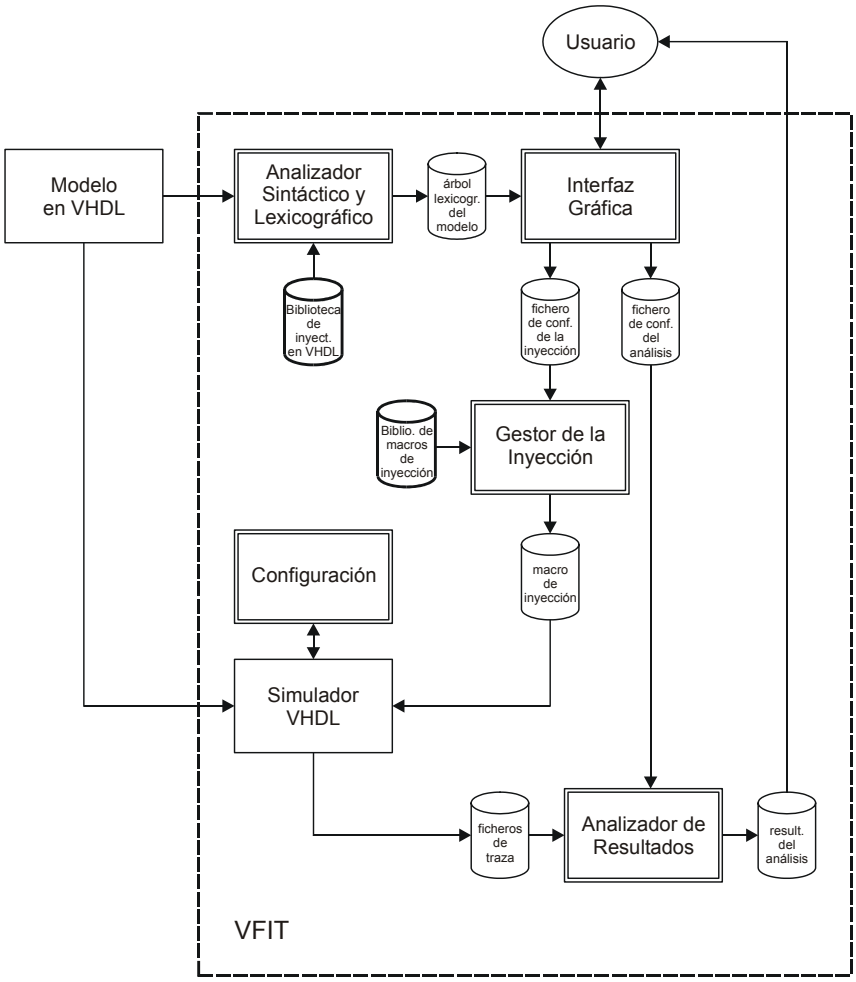


Figura 5.2: Diagrama de bloques detallado de VFIT [Baraza02].

Analizador Sintáctico y Lexicográfico

La misión de este módulo es el análisis de todos los ficheros del modelo para obtener su *Árbol Sintáctico*. Este árbol incluye, básicamente, todos los posibles puntos del modelo donde es posible realizar una inyección. La estructura en árbol refleja la estructura jerárquica del modelo, incluyendo *componentes*, *bloques* y *procesos*. Dependiendo de la técnica de inyección usada, los posibles puntos de inyección pueden variar:

- **Órdenes del simulador**. Las señales y variables atómicas del modelo del sistema, pertenecientes a las arquitecturas estructurales o comportamentales.
- **Perturbadores**. Las señales atómicas pertenecientes a descripciones estructurales del modelo del sistema.
- **Mutantes**. Los elementos sintácticos de las arquitecturas comportamentales del modelo VHDL.

Biblioteca de inyectores en VHDL

Esta biblioteca contiene un conjunto de perturbadores VHDL predefinidos que serán incluidos en el modelo cuando esta técnica sea usada. La librería también contendrá los mutantes generados a partir del modelo original.

Interfaz gráfica

Esta utilidad es un amplio conjunto de menús basados en ventanas que permiten al usuario especificar todos los parámetros y condiciones (relacionados con la inyección o el análisis), necesarios para realizar un experimento de inyección. Con estos parámetros y condiciones se generan los ficheros de *configuración de la inyección* y de *configuración del análisis*.

Biblioteca de *macros* de inyección

Esta biblioteca está compuesta por un conjunto de *macros* predefinidas utilizadas para activar los mecanismos de inyección según la técnica de inyección utilizada. Las *macros* se diseñan a partir de un subconjunto de órdenes del simulador.

Gestor de la inyección

Este módulo controla el proceso de inyección. Utilizando el fichero de *configuración de la inyección* generado por la *interfaz gráfica*, este módulo:

1. Crea una serie de *macros* de inyección, con el fin de realizar una simulación sin fallos y el número de simulaciones con fallos inyectados especificados en los parámetros, e
2. Invoca al simulador para ejecutar estas *macros*, obteniendo las *trazas de simulación*.

Simulador VHDL

Como se ha indicado anteriormente, se ha utilizado el simulador comercial de VHDL *ModelSim*, el cual proporciona un entorno para IBM-PC (o compatibles) bajo *Microsoft Windows*™. Este simulador está dirigido por eventos, y es muy simple y fácil de usar. Cuando se activa, el simulador ejecuta el fichero con las *macros* y genera las trazas de salida de la simulación.

Analizador de resultados

Este módulo toma como entradas el fichero de *configuración del análisis* generado por la *interfaz gráfica*. Según estos parámetros, compara la traza sin fallos con las trazas con fallos inyectados, buscando discordancias entre ellas, y extrayendo los parámetros especificados del análisis.

5.5 Resumen y conclusiones

En este capítulo se ha descrito VFIT, la herramienta de inyección de fallos utilizada durante los experimentos de inyección realizados en este trabajo. Las características más significativas son:

- Todas las fases de la inyección son ejecutadas automáticamente por una única aplicación.
- Se trata de una herramienta completa, de manera que puede:
  - Implementar diferentes técnicas de inyección: órdenes del simulador, perturbadores y mutantes.
  - Inyectar un amplio conjunto de modelos de fallos.
  - Inyectar fallos con distintas temporizaciones: transitorios, permanentes e intermitentes.
- Presenta una interfaz gráfica simple y fácil de usar.
- VFIT puede realizar dos tipos de análisis:
  - Síndrome de error.
  - Validación de un Sistema Tolerante a Fallos.

6

Estudio de los efectos de los fallos intermitentes

6.1 Introducción

Hay pocos estudios documentados de los efectos de los fallos intermitentes usando inyección de fallos basada en simulación. En la mayoría de los trabajos relacionados, se han monitorizado sistemas reales, y utilizando los informes de errores se han observado los fallos y averías producidas y se han estudiado para determinar las causas de errores más frecuentes y su manifestación [Constantinescu07].

La intención de este capítulo es presentar el trabajo realizado por el autor hasta el momento, estudiando el impacto de los distintos parámetros de los fallos intermitentes en el comportamiento del sistema. También se hará una comparación de los efectos de los fallos permanentes, transitorios e intermitentes.

Para ello se ha utilizado VFIT, la herramienta de simulación desarrollada en el GSTF y descrita en el capítulo anterior. Se han utilizado los modelos de fallos descritos en el capítulo correspondiente. Para el estudio se ha decidido utilizar el modelo VHDL del microcontrolador 8051 [MC8051]. Hay que hacer notar que, aunque el 8051 no sea un procesador fabricado con tecnología submicrométrica, la inyección de fallos se realizaría de forma similar. De hecho, la metodología utilizada se puede generalizar a cualquier microprocesador o microcontrolador más complejo.

6.2 El sistema bajo prueba y las cargas de trabajo

El 8051 es un microcontrolador muy popular. Pensado para su uso en sistemas empotrados, su núcleo aparece en muchos microcontroladores de otros 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.

En la figura 6.1 se puede observar un esquema de bloques del modelo del 8051 utilizado en las simulaciones.

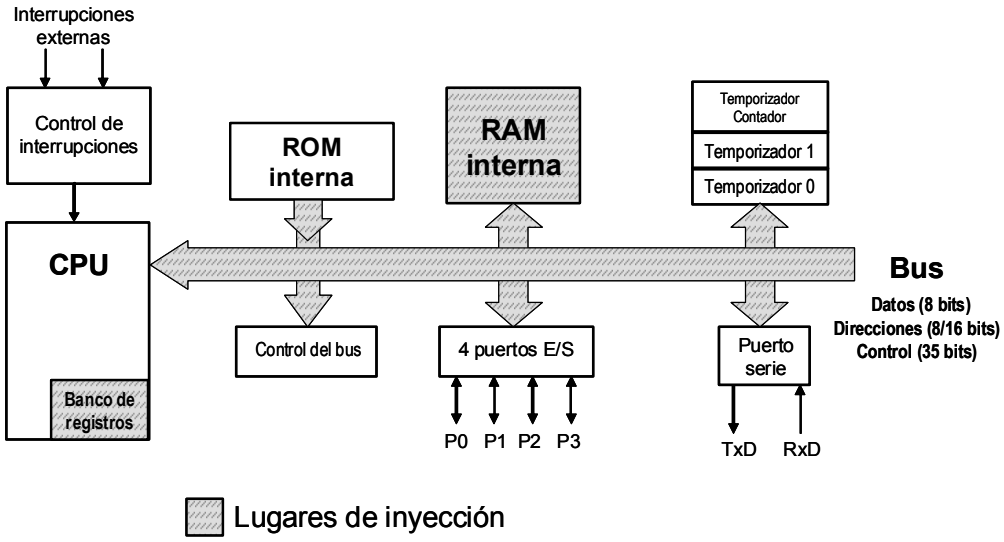


Figura 6.1: Esquema de bloques del modelo del procesador 8051 [DGil08].

El modelo permite simular la ejecución de distintos programas, denominados cargas de trabajo. En el caso de las simulaciones que se han planificado para llevar a cabo en los experimentos, se dispone de tres cargas distintas:

- Serie aritmética: consiste en sumar los números del 1 al 20.
- Algoritmo *Bubblesort*: ordenar un vector de 10 números con el algoritmo de la burbuja.
- Matrix: realiza el producto de dos matrices de tamaño 4×4.

Cada carga de trabajo tiene unas características particulares en cuanto a los recursos del sistema que utiliza y a su duración. Combinar los resultados obtenidos para todas ellas, permitirá obtener conclusiones suficientes para poder generalizar los resultados.

En cuanto a los lugares donde se van a inyectar los fallos, en la figura aparecen sombreados. Como se puede ver, se han seleccionado los lugares que aparecían en el estudio de las causas físicas de los fallos intermitentes como más susceptibles: elementos de almacenamiento (memoria RAM y banco de registros) y buses del sistema.

6.3 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
Campaña de inyección	Conjunto de experimentos de inyección en los que pueden variar algunos parámetros
$\mathbf{t_{inj}}$	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 del fallo intermitente (véase la figura 4.5)
$\mathbf{t_I}$	Tiempo de inactividad del fallo intermitente (separación entre activaciones; véase la figura 4.5)
$\mathbf{L_{Burst}}$	Longitud de la ráfaga (número de activaciones del fallo intermitente; véase la figura 4.5)
$\mathbf{N_{Injected}}$	Número de inyecciones realizadas
$\mathbf{N_{Propagated}}$	Número de errores propagados. En las campañas realizadas, se define como los 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 en la bibliografía en inglés *golden-run*) con la traza generada tras la simulación de la inyección. Se comparan los contenidos de los registros y/o posiciones de memoria en las 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.

El instante de inyección se selecciona aleatoriamente dentro de la duración de la carga de trabajo, siguiendo una distribución uniforme. En sistemas reales se ha observado que la aparición de los fallos

puede seguir otras distribuciones de probabilidad, como la Exponencial o la Weibull [Siewiorek92]. Por ejemplo, una distribución Weibull con una tasa de fallos creciente se puede utilizar para emular un proceso de desgaste o envejecimiento.

6.4 Influencia de los parámetros

En este apartado se describen las campañas de inyección realizadas con el objetivo de observar la influencia de distintos parámetros a la hora de inyectar fallos intermitentes. Se han realizado un total de cuatro campañas, variando determinados parámetros. Las campañas realizadas son:

- Fallos intermitentes simples en buses (nueve experimentos).
- Fallos intermitentes múltiples en buses (nueve experimentos).
- Fallos intermitentes simples en elementos de almacenamiento (nueve experimentos).
- Fallos intermitentes múltiples en elementos de almacenamiento (nueve experimentos).

Éstos son los principales parámetros:

Sistema bajo estudio: 8051.

Carga de trabajo: algoritmo *bubblesort*.

Puntos de inyección:

- Elementos de almacenamiento (banco de registros y memoria RAM).
- Buses del sistema.

Técnica de inyección: órdenes del simulador

Simulaciones por experimento: 1000.

Multiplicidad espacial:

- 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*.

$\mathbf{t_A}$: 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 dentro de la campaña.

$\mathbf{t_I}$: 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 dentro de la campaña.

$\mathbf{L_{Burst}}$: generada mediante una distribución Uniforme en el rango [1, 10].

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

6.4.1 Influencia de los parámetros de la ráfaga

En el apartado 4.4.3 se hablaba del modelado de la ráfaga, 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.

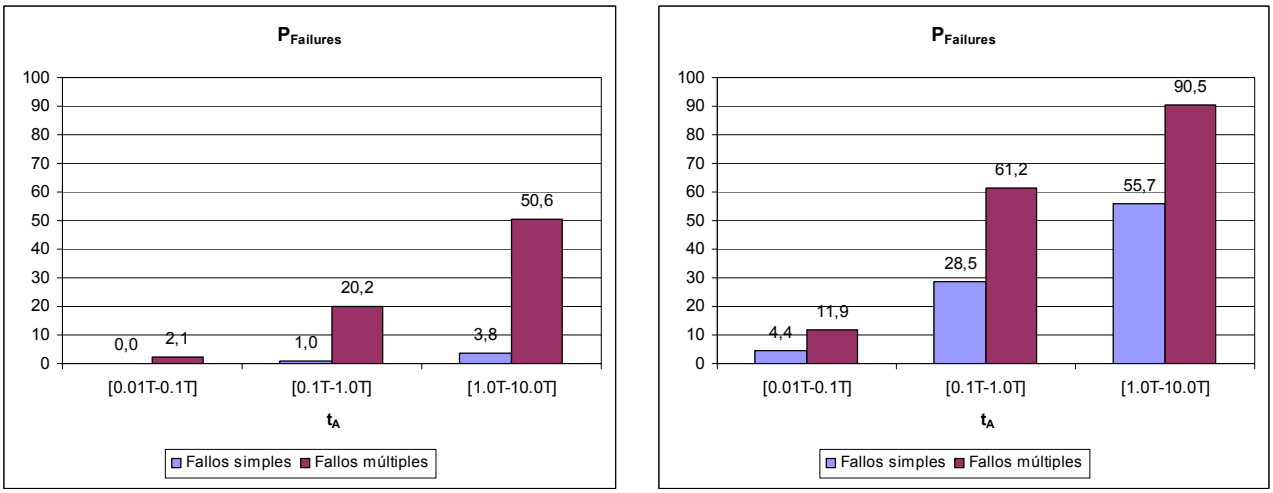
Tiempo de actividad

Las figuras 6.2 y 6.3 presentan los resultados de las distintas simulaciones tras variar el tiempo de actividad. La figura 6.2 muestra el porcentaje de averías, y la 6.3 el porcentaje de errores latentes. Las gráficas 6.2 (a) y 6.3 (a) representan los datos obtenidos en las inyecciones realizadas en elementos de almacenamiento, mientras que las gráficas 6.2 (b) y 6.3 (b) indican los datos obtenidos en las inyecciones en buses. En cada gráfica se pueden observar tres pares de datos, donde la columna izquierda de cada par representa el dato obtenido con fallos simples, mientras que la derecha representa el dato obtenido con fallos múltiples. Cada par representa un intervalo de tiempos de actividad distinto.

Como se puede comprobar, y como era previsible, conforme se va aumentando el tiempo de actividad se observa mayor porcentaje de averías. Este comportamiento se repite tanto en fallos simples como en fallos múltiples, y tanto en elementos de almacenamiento como en buses.

También se puede constatar, aunque se ampliará posteriormente, que los porcentajes de avería son mayores para fallos múltiples que para fallos simples, como cabía esperar. También se observa que en los buses aparecen porcentajes de averías mayores que en los elementos de almacenamiento.

En cuanto a los porcentajes de errores latentes, se puede comprobar que no muestran una tendencia clara respecto del tiempo de actividad. Mientras que en los elementos de almacenamiento se produce un descenso, en los buses no presenta un patrón determinado.



(a) **(b)**

Figura 6.2: Influencia del tiempo de actividad en el porcentaje de averías en función del lugar de inyección. (a) En elementos de almacenamiento. (b) En buses.

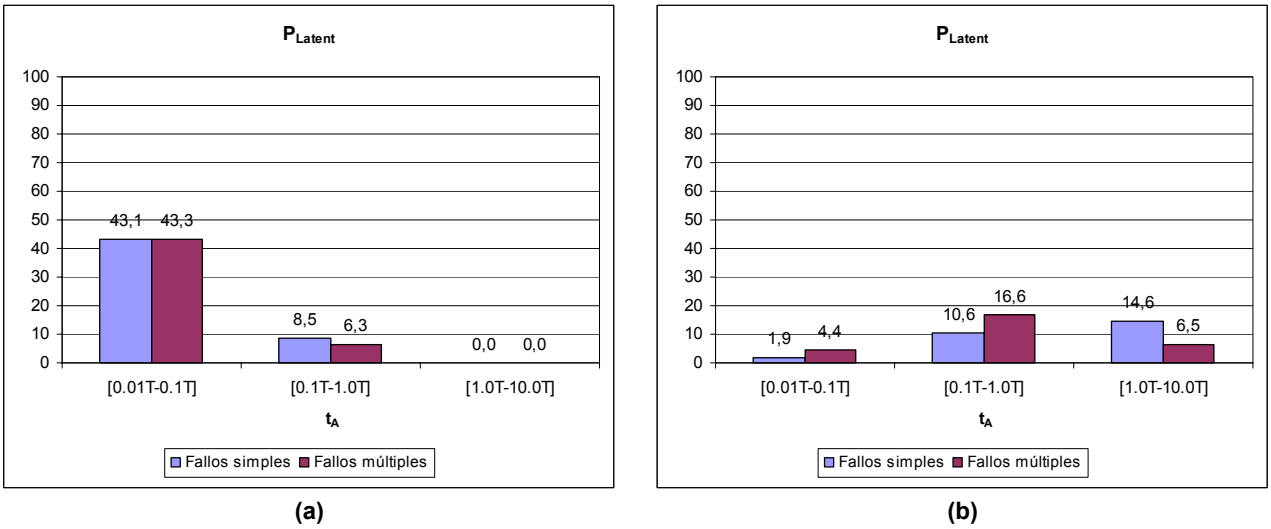


Figura 6.3: Influencia del tiempo de actividad en el porcentaje de errores latentes en función del lugar de inyección. (a) En elementos de almacenamiento. (b) En buses.

Tiempo de inactividad

Los resultados de variar el tiempo de inactividad (separación entre activaciones) se muestran en las figuras 6.4 y 6.5. La figura 6.4 muestra el porcentaje de averías, y la 6.5 el porcentaje de errores latentes. Las gráficas 6.4 (a) y 6.5 (a) representan los datos obtenidos en las inyecciones realizadas en elementos de almacenamiento, mientras que las gráficas 6.4 (b) y 6.5 (b) indican los datos obtenidos en las inyecciones en buses. Como en el apartado anterior, en cada gráfica se pueden observar tres pares de datos, siendo la columna izquierda de cada par el dato obtenido con fallos simples, y la derecha el dato obtenido con fallos múltiples. En este caso cada par representa un intervalo de tiempos de inactividad distinto.

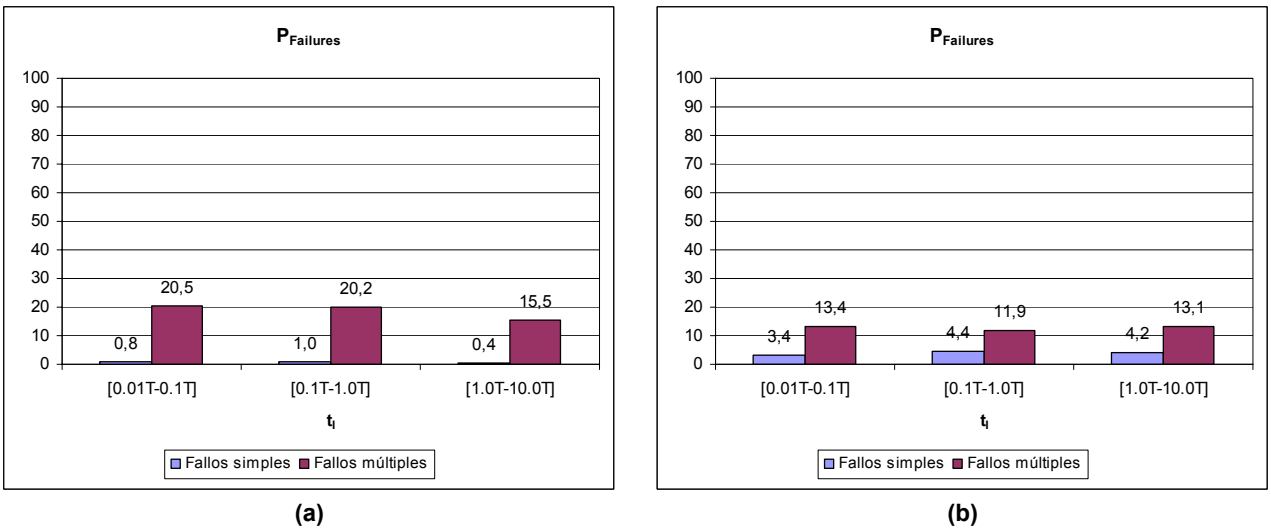


Figura 6.4: Influencia del tiempo de inactividad en el porcentaje de averías en función del lugar de inyección. (a) En elementos de almacenamiento. (b) En buses.

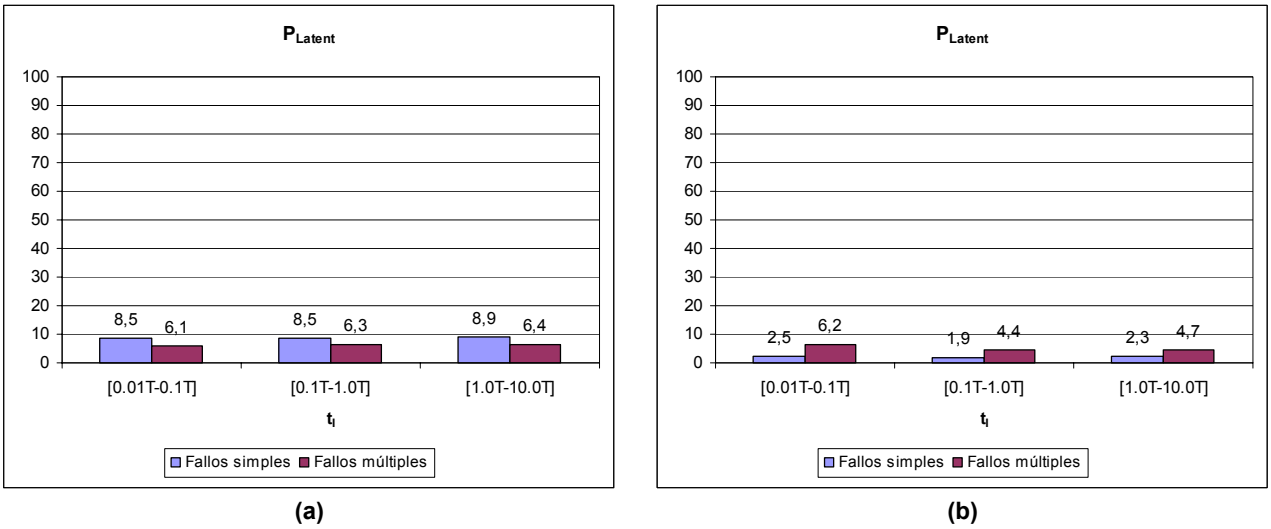


Figura 6.5: Influencia del tiempo de inactividad en el porcentaje de errores latentes en función del lugar de inyección. (a) En elementos de almacenamiento. (b) En buses.

En cuanto a los resultados, no se aprecian diferencias significativas al variar el tiempo de inactividad. La separación entre activaciones del fallo dentro de una ráfaga no provoca variaciones apreciables en el porcentaje de averías o de errores latentes. Este comportamiento es independiente del lugar de inyección.

Longitud de la ráfaga

El número de activaciones del fallo dentro de la ráfaga se ha variado entre 1 y 10. En un proceso de envejecimiento, la tendencia esperada es que el número de activaciones dentro de una ráfaga se vaya incrementando con el paso del tiempo, hasta que se convierte en un fallo permanente.

La figura 6.6 muestra los resultados obtenidos variando la longitud de la ráfaga. La gráfica 6.6 (a) refleja los datos tras inyectar en elementos de almacenamiento, y la 6.6 (b) los valores de las inyecciones en buses. En ambas, la columna izquierda de cada par representa el porcentaje de averías y la derecha el porcentaje de errores latentes. Cada par representa un número de activaciones del fallo distinto.

Respecto a las averías, conforme aumenta la longitud de la ráfaga, se incrementa su porcentaje, pero no lo hace de una forma lineal; más bien aumenta asintóticamente, hasta alcanzar alrededor del 28% en el caso de los elementos de almacenamiento, y del 75% en el caso de los buses. Esto supone una estabilización de los porcentajes de averías a partir de 8 ó 9 activaciones del fallo en una ráfaga.

En cuanto a los errores latentes, aparecen pequeñas variaciones sin una tendencia definida.

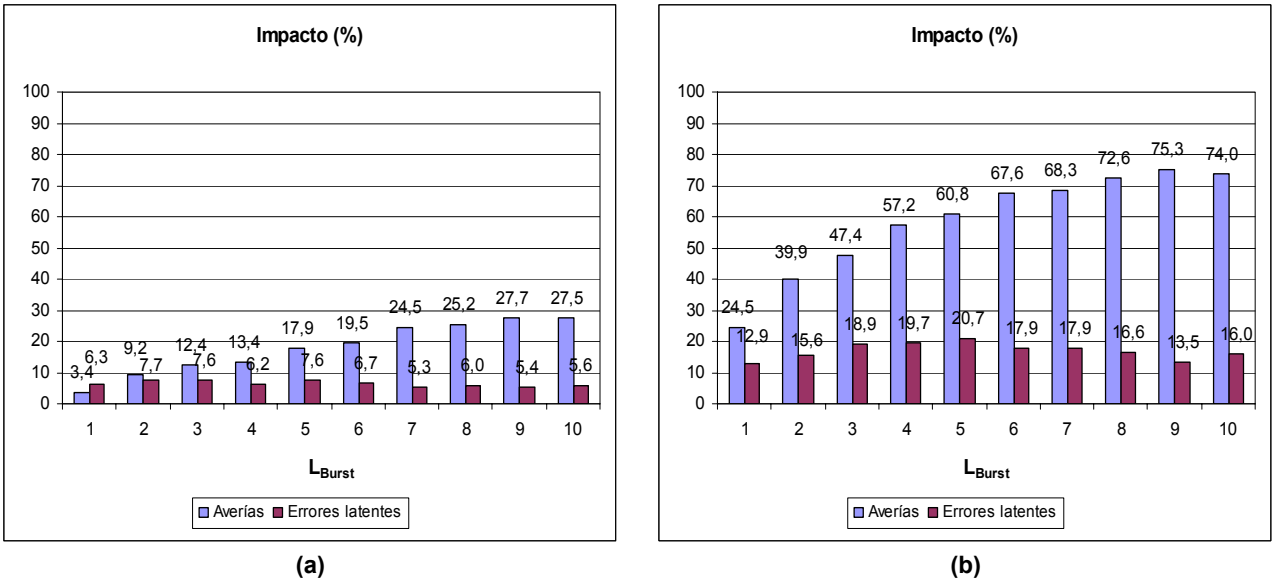


Figura 6.6: Influencia de la longitud de la ráfaga en los porcentajes de averías y de errores latentes en función del lugar de inyección. (a) En elementos de almacenamiento. (b) En buses.

6.4.2 Influencia de otros parámetros

Consideraciones generales

Por supuesto, no sólo los parámetros de la ráfaga afectan a los efectos de los fallos intermitentes. Los efectos observados en las simulaciones dependerán de muchos 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 este estudio inicial se ha utilizado un solo sistema, el 8051, y concretamente una versión del modelo VHDL desarrollado por Oregano Systems. 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.

Estos resultados también dependerán de la **carga de trabajo** utilizada. Dependiendo de su duración, el uso de los recursos del sistema, las operaciones que se realizan, etc. se pueden obtener distintos resultados. De nuevo, para generalizar conclusiones hay que realizar experimentos con varias cargas.

Más específicamente de la forma en que se realizan las simulaciones, los resultados dependerán de la herramienta de simulación y de la **técnica de inyección** de fallos utilizada. En el caso de este estudio inicial la herramienta utilizada ha sido VFIT, como se ha comentado anteriormente, y la técnica de inyección de fallos ha sido el uso de 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 ampliar este estudio con más modelos de fallo habrá que utilizar otras técnicas (perturbadores).

No obstante lo comentado en el párrafo anterior, la influencia de la herramienta de simulación debe ser mínima, si la herramienta de simulación 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.

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.

A continuación se van a estudiar con algo más de detalle algunos parámetros que pueden tener especial incidencia o que pueden ayudar a obtener algunas conclusiones.

Multiplicidad espacial

Como se ha podido observar en los datos mostrados anteriormente, 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 valores hasta 20. Merece la pena resaltar que los mayores valores de este ratio se han obtenido para los menores valores de t_A, y especialmente en las inyecciones realizadas en elementos de almacenamiento.

Lugar de la inyección

A partir de los resultados mostrados anteriormente se puede observar que los fallos intermitentes en buses son más dañinos que en elementos de almacenamiento. Una de las razones es la diferencia en el área ocupada por estos dos objetivos de la inyección. El tamaño en bits de los registros más la memoria es mucho mayor que el de los buses. Esto implica que la probabilidad de afectar a la carga de trabajo es menor. Además, los buses son el cuello de botella en las fases de búsqueda y ejecución de las instrucciones.

Esto sugiere que los buses son un área especialmente sensible, y que sería necesario protegerla con técnicas de mitigación para detectar y tolerar los fallos intermitentes.

Frecuencia del sistema

Una de las tendencias actuales, y que afecta de manera importante a la aparición de fallos intermitentes, es el constante aumento de la frecuencia de funcionamiento de los sistemas VLSI. En este estudio se ha querido comprobar cómo afecta este aumento a los efectos de los fallos intermitentes. La figura 6.7 muestra los porcentajes de averías y de errores latentes para fallos simples y múltiples en buses. En ambos casos, $P_{Failures}$ crece asintóticamente. Para fallos simples tiende al 60% y para fallos múltiples se estabiliza alrededor del 90%.

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 clara. 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%.

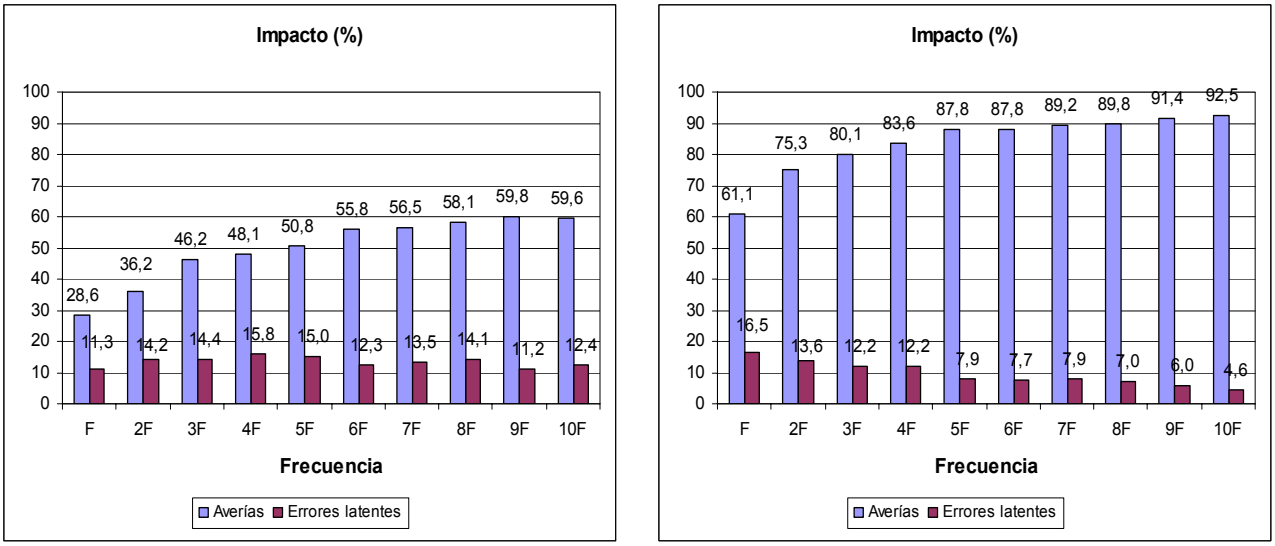


Figura 6.7: Influencia de la frecuencia de reloj en los porcentajes de averías y de errores latentes en buses. (a) Fallos simples. (b) Fallos múltiples.

La figura 6.8 muestra los resultados obtenidos al inyectar los fallos en elementos de almacenamiento (memoria y registros). El resultado es un comportamiento asintótico en el porcentaje de averías similar al observado anteriormente, aunque los valores obtenidos son notablemente inferiores a los observados en buses. Los valores de P_{Latent} muestran un descenso exponencial muy pronunciado.

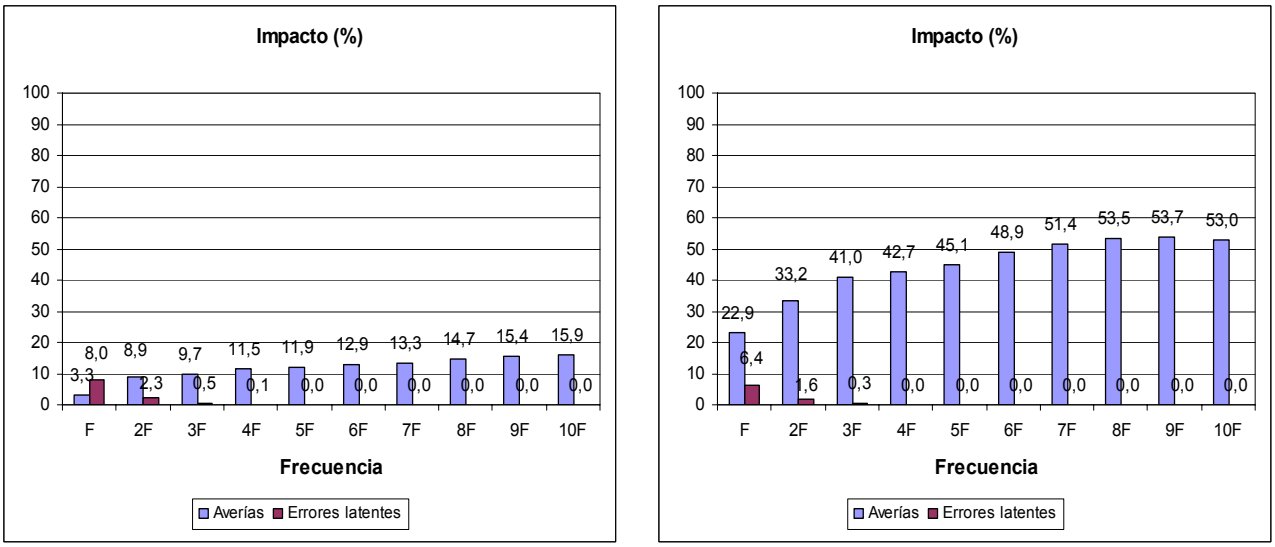


Figura 6.8: Influencia de la frecuencia de reloj en los porcentajes de averías y de errores latentes en elementos de almacenamiento. (a) Fallos simples. (b) Fallos múltiples.

6.5 Comparación de la influencia de la duración del fallo

En este apartado se describen las campañas de inyección realizadas con el objetivo de comparar la influencia de la duración de los fallos: permanentes, intermitentes y transitorios. Se han realizado un total de seis campañas, variando determinados parámetros. Las campañas realizadas son:

- Fallos permanentes simples en el espacio (dos simulaciones).
- Fallos permanentes múltiples en el espacio (dos simulaciones).
- Fallos transitorios simples en el espacio (seis simulaciones).
- Fallos transitorios múltiples en el espacio (seis simulaciones).
- Fallos intermitentes simples en el espacio (dieciocho simulaciones).
- Fallos intermitentes múltiples en el espacio (dieciocho simulaciones).

Éstos son los principales parámetros:

Sistema bajo estudio: 8051.

Carga de trabajo: algoritmo *bubblesort*.

Puntos de inyección:

- Elementos de almacenamiento (banco de registros y memoria RAM).
- Buses del sistema.

Técnica de inyección: órdenes del simulador

Simulaciones por experimento: 1000.

Multiplicidad espacial:

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

Modelos de fallo:

- Para fallos transitorios: *bit-flip* en elementos de almacenamiento, *pulse* en buses.
- 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.

Δt_{inj} (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 dentro de la campaña.

t_A (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 dentro de la campaña.

t_I (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 dentro de la campaña.

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

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

Los resultados más interesantes se pueden observar en las figuras 6.9 y 6.10. La gráfica 6.9 representa los resultados de las inyecciones en buses, mientras que la 6.10 muestra los resultados en elementos de almacenamiento. Para las figuras se han utilizado tres rangos distintos de tiempo de actividad para los fallos intermitentes, los mismos que para la duración de los fallos transitorios. Sin embargo, como el fallo *bit-flip* no perdura en el tiempo, no tiene sentido variar su duración en los fallos transitorios en elementos de almacenamiento.

En los resultados en buses se puede constatar que los fallos intermitentes provocan mayor porcentaje de averías para un rango concreto de tiempo de actividad que los transitorios, y menor que los permanentes. Este es el resultado lógico, ya que los fallos intermitentes se manifiestan de forma similar a una serie de fallos transitorios, aunque el origen y mecanismos sean distintos. Calculando la ratio $P_{Failures\ (intermittent)}$ / $P_{Failures\ (transient)}$ se obtienen valores entre 1.5 y 6, siendo los mayores valores para los fallos simples.

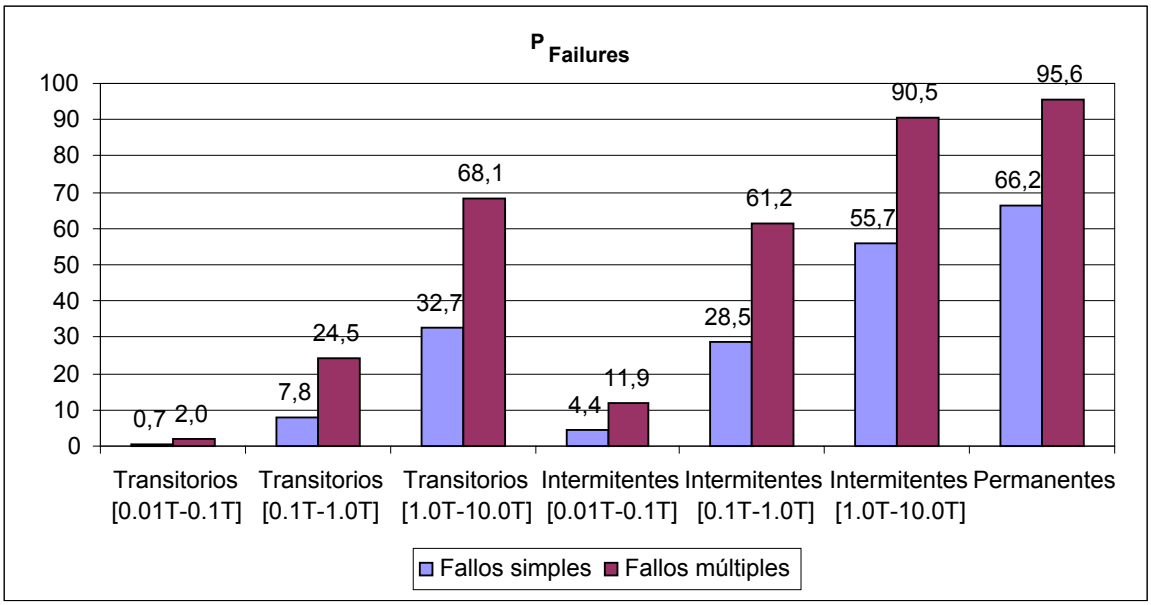


Figura 6.9: Influencia de la duración del fallo en el porcentaje de averías en buses.

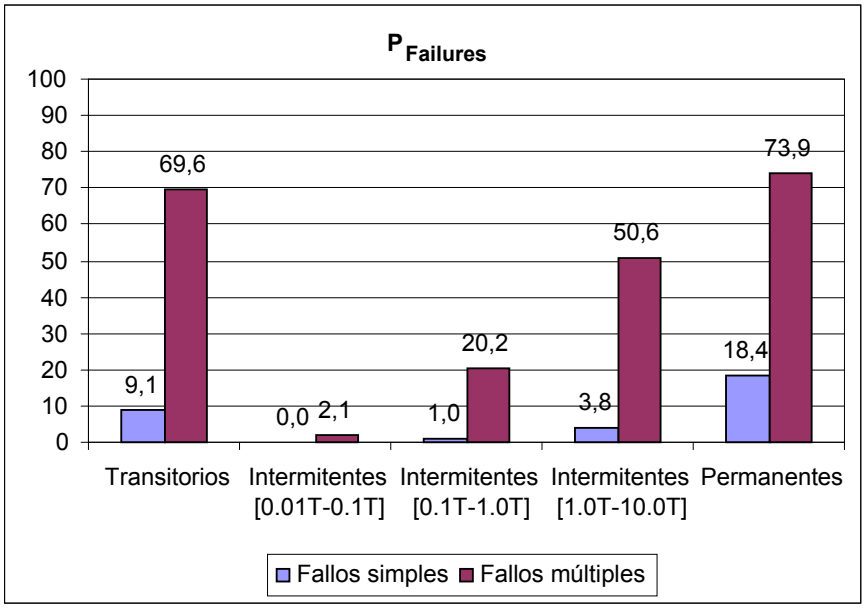


Figura 6.10: Influencia de la duración del fallo en el porcentaje de averías en elementos de almacenamiento.

Sin embargo, en los resultados en elementos de almacenamiento se da 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, tanto en elementos de almacenamiento como en buses, el mayor impacto corresponde a los fallos permanentes, debido a su duración infinita.

6.6 Resumen y conclusiones

En este capítulo se han presentado los resultados de los primeros experimentos de inyección aplicando algunos de los modelos de fallos intermitentes deducidos tras el estudio realizado en el capítulo 4.

En primer lugar se han definido las condiciones principales en que se han llevado a cabo los experimentos, como la herramienta de inyección (VFIT), el sistema bajo estudio (el microcontrolador 8051), la carga de trabajo utilizada (el algoritmo *bubblesort*) y las partes del sistema donde se inyectarán los fallos (los elementos de almacenamiento y los buses del sistema).

Tras definir diferentes conceptos básicos, se ha estudiado la influencia de distintos parámetros que afectan a los fallos intermitentes. Inicialmente se han estudiado los parámetros que afectan a las características de una ráfaga: los tiempos de actividad e inactividad y la longitud de la ráfaga (esto es, el número de activaciones). La primera conclusión es que, evidentemente, cuanto más tiempo esté activo el fallo, mayor es la probabilidad de que provoque una avería. Lo mismo sucede con la longitud de la ráfaga: a mayor número de activaciones del fallo dentro de una ráfaga, la probabilidad de que se produzca una avería

es mayor. Entre los factores que parecen no tener excesiva influencia en la aparición de averías o errores latentes tras un fallo intermitente, se encuentra el tiempo de inactividad. Parece lógico pensar que esto es así debido a que se está trabajando con un sistema que no es tolerante a fallos, y que si una activación del fallo intermitente provoca (o no) avería, es indiferente la distancia a la que se encuentre una nueva activación posterior. También parece lógico pensar que esta situación cambiará en sistemas tolerantes a fallos, en los que ese tiempo de separación podría servir para recuperarse del error producido. Estudiar la influencia de los distintos parámetros en sistemas tolerantes a fallos es parte del trabajo que se pretende realizar en la tesis.

A continuación se ha estudiado la importancia de otros factores como la multiplicidad espacial, el lugar de la inyección y la frecuencia de trabajo del sistema. Se ha constatado que la multiplicidad espacial tiene gran influencia en el porcentaje de averías. Este porcentaje es mayor para fallos múltiples que para fallos simples, ya que los fallos múltiples afectan simultáneamente a varios puntos físicos del sistema. Comparando la relación entre los porcentajes de averías producidos por fallos múltiples y simples, se observan valores de hasta 20, con mayores valores para los menores tiempos de actividad, y especialmente en las inyecciones realizadas en elementos de almacenamiento. Este resultado es significativo ya que la complejidad de los procesos de fabricación en las nuevas tecnologías submicrométricas puede favorecer la aparición de fallos múltiples intermitentes.

También tiene su incidencia el lugar de inyección. En los estudios realizados hasta ahora, los buses del sistema se han mostrado como puntos enormemente sensibles a la aparición de los fallos intermitentes. A la hora de diseñar un sistema tolerante a fallos, y en concreto pensando en los fallos intermitentes, será importante comprobar que los buses del sistema son tolerantes a este tipo de fallos. Sin embargo, el porcentaje de averías al inyectar fallos intermitentes en elementos de almacenamiento ha sido menor. Aunque seguramente esto pueda ser así en la realidad, la carga de trabajo puede afectar a los resultados ya que depende de las operaciones en memoria que se realicen, así como de la cantidad de memoria utilizada. Para poder generalizar estos resultados, como parte del trabajo futuro se desarrollarán y utilizarán nuevas cargas de trabajo, con mayor duración y utilización de la memoria.

Otro de los factores que tienen influencia en el porcentaje de averías es la frecuencia de trabajo del sistema bajo estudio. Se ha podido comprobar que, a mayor frecuencia de trabajo, mayor es la probabilidad de avería tras la aparición de un fallo intermitente. Esto es debido a que a mayores frecuencias la probabilidad de capturar un fallo en flancos activos de los componentes síncronos (como la memoria y los registros) aumenta. Dado que el constante aumento de las frecuencias de trabajo es una de las tendencias actuales en los sistemas VLSI, se refuerza la importancia del estudio de las causas, efectos y técnicas de mitigación de los fallos intermitentes.

Finalmente, 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 en buses provocan un mayor porcentaje de averías que los fallos transitorios, pero menor que los fallos permanentes. Sin embargo, se ha registrado un comportamiento anómalo al inyectar fallos transitorios en los elementos de almacenamiento. El fallo transitorio más característico en los elementos de almacenamiento es el *bit-flip*; a causa de las características de este fallo, y a la carga de trabajo utilizada, la duración *de facto* de los fallos transitorios es infinita, por lo que el efecto resultante es como si el fallo fuese permanente. Esto causa que los fallos transitorios sean más dañinos que los fallos intermitentes.

Como conclusión final, se puede afirmar 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. Por el contrario, el tiempo de inactividad parece no tener influencia en los resultados.

7

Conclusiones y trabajo futuro

7.1 Introducción

Para finalizar esta monografía, en este capítulo se resume el trabajo llevado a cabo y se esbozan las directrices de la investigación del autor dentro del GSTF para la consecución de su tesis doctoral. En primer lugar, se presenta un resumen del trabajo descrito en los capítulos previos. A continuación, se enumeran las publicaciones realizadas en relación con este documento. Finalmente se presentan las líneas de investigación abiertas y el trabajo futuro. No hay que perder de vista que este trabajo de investigación tutelado no es más que un primer paso hacia la consecución del doctorado y la presentación de la correspondiente tesis.

7.2 Resumen

7.2.1 Introducción y objetivos

Las tecnologías submicrométricas utilizadas para diseñar y fabricar los sistemas informáticos actuales, y los avances en las técnicas de integración, tienen como resultado una reducción del tamaño de los transistores empleados, una disminución de la tensión de alimentación y un incremento importante en la frecuencia de trabajo. Esto conlleva mejores prestaciones, pero se hace a costa de comprometer la confiabilidad de los circuitos integrados. Se prevé un aumento en la tasa de ocurrencia de fallos, y como consecuencia, los diseñadores de sistemas deben asumir que los circuitos integrados son más sensibles a distintos tipos de fallos.

Los fallos intermitentes han sido considerados habitualmente como el inicio de un proceso de desgaste, y por tanto como el preludio de un defecto físico irreversible que acaba en un fallo permanente. Sin embargo, en la actualidad se ha observado que existen fallos intermitentes que no conducen a fallos permanentes. Origen de estos fallos pueden ser la complejidad de los procesos de fabricación, que provoca residuos o variaciones en el proceso, junto con mecanismos de desgaste. La ocurrencia de determinados mecanismos físicos, donde se combina un *hardware* defectuoso con unas condiciones de trabajo concretas,

puede propiciar la aparición de fallos intermitentes que no conduzcan a un fallo permanente. Por otra parte, diversos estudios destacan la importancia que van a alcanzar los fallos intermitentes en las nuevas tecnologías submicrométricas.

Los errores inducidos por los fallos transitorios y los intermitentes se manifiestan de forma similar. La principal característica de los fallos intermitentes está en que éstos se activan repetidamente en el mismo lugar, y habitualmente aparecen en ráfagas. Además, cambiar el componente afectado elimina el fallo intermitente, lo que no sucede con los transitorios, que no pueden ser reparados. Los fallos intermitentes también se pueden activar o desactivar por cambios de temperatura, tensión de alimentación o frecuencia de trabajo (cambios conocidos como variaciones PVT, del inglés *Process, Voltage and Temperature*).

En relación con los fallos intermitentes, hay varias cuestiones difíciles de contestar: ¿dónde se producen los fallos intermitentes? ¿Cuándo se activan? ¿Cuántas veces se activa el fallo en una ráfaga? ¿Cómo se manifiesta el fallo a niveles de abstracción superiores? ... Para responder a estas cuestiones es importante entender los mecanismos físicos que tienen lugar en las tecnologías submicrométricas. Pero esta investigación es compleja, dependiente de la tecnología, y todavía está en una fase temprana de desarrollo.

El conocimiento de las causas de los fallos intermitentes no es suficientemente profundo, ya que se han publicado pocos trabajos donde se hayan observado fallos reales, o estudiado sus causas y mecanismos físicos.

Los fallos intermitentes han sido tradicionalmente olvidados a la hora de analizar las causas y mecanismos de los fallos reales que ocurren en los sistemas digitales. No ha sido hasta hace unos pocos años cuando los investigadores han prestado atención a estos tipos de fallos, y alertado de su creciente importancia. En la mayoría de los trabajos publicados se han monitorizado sistemas reales, observando los fallos y averías provocados. Tras su estudio, se han determinado las fuentes de errores y sus manifestaciones más frecuentes, y se ha descubierto que eran causados por fallos intermitentes. Sin embargo, en la literatura existente no se han encontrado trabajos en los que se haya intentado estudiar los efectos de los fallos intermitentes mediante simulación.

A la hora de utilizar modelos de fallos, una de las cuestiones fundamentales es su representatividad, es decir, la equivalencia del modelo de fallo obtenido con el fallo real, para el nivel de abstracción utilizado. El objetivo de este trabajo es determinar y utilizar modelos de fallos a nivel de puertas lógicas y de transferencia entre registros. La representatividad a nivel de *hardware* de los modelos de fallos utilizados para fallos permanentes y transitorios a este nivel de abstracción ha sido profusamente documentada, y los modelos definidos están bien establecidos por la comunidad científica. Sin embargo, para fallos intermitentes esto no ha sido así, y no han sido muchos los trabajos 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 uno permanente. Lo cierto es que con las nuevas tecnologías submicrométricas, la reducción de tamaños, el incremento de la frecuencia de funcionamiento y la necesidad de ajustar el consumo de energía, cada vez es más frecuente la aparición de fallos intermitentes que no dan lugar a fallos permanentes. Para ello, se debe tener en cuenta la influencia de los defectos de fabricación, como residuos o variaciones del proceso. El estudio de las causas, mecanismos y efectos de los fallos intermitentes es una línea de investigación abierta en la que es necesario profundizar.

En todo caso, con la investigación realizada hasta el momento, y los conocimientos actuales de las causas y mecanismos de los fallos intermitentes, ya se pueden deducir algunos modelos de fallos intermitentes. Determinar estos modelos, incluyendo su parametrización, es uno de los objetivos de este trabajo.

7.2.2 Conceptos básicos sobre tolerancia a fallos

La confiabilidad 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. 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 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 de propagación y de detección y recuperación de errores, y los coeficientes de cobertura de detección y recuperación de errores, de una manera más sencilla que con los métodos analíticos. Se puede realizar mediante inyección de fallos, esto es, introduciendo deliberadamente fallos en un modelo experimental o prototipo del sistema.

7.2.3 Técnicas de inyección de fallos

Las técnicas de inyección de fallos se pueden aplicar tanto sobre modelos como sobre prototipos. Sobre modelos se pueden inyectar fallos mediante simulación (sobre un modelo virtual en un computador) o mediante emulación (sobre un circuito real en el que se ha sintetizado el sistema). Sobre prototipos se puede hacer utilizando herramientas *hardware* o programas.

En el capítulo 3 se han descrito las técnicas más representativas, mostrando las variantes que han aparecido junto con los trabajos más relevantes, y se han analizado las ventajas e inconvenientes de cada una de ellas. En cada una de las distintas técnicas revisadas se ha hecho hincapié en los modelos de fallos que se pueden inyectar.

En particular, se ha hecho especial énfasis en las técnicas de inyección de fallos mediante simulación de modelos en VHDL, ya que son las que se han utilizado para la realización de los experimentos y obtención de resultados.

7.2.4 Modelos de fallos

En el capítulo 4 se ha presentado un resumen de los principales mecanismos de fallo en las tecnologías submicrométricas actuales, y se ha deducido un conjunto de modelos de fallos más amplio que los utilizados tradicionalmente en los experimentos de inyección de fallos (*stuck-at* para los fallos permanentes y *bit-flip* para los transitorios). Estos modelos se han deducido para los niveles lógico y de transferencia de registro (RTL) y para fallos permanentes, intermitentes y transitorios.

La reducción de las geometrías ha potenciado la aparición de nuevos mecanismos de fallos permanentes e intermitentes. Estos mecanismos afectan principalmente a la capa de óxido de los transistores, las conexiones y el encapsulado de los circuitos integrados. En lo que se refiere a los fallos permanentes, aparecen defectos irreversibles en los circuitos, producidos durante el proceso de fabricación o por desgaste, y se manifiestan básicamente a nivel electrónico como cortocircuitos y circuitos abiertos, que a nivel lógico se pueden modelar como fallos de tipo *stuck-at*, *open-line*, *stuck-open*, *indetermination*, *delay*, *short* y *bridging*.

En cuanto a los fallos transitorios, la reducción de las geometrías y de las tensiones de alimentación, junto con el aumento de las frecuencias de funcionamiento, han causado una notable reducción del efecto de los mecanismos naturales de enmascaramiento de los circuitos digitales y la aparición de efectos capacitivos que afectan a la respuesta temporal de los circuitos. También son fuente de errores las radiaciones, tanto internas como externas. Los modelos de fallo establecidos son *bit-flip* (en elementos de almacenamiento), *pulse* (en lógica combinacional), *indetermination* y *delay*.

Se espera también un aumento en la tasa de fallos intermitentes, causado por las nuevas tecnologías submicrométricas, en las que aparecen nuevos mecanismos físicos que provocan fallos intermitentes que eventualmente no terminan en fallo permanente. Por ello es necesario profundizar en sus causas y mecanismos.

La complejidad de los procesos de fabricación puede provocar residuos o variaciones en el proceso, que producen contactos intermitentes en las celdas de memoria. Las soldaduras defectuosas pueden producir cortocircuitos o circuitos abiertos intermitentes en líneas de interconexión. La deslaminación de la capa de barrera y la electromigración pueden incrementar la resistencia del material, produciendo retardos, cortocircuitos o circuitos abiertos de forma intermitente. La diafonía aparece cuando hay un acople capacitivo entre dos líneas adyacentes, y puede producir picos (en una línea que debería permanecer a valor constante), retardos (conmutación más lenta de lo esperado) o aceleraciones (conmutación más rápida de lo esperado). Daños en la capa de óxido, como el mecanismo denominado *soft breakdown*, pueden producir que la corriente de fuga del óxido de la puerta fluctúe, de forma que se obtienen valores erráticos del voltaje de salida que pueden llevar a valores lógicos indeterminados.

Los fallos intermitentes se ven potenciados, y se pueden activar o desactivar, por cambios de temperatura, tensión de alimentación o frecuencia de trabajo (variaciones PVT). Los modelos propuestos para estos fallos son *intermittent stuck-at*, *intermittent pulse*, *intermittent short*, *intermittent open*, *intermittent delay*, *intermittent speed-up* e *intermittent indetermination*.

Para finalizar, es preciso comentar que no se ha intentado hacer un estudio exhaustivo de los distintos mecanismos de fallos, nuevas tecnologías, etc. Simplemente, se ha querido hacer un resumen de modelos de fallos deducidos para los distintos tipos de fallo, haciendo énfasis en los nuevos modelos propuestos para fallos intermitentes.

7.2.5 Estudio de los efectos de los fallos intermitentes

Los modelos de fallos intermitentes deducidos en el capítulo 4 se han aplicado sobre el modelo en VHDL del microcontrolador 8051 ejecutando el algoritmo *bubblesort* y se han inyectado fallos en los elementos de almacenamiento y los buses del sistema.

El objetivo de los experimentos ha sido estudiar la influencia de distintos parámetros que afectan a los fallos intermitentes. Inicialmente se han estudiado los parámetros que afectan al comportamiento de una ráfaga: los tiempos de actividad e inactividad y el número de activaciones. Posteriormente se han estudiado otros factores como la multiplicidad espacial, el lugar de la inyección y la frecuencia de trabajo del sistema. Finalmente, se han comparado los efectos de los fallos intermitentes con los de los fallos transitorios y permanentes.

Como conclusión final, se puede afirmar 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. Por el contrario, el tiempo de inactividad parece no tener influencia en los resultados.

7.3 Publicaciones

Hasta la fecha, el trabajo realizado ha dado lugar a la publicación de cuatro artículos:

[Gracia08] "Analysis of the influence of intermittent faults in a microcontroller"; Joaquín Gracia-Morán, Luis-José Saiz-Adalid, Juan-Carlos Baraza-Calvo, Daniel Gil-Tomás, Pedro Gil-Vicente; Procs. 11 th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'08), Bratislava (Eslovaquia), Abril 2008.

[Saiz08] "Applying Fault Injection to Study the Effects of Intermittent Faults"; Luis-José Saiz-Adalid, Joaquín Gracia-Morán, Juan-Carlos Baraza-Calvo, Daniel Gil-Tomás, Pedro Gil-Vicente; Procs. 7th European Dependable Computing Conference (EDCC-7), Kaunas (Lituania), Mayo 2008.

[DGil08] "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", Daniel Gil-Tomás, Luis-José Saiz-Adalid, Joaquín Gracia-Morán, Juan-Carlos Baraza-Calvo, Pedro Gil-Vicente; Procs. IEEE International High Level Design Validation and Test Workshop 2008 (HLDVT'08), Incline Village (Nevada-USA), Noviembre 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.

[Gracia08] es un artículo que se presentó en el DDECS'08. En él se introducía el primer estudio acerca de los modelos de fallos intermitentes, y un análisis de los resultados obtenidos tras los primeros experimentos realizados inyectando en elementos de almacenamiento.

A continuación se presentó un *fast-abstract* en el EDCC-7 [Saiz08]. En esta publicación se profundizaba en el análisis de los efectos de los fallos intermitentes, ampliando las inyecciones a los buses.

[DGil08] es un artículo presentado en el HLDVT'08. En este trabajo se completaba el estudio de la influencia de los distintos parámetros que intervienen en los efectos de los fallos intermitentes, inyectando los modelos *intermittent stuck-at* en elementos de almacenamiento e *intermittent pulse* en buses.

Con todo lo investigado hasta ese momento, y con el camino a seguir para la consecución de la tesis doctoral ya esbozado, el autor presentó un artículo al *student-forum* del DSN 2009 [Saiz09]. En él se hacía un resumen de lo investigado hasta ahora, y se presentaban los próximos objetivos: estudiar los efectos de otros modelos de fallo, analizar sistemas tolerantes a fallos en presencia de fallos intermitentes para ver si son bien tolerados, y estudiar y proponer técnicas de mitigación adecuadas para tolerar los fallos intermitentes.

7.4 Trabajo futuro

De cara al futuro, quedan varias líneas de investigación abiertas. La primera es seguir estudiando las causas y mecanismos físicos que provocan los fallos intermitentes, para profundizar en su comprensión. Dado que el autor no es experto en física ni en electrónica de bajo nivel, esta parte del trabajo será de estudio bibliográfico, al igual que se ha hecho en esta monografía.

Con el estudio de las causas físicas hay que proponer nuevos modelos de fallo, o bien determinar cómo aplicar los ya deducidos en otros puntos del sistema, especialmente en la lógica combinacional. La evolución de la importancia de los fallos intermitentes hace pensar que éstos también pueden afectar a los circuitos combinacionales.

Como se ha podido observar en los experimentos realizados, los resultados pueden ser muy dependientes de la carga de trabajo a la que se somete al sistema. Por tanto, es fundamental aplicar distintas cargas de trabajo en los experimentos, con el fin de poder generalizar las conclusiones. Para ello se van a utilizar nuevas cargas.

Se ha comentado la posible dependencia de los resultados del sistema concreto bajo estudio. Por tanto, para generalizar los resultados y conclusiones obtenidas hay que realizar inyecciones en distintos sistemas. También hay que planificar experimentos en sistemas tolerantes a fallos, para estudiar si sus mecanismos de tolerancia son válidos. El mayor problema en esta cuestión es la poca disponibilidad de modelos en VHDL de sistemas tolerantes a fallos.

Una vez estudiados los problemas y sus causas, hay que buscar las soluciones. En primer lugar se estudiarán las técnicas de mitigación actuales, para ver cuáles podrían resultar útiles para tolerar los errores provocados por los fallos intermitentes. Finalmente, el objetivo es proponer nuevas técnicas de mitigación para una rápida detección y recuperación, y aplicarlas en el modelo VHDL de un sistema para comprobar la validez del esquema propuesto.

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.

[Aidemark03] "GOOFI: Generic Object-Oriented Fault Injection Tool"; J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson; Procs. 2003 International Conference on Dependable Systems and Networks (DSN 2003), pp. 668, San Francisco (EE.UU.), Junio 2003.

[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.

[Antoni02] "Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes"; L. Antoni, R. Leveugle, B. Fehér; Procs. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2002), pp. 405-413, Vancouver, Canadá, Noviembre 2002.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[Burgun96] "Serial Faul Emulation"; L. Burgun, F. Reblewski, G. Fenelon, J. Bariber, O. Lepape; Procs. 1996 Design Automation Coference (DAC'96), pp. 801-806, Las Vegas (Nevada, EE.UU.), Junio 1996.

[Cai03] "Transaction level modeling: an overview", L. Cai, D. Gajski; Proceedings of the 1st IEEE/ACM/IFIP international Conference on Hardware/Software Codesign and System Synthesis, Newport Beach, CA, USA, Octubre, 2003.

[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.

[CEC91] "Information Technology Security Evaluation Criteria (ITSEC), Provisional Harmonised Criteria, Version 1.2", Commision of the European Communities (CEC). 1991.

[Cha93] "A fast and accurate gate-level transient fault simulation environment"; H. Cha, E. M. Rudnick, G.S. Choi, J.H. Patel, R.K. Iyer; Procs. 23rd Symposium on Fault-Tolerant Computing Systems (FTCS-23), Toulouse, Francia, pp. 310-319, Junio 1993.

[Cha96] "A Gate-Level Simulation Environment for Alpha-Particle-Induced Transient Faults"; H. Cha, E.M. Rudnick, J.H. Patel, R.K. Iyer, G.S. Choi; IEEE Transactions on Computes, 45(11):1248-1256, Noviembre 1996.

[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.

[Cheng99] "Fault Emulation: A New Methodology for Fault Grading"; K.T. Cheng, S.Y. Huang, W.J. Dai; IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18(10):1487-1495, Octubre 1999.

[Choi92] "FOCUS: An experimental environment for fault sensitivity analysis"; G.S. Choi, R.K. Iyer; IEEE Transactions on Computers, 41(12):1515-1526, Diciembre 1992.

[Choi93] "Wear-out simulation environment for VLSI designs"; G.S. Choi, R.K. Iyer; Procs. 23rd International Symposium on Fault-Tolerant Computing (FTCS-23), pp. 320-329, Toulouse, Francia, Junio 1993.

[Civera01a] "FPGA-Based Fault Injection Techniques for Fast Evaluation of Fault Tolerance in VLSI Circuits"; P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante; Procs. 11th International Conference Field Programmable Logic and Applications (FPL2001), pp. 493-502, Belfast, Reino Unido, Agosto 2001.

[Civera01b] "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.

[Civera01c] "Exploiting FPGA-based Techniques for Fault Injection Campaigns on VLSI Circuits"; P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante; Procs. 2001 International Symposium on Defect and Fault Tolerance (DFT'01), pp. 250-258, San Francisco (California, EE.UU.), Octubre 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.

[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.

[Courtois92] "Sûreté de Fonctionnement Informatique. Evolutions 1987-1992. Tendances et Perspectives"; B. Courtois, M.C. Gaudel, J.C. Laprie, D. Powell; Rapport rédigé à la demande de la Direction Centrale de la Qualité du CNES, Diciembre 1992.

[Czeck90] "Effects of transient gate-level faults on program behaviour"; E.W. Czeck, D.P. Siewiorek; Procs. 20th International Symposium on Fault-Tolerant Computing (FTCS-20), pp. 236-243, Newcastle Upon Tyne, Reino Unido, Junio 1990.

[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. Available online at: http://www.laas.fr/dbench.

[deAndrés03] "Reconfiguración Dinámica de FPGAs para la aceleración de la Inyección de Fallos Basada en Simulación"; D. de Andrés, J. Albaladejo, L. Lemus; III Jornadas Sobre Computación Reconfigurable y Aplicaciones (JCRA2003), pp. 39-46, Madrid, España, Septiembre 2003.

[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.

[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, Universidad Politécnica de Valencia, Julio 1999.

[DGil00] "A Study of the effects of Transient Fault Injection into the VHDL Model of a Fault-Tolerant Microcomputer System"; D. Gil, J. Gracia, J.C. Baraza, P.J. Gil; Procs. 6th IEEE International On-Line Testing Workshop (IOLTW'2000), pp. 73-79, Palma de Mallorca, España, Julio 2000.

[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.

[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.

[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, DSN03, pp. 105-114. 2003. San Francisco, EE.UU.

[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.

[Fang93] "A Mechanism for Gate Oxide Damage in Nonuniform Plasmas"; S. Fang, J. McVittie; Procs. 31st International Reliability Physics Symposium (IRPS '93), pp. 13-17, Atlanta (Georgia, EE.UU.), Marzo 1993.

[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.

[Folkesson99] "Assessment and Comparison of Physical Fault Injection Techniques"; P. Folkesson; Thesis for the degree of Doctor of Philosophy, Department of Computer Engineering, Chalmers University of Technology, Göteborg (Suecia), 1999.

[Galiay80] "Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability"; J. Galiay, Y. Crouzet, M. Vergniault; IEEE Transactions on Computers, 29(6):527-531, Junio 1980.

[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.

[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.

[Gracia02] "Studying Hardware Fault Representativeness with VHDL Models"; J. Gracia, D. Gil, L. Lemus, P.J. Gil; XVII Conference on Design of Circuits and Integrated Systems (DCIS 2002), pp. 33-38, Santander, Noviembre 2002.

[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.

[Gunneflo89] "Evaluation of error detection schemes using fault injection by heavy-ion radiation"; U. Gunneflo, J. Karlsson, J. Torin; Procs. 19th International Symposium on Fault-Tolerant Computing (FTCS-19), pp. 218-227, Chicago (Illinois, EE.UU.), Junio 1989.

[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. 204-2 13.

[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.

[Hong96] "An FPGA-Based Hardware Emulator for Fast Fault Emulation"; J.H. Hong, S.A. Hwang, C.W. Wu; Procs. 1996 Midwest Symposium on Circuit and Systems, Ames (Iowa, EE.UU.), Agosto 1996.

[Hu99] "A Unified Gate Oxide Reliability Model"; C. Hu, Q. Lu; Procs. 39th International Reliability Physics Symposium (IRPS '99), pp. 47-52, San Diego (California, EE.UU.), Marzo 1999.

[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.

[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.

[Iyer86] "A measurement-based model for workload dependence of CPU errors"; R.K. Iyer, D. Rosseti; IEEE Transactions on Computers, 35(6):511-519. Junio 1986.

[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.

[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.

[Jones87] "Line Width Dependence of Stresses in Aluminium Interconnect"; R.E. Jones; Procs. 25th International Reliability Physics Symposium (IRPS'87), pp. 9-14, San Diego (California, EE.UU.), Abril 1987.

[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 Sûreté 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.

[Karlsson90] "Transient fault effects in the MC6809E 8-bit microprocessor: A comparison of results of physical and simulated fault injection experiments"; J. Karlsson; Technical Report 96, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Suecia, 1990.

[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

[Kilty95] "VHDL/VITAL Fault Simulation"; P. Kilty; Procs. 1995 VHDL Users Group/VHDL International Users' Forum – Fall Conference (VUG/VIUF FALL 95), Boston (Massachusetts, EE.UU.), Octubre 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.

[Laprie85] "Dependable Computing and Fault-Tolerance: Concepts and Terminology"; J.C. Laprie; Procs. 15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15), pp. 2-11, Ann Arbor (Michigan, EE.UU.), Junio 1985.

[Laprie92] "Dependability, Basic Concepts and Terminology"; J.C. Laprie; Springer-Verlag, 1992.

[Leveugle00] "Fault Injection in VHDL Descriptions and Emulation"; R. Leveugle; Procs. 2000 International Symposium on Defect and Fault Tolerance (DFT'00), pp. 414-420, Yamanashi, Japón, Octubre 2000.

[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.

[Li95] "Cellular Automata for Efficient Parallel Logic and Fault Simulation"; Y.L. Li, C.W. Wu; IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 14(6):740-749, Junio 1995.

[Li97] "VLSI Design of a Cellular-Automata Based Logic and Fault Simulator"; Y.L. Li, Y.C. Lai, C.W. Wu; Procs. 1997 National Science Council Part A: Physical Science and Engineering, 21:189-199, Taipei (Taiwan, China), Mayo 1997.

[Lima01] "On the Use of VHDL Simulation and Emulation to Derive Error Rates"; F. Lima, S. Rezgui, L. Carro, R. Velazco, R. Reis; Procs. 6th European Conference on Radiation and its Effects on Components and Systems (RADECS 2001), Grenoble, Francia, Septiembre 2001.

[Lomelino86] "Error Propagation in a Digital Avionic Processor – A Simulation-Based Study"; D. Lomelino, R.K. Iyer; Procs. 1986 Real-Time Systems Symposium (RTSS'86), pp. 218-225, New Orleans (Louisiana, EE.UU), Diciembre 1986.

[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.

[MC8051] 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.

[McVittie96] "Plasma Charging Damage: An Overview"; J. McVittie; Procs. 1st International Symposium on Plasma Process-Induced Damage (P2ID), pp. 7-20, Santa Clara (California, EE.UU.), Mayo 1996.

[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.

[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.

[Oates93] "Electromigration in Stress-Voided Al Alloy Metallizations for Submicron IC Technologies"; A. Oates; Procs. 31st International Reliability Physics Symposium (IRPS '93), pp. 297-303, Atlanta (Georgia, EE.UU.), Marzo 1993.

[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.

[Parreira03] "A Novel Approach to FPGA-Based Hardware Fault Modeling and Simulation"; A. Parreira, J.P. Teixeira, M. Santos; Procs. 6th International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2003), pp.17-24, Poznań, Polonia, Abril 2003.

[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.

[Pol96] "Short Loop Monitoring of Metal Step Coverage by Simple Electrical Measurements"; J.A. van der Pol, E.R. Ooms, H.T. Brugman; Procs. 34th International Reliability Physics Symposium (IRPS'96), pp. 148-155, Dallas (Texas, EE.UU.), Abril-Mayo 1996.

[Pradhan86] "Fault-Tolerant Computing Theory and Techniques"; D.K. Pradhan; Prentice-Hall, 1986.

[Pradhan96] "Fault-Tolerant Computer System Design"; D.K. Pradhan; Prentice-Hall; 1996.

[Rimén92] "A Study of the Error Behavior of a 32-bit RISC Subjected to Simulated Fault Injection"; M. Rimén, J. Ohlsson; Procs. IEEE International Test Conference; pp. 696-704, Baltimore (Maryland, EE.UU.), Septiembre 1992.

[Rimén93] "A study of the error behaviour of a 32 bit RISC subjected to simulated transient fault injection"; M. Rimén, J. Ohlsson; PFCS2 Report, pp. 445-460, Septiembre 1993.

[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.

[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.

[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.

[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.

[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.

[Shaw01] "Accurate CMOS Bridge Fault Modeling With Neural Network-Based VHDL Saboteurs"; D.B. Shaw, D. Al-Khalili, C.N. Rozon; Procs. 2001 International Conference on Computer-Aided Design (ICCAD'01), pp. 531-536, San Jose (California, EE.UU.), Noviembre 2001.

[Shen85] "Inductive Fault Analysis of MOS Integrated Circuits"; J.P. Shen, W. Maly, F.J. Ferguson; IEEE Design and Test of Computers, 2(4):13-26, Diciembre 1985.

[Shivakumar02] "Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic"; P. Shivakumar, M. Kistler, S. Keckler, D. Burger, L. Alvisi; Procs. 2002 International Conference on Dependable Systems and Networks (DSN 2002), pp. 389-402, Washington (D.C., EE.UU.), Junio 2002.

[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.

[Siewiorek82] "The Theory and Practice of Reliable System Design"; D.P. Siewiorek, R.S. Swarz; Digital Press, Bedford (Massachusetts, EE.UU.), 1982.

[Siewiorek92] "Reliable Computer Systems. Design and Evaluation"; D.P. Siewiorek, R.S. Swarz; 2ª ed., Digital Press, 1992.

[Siewiorek94] "Reliable Computer Systems. Design and Evaluation"; D. P. Siewiorek; 3rd ed., Digital Press. 1994.

[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.

[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.

[Sylvester99] "Rethinking Deep-Submicron Circuit Design"; D. Sylvester, K. Keutzer; IEEE Computer, 32(11):25-33, Noviembre 1999.

[Tezaki90] "Measurement of Three Dimensional Stress and Modeling of Stress-Induced Migration Failure in Aluminium Interconnects"; A. Tezaki, T. Mineta, H. Egawa, T. Noguchi; Procs. 28th International Reliability Physics Symposium (IRPS '90), pp. 221-229, New Orleans (Louisiana, EE.UU.), Marzo 1990.

[Turner82] "A New Failure Mechanism: Al-Si Bond Pad Whisker Growth During Lifetest"; T. Turner, R.D. Parsons; IEEE Transactions on Component, Hybrids and Manufacturing Technology, 5:431-435, 1982.

[Vargas99] "Design of SEU-Tolerant Processors for Radiation-Exposed Systems"; F. Vargas, A. Amory; 4th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'99), pp. 110-116, San Diego, (California, EE.UU.), Noviembre 1999.

[Vargas00a] "Estimating Circuit Fault-Tolerance by Means of Transient-Fault Injection in VHDL"; F. Vargas, A. Amory, R. Velazco; Procs. 6th IEEE International On-Line Testing Workshop (IOLTW'2000), pp. 67-72, Palma de Mallorca, España, Julio 2000.

[Vargas00b] "Fault-Tolerance in VHDL Description: Transient-Fault Injection & Early Reliability Estimation"; F. Vargas, A. Amory, R. Velazco; 1st Latin-American Test Workshop (LATW'00), pp. 29-35, Rio de Janeiro, Brasil, Marzo 2000.

[Velazco01a] "Upset-like Fault Injection in VHDL Descriptions: A Method and Preliminary Results"; R. Velazco, R. Leveugle, O. Calvo; TIMA Research Report ISRN TIMA-RR-01/10-6-FR, TIMA Laboratory, 2001.

[Velazco01b] "Upset-like Fault Injection in VHDL Descriptions: A Method and Preliminary Results"; R. Velazco, R. Leveugle, O. Calvo; Procs. 2001 IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2001), pp. 259-267, San Francisco, EE.UU., Octubre 2001.

[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.

[Walker00] "Modelling the Wiring of Deep Submicron ICs"; M.G. Walker; IEEE Spectrum, 27(3):65-71, Marzo 2000.

[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.

[Ying06] "A MDA based SoC modeling approach using UML and SystemC", W. Ying, Z. Xue-Gong, Z. Bo, L. Liang, and P. Cheng-Lian; The Sixth IEEE International Conference on Computer and Information Technology (CIT), pp. 245–245, 2006.

[Yu0l] "A Perspective on the State of Research on Fault Injection Techniques", Y. Yu; Research Report, Mayo 2001.

[Yu03] "Fault Injection Techniques. A Perspective on the State of Research"; Y. Yu, B.W. Johnson; en Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Edited by Alfredo Benso and Paolo Prinetto, Kluwer Academic Publishers, pp. 7-39, Octubre 2003.

[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.

Anexo I: Publicaciones

Este anexo incluye las publicaciones realizadas hasta la fecha:

[Gracia08] "Analysis of the influence of intermittent faults in a microcontroller"; Joaquín Gracia-Morán, Luis-José Saiz-Adalid, Juan-Carlos Baraza-Calvo, Daniel Gil-Tomás, Pedro Gil-Vicente; Procs. 11 th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'08), Bratislava (Eslovaquia), Abril 2008.

[Saiz08] "Applying Fault Injection to Study the Effects of Intermittent Faults"; Luis-José Saiz-Adalid, Joaquín Gracia-Morán, Juan-Carlos Baraza-Calvo, Daniel Gil-Tomás, Pedro Gil-Vicente; Procs. 7th European Dependable Computing Conference (EDCC-7), Kaunas (Lituania), Mayo 2008.

[DGil08] "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", Daniel Gil-Tomás, Luis-José Saiz-Adalid, Joaquín Gracia-Morán, Juan-Carlos Baraza-Calvo, Pedro Gil-Vicente; Procs. IEEE International High Level Design Validation and Test Workshop 2008 (HLDVT'08), Incline Village (Nevada-USA), Noviembre 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.

Analysis of the influence of intermittent faults in a microcontroller

J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) - Departamento de Informática de Sistemas y Computadores (DISCA)
Universidad Politécnica de Valencia, Spain
e-mail: {jgracia, ljsaiz, jcbaraza, dgil, pgil}@disca.upv.es

***Abstract*— Nowadays, new submicron technologies have allowed increasing processors performance while decreasing their size. However, as a side effect, their reliability has been negatively affected. Although mainly permanent and transient faults have been studied, intermittent faults are expected to be a big challenge in modern VLSI circuits. Usually, intermittent faults have been assumed to be the prelude of permanent faults. Currently, intermittent faults due to process variations and residues have grown, being necessary to study their effects.**

The objective of this work has been to analyse the impact of intermittent faults, taking advantage of the power of the simulation-based fault injection methodology. Using as background faults observed in real computer systems, we have injected intermittent faults in the VHDL model of a microcontroller. The controllability and flexibility of VHDL-based fault injection technique has allowed us to do a detailed analysis of the influence of some parameters of intermittent faults. We have also compared the results obtained with the impact of transient and permanent faults.

***Index Terms*— VHDL-based fault injection, Intermittent faults.**

I. Introduction

During last years, advances in integration techniques have allowed rising microprocessors operating frequency as well as reducing their size and power voltage (V_{DD}), achieving in this way a greater productivity. Nevertheless, these advances have a negative impact on reliability. As the new submicron technologies shrink device sizes, the rate of occurrence of faults increases [1]. The consequence is that designers have to deal with an increasing number of different fault types due to manufacturing defects.

Usually, only the effects of permanent and transient faults have been studied [2][3]. Permanent faults are provoked by irreversible physical defects due to manufacturing defects and wearout mechanisms. Transient faults are commonly generated by environmental conditions like electromagnetic interferences and cosmic radiation. The increase on integration scale has caused that transient faults normally observed in space are now common at sea level [4][5]. It is expected that transient faults will have a greater influence than permanent faults in the new VLSI circuits [6].

Intermittent faults are another type of fault not considered normally. It is foreseen that intermittent faults will have a great impact in deep submicron technologies, together with transient faults. The complexity of the manufacturing process (that provokes residuals and process variations) and some wearout mechanisms can be the origin of such faults [6][7][8].

Errors induced by transient and intermittent faults manifest in a similar way, but the last ones are activated repeatedly in the same place, and they usually are grouped in bursts [9]. On the other hand, replacement of the affected part eliminates an intermittent fault, while transients cannot be fixed by repair. Additionally, intermittent faults may be deactivated or activated by temperature, voltage and frequency changes [10].

So, intermittent faults are a big challenge in modern VLSI circuits. Our objective in this work is to study the impact of intermittent faults, and compare their consequences to those provoked by permanent and transient faults.

A common method to study experimentally the dependability parameters of a microprocessor is Fault Injection [11]. This technique allows a controlled introduction of faults in the system, not being necessary to wait a long time for logging the apparition of real faults.

Fault injection techniques can be classified in three main categories [3]: physical (or Hardware Implemented Fault Injection, HWIFI), software implemented (SWIFI) and simulation-based.

Simulation-based fault injection is a useful experimental way to evaluate the dependability of a system during the design phase. An early diagnosis allows saving costs in the design process, avoiding redesigning in case of error, and thus reducing time-to-market. We have used VHDL-based fault injection due to its flexibility, as well as the high observability and controllability of all the modelled components [12]. Moreover, VHDL is a widely utilized hardware description language in academic and industrial realms [13].

So far, very few tries have been done in order to study the effects of intermittent faults by fault injection. In most of the works issued, real systems were monitored, and faults and failures produced were observed to determine the most frequent sources of errors and their manifestation [10].

Our intention with this work is to study systematically and exhaustively the impact of the parameters of intermittent faults on the behaviour of the system, as well as to compare to the effects of permanent and transient faults, taking advantage of the power of fault injection.

Using as background the existing information about observed and logged errors in real systems, we have generated the adequate fault models to inject, according to the abstraction level of the system under study.

Intermittent faults have been injected into the VHDL model of a typical microcontroller and their effects have been compared with permanent and transient faults.

The organization of the paper is as follows. In Section II we describe the intermittent fault models injected. Section III depicts the fault injection environment. In Section IV, the fault injection experiments are explained, and the results obtained are discussed in Section V. Finally, some conclusions and a proposal of future work are provided in Section VI.

II. Intermittent Fault Models

When performing fault injection campaigns, one of the most important questions is fault representativeness, i.e., the equivalence of the supported fault model with respect to actual faults. When injecting faults in a system model, the fault models applied must be suitable to the abstraction level of the system model. Our objective in this work is to use fault models at logic and RT levels.

Hardware fault representativeness for transient and permanent faults has been extensively studied [2][14][15]. Nevertheless, and as far as we know, the causes and mechanisms of intermittent faults in new submicron technologies have not been studied so much.

One possibility is to use similar models to those for permanent faults. In fact, in a wearout process, faults initially may appear intermittently but eventually result in permanent faults [2][16].

However, the introduction of new submicron technologies makes necessary to keep in mind also the influence of manufacturing defects, like process variations and residuals [6][17]. A deeper study is necessary, being this an open line of research.

For these reasons, we have decided in this paper to select a set of intermittent faults frequently observed in real computer systems by means of fault logging. Then, we have extrapolated their effects to fault models applicable to the logic/RT abstraction levels. In this way, we can inject and simulate these faults into VHDL models.

Figure 1 shows the main causes and mechanisms of the intermittent faults considered [10], as well as the associated fault models.

Bursts of Single Bit Errors (or SBEs) have been observed in memory [18]. Failure analysis found that polymer residues led to an intermittent contact. The fault model chosen to represent a SBE in memory and register cells is an *intermittent stuck-at*, meaning that the value of the cell changes intermittently between the two logical values. The origin of the fault lies in an unstable hardware. This is not related to an external environmental effect (e.g. cosmic radiation). Thus, it is not an intermittent bit-flip, so we have called it *intermittent stuck-at*.

Bursts of SBEs have also been observed in data buses [18]. Failure analysis revealed that the source of errors were solder intermittent contacts. We have associated this mechanism to the following fault models [12]: *intermittent pulses*, *short* and *open* in bus lines.

Also, timing failures may occur due to propagation delays over the interconnection lines. For instance, barrier layer material delamination and electromigration lead to a higher resistance and, as a result, to timing violations [19]. Moreover, they can provoke *intermittent short/open* faults in metal lines. Crosstalk delays may occur when adjacent signals switch in opposite directions, and are related to capacitive effects in adjacent metallic lines. The fault models assigned to these mechanisms are *intermittent delay* and *short/open* in interconnection lines and buses [12].

Another type of causes and mechanisms of intermittent faults are due to oxide defects and wearout. Soft breakdown belongs to this class. In this case, the leakage current of the gate oxide fluctuates in time, without inducing the thermal damage [20]. This produces erratic fluctuations of the minimum supply voltage, that can be modelled with the *intermittent indetermination* fault model [12].

In the future, we want to carry out a deeper study of the fault mechanisms related with wearout processes and process variations in new submicron technologies.

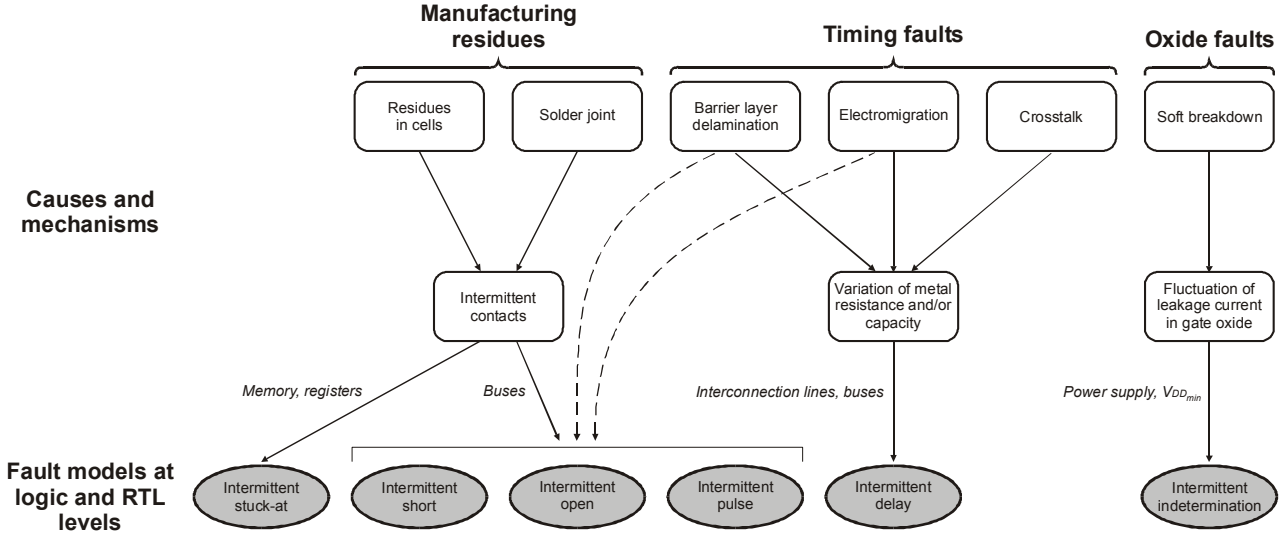


Figure 1. Some representative mechanisms and models for intermittent faults

III. The Fault Injection Environment

The *Grupo de Sistemas Tolerantes a Fallos* (GSTF) have developed a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [21], that runs on PC computers (or compatible) under Windows® and is model-independent. Although it admits VHDL models at any abstraction level, it has been mainly used on models at logic and RT levels.

With VFIT it is possible to inject faults applying simulator commands, saboteurs and mutants techniques [3][2].

VFIT can inject a great range of faults according to various aspects. With regard to the fault duration, it can inject permanent, transient and intermittent faults. In relation to the number of faults injected, it is possible to inject faults in a single location (i.e. Single Bit Faults or SBFs) or in multiple locations, (that is, Multiple Bit Faults, or MBFs), either adjacent and/or non adjacent. On the other hand, independently of the number of fault targets, these faults can be single or multiple in time domain. That is, multiple occurrences of the fault can be injected along the simulation time.

When applied to models at logic and RT levels, VFIT uses a wide set of fault models that try to be representative of deep submicron technologies, as it can be seen in Table I. This table shows models of transient, permanent and intermittent faults. As commented in Section II, intermittent fault models are not still well established, and are an open research subject. Also in Section II, we have proposed some models for frequent intermittent faults observed in real systems. We have reflected these fault models in Table I, according to the injection technique that can inject them.

The output of an injection experiment is a set of tables whose values depend on the objective of the injection experiment. In case of an error syndrome analysis, output tables contain among other values: propagation latencies, percentages of propagated errors and percentages of failures. In case of performing a dependability validation, tables show propagation, detection and recovery latencies, percentages of propagated, detected, and recovered errors, detection and recovery coverages, and failure percentages.

IV. Fault Injection Experiments

A. Notation

As mentioned above, intermittent faults manifest in bursts. That is, when a fault appears, it does not activate only once. Instead, it activates and deactivates repeatedly several times until the fault finally disappears. So, to inject intermittent faults, the following parameters must be configured: the number of activations in the burst (that we will call the *burst length*, or L_{Burst}), the duration of each activation (we will refer to it as the *activity time*, or t_A), and the separation between two consecutive activations (we will name it as the *inactivity time*, or t_I). Figure 2 illustrates this explanation.

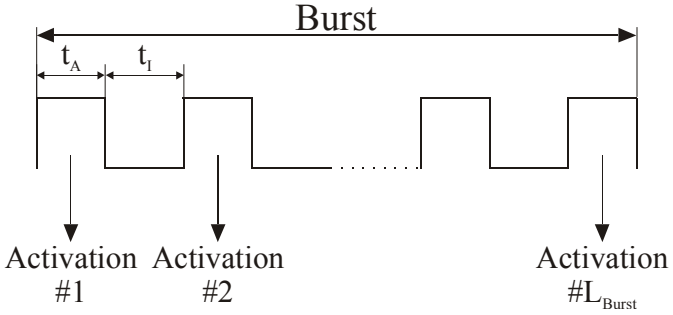


Figure 2. Description of the main elements of a burst.

The following terms and notation are used in the remaining of this paper:

Fault simulation	A simulation of the model of the system under study in presence of faults;
Injection experiment	A set of fault simulations configured with the same injection parameters;
Injection campaign	A set of injection experiments in which a number of injection parameters can vary;
t_A	Activity time;
t_I	Inactivity time;
L_{Burst}	Burst length;
t_{inj}	Injection instant;
$N_{Injected}$	Number of faults injected;
$N_{Propagated}$	Number of propagated errors, defined in this campaign as the injected faults that propagate to the microprocessor registers and memory;
$N_{Failures}$	Number of failures, defined as the number of propagated errors that provoke failures. A failure is produced when the result is erroneous at the end of the simulation time;
N_{Latent}	Number of latent errors. A latent error is a propagated error that does not provoke a failure;
$N_{Non\_effective}$	Number of non effective faults. It is the number of faults injected that have not produced either a failure or a latent error;
$P_{Failures}$	Percentage of failures;
P_{Latent}	Percentage of latent errors;
$P_{Non\_effective}$	Percentage of non effective errors.

TABLE I. FAULT MODELS INJECTED BY VFIT

Injection technique	Transient faults	Permanent faults	Intermittent faults
Simulator commands	Pulse*, Bit-flip**, Indetermination, Delay	Stuck-at (0,1), Indetermination, Open, Delay	Intermittent stuck-at (0,1), Pulse***, Indetermination, Open
Saboteurs	Pulse*, Bit-flip**, Indetermination, Delay	Stuck-at (0,1), Indetermination, Open, Delay, Short, Bridging, Stuck-open	Intermittent stuck-at (0,1), Pulse***, Indetermination, Open, Delay, Short
Mutants	Syntactical changes	Syntactical changes	Not yet defined

* In combinational logic and interconnection lines. Represents a Single Event Transient (SET)
** In storage elements (registers and memory). Represents a Single Event Upset (SEU)
*** Sequence of pulses due to intermittent fault mechanisms

B. Experiment set-up

We have scheduled two injection campaigns for intermittent faults, where the injection targets are the storage elements, that is, the register bench and the memory. The injection campaigns differ on the number of injection targets. In the first campaign we have injected SBFs (that is, only one intermittent fault in a single bit location). In the second one, we have injected MBFs in multiple adjacent and non-adjacent locations. The aim of these injection campaigns is to analyse the influence of the target size (single or multiple) and the activity time (t_A).

On the other hand, we have also injected transient and permanent faults in order to compare them to intermittent faults. The different campaigns have been carried out in a VHDL model of the 8051 microcontroller [22] running the Bubblesort sorting algorithm as a workload. Although the chosen system has not a deep submicron technology, the methodology used can be generalised to more complex microprocessors/microcontrollers.

The most relevant injection parameters are the following:

1) *Injection targets:* The register bench and the memory.
2) *Injection technique:* Simulator commands.
3) *Number of faults per experiment:* 1000 single or multiple adjacent and non adjacent faults per experiment[1].
4) *Fault models:*
   - For transient faults: Bit-flip;
   - For permanent faults: Stuck-at(0,1), Open, and Indetermination;
   - For intermittent faults: Intermittent stuck-at(0,1) (see Section II).
5) *Fault types and duration:*
   - Transient single and multiple in space[2];
   - Permanent single and multiple in space;
   - Intermittent single and multiple in space, with
     - an activity time (t_A) defined according to a Uniform distribution function in the ranges [0.01T, 0.1T], [0.1T, 1.0T], and [1.0T, 10.0T], where T is the CPU clock cycle;
     - an inactivity time (t_I) defined according to a Uniform distribution function in the ranges [0.01T, 0.1T], [0.1T, 1.0T], and [1.0T, 10.0T].
     - a burst length defined according to a Uniform distribution function in the range [2, 9].

The objective of the campaigns is to inject intermittent faults where bursts have different shapes and lengths.

In order to check the effects of the intermittent faults, we have calculated in all cases the following data:

- The percentage of failures:

$$P_{Failures} = \frac{N_{Failures}}{N_{Injected}} \times 100$$

  A failure is detected by comparing the golden run with the faulty trace, and checking disparity in the result registers.
- The percentage of latent errors:

$$P_{Latent} = \frac{N_{Latent}}{N_{Injected}} \times 100$$

  where N_{Latent} can be obtained as ($N_{Propagated} - N_{Failures}$) or as ($N_{Injected} - N_{Failures} - N_{Non\_effectivet}$).
  A latent error is detected by checking changes in the content of registers or memory at the end of simulation. Checked registers are not the result registers.

[1] This implies that, for instance, in a given fault simulation, faults might be injected simultaneously in bits 3 and 2 of the 37th memory byte, and in bits 7, 6, and 5 of register #6.

[2] A multiple fault in space are simultaneous faults produced in various targets of the model.

V. RESULTS

Table II shows the percentages of failures, latent errors and non effective faults for single intermittent faults, while Table III shows the same percentages for multiple intermittent faults. In both tables, we have varied the duration of the pulses of the burst, i.e., the activity time. When increasing the activity time, we can observe an augment of the percentage of failures. This is a logical result if we think that a longer duration of the intermittent pulses must have a bigger impact on the system.

In relation to the percentage of latent errors, a decrease is observed, because they are propagated errors that do not provoke failures. As $N_{Propagated} = N_{Failures} + N_{Latent}$, an increase in the first addend causes a drop in the second, since they are exclusive cases. We can also verify that $P_{Failures} + P_{Latent} + P_{Non\_effective} = 100\%$, as $N_{Failures} + N_{Latent} + N_{Non\_effective} = N_{Injected}$.

TABLE II. RESULTS FOR SINGLE INTERMITTENT FAULTS

% \ t_A	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	0.1 %	0.4 %	3.9 %
Latent errors	41.9 %	8.9 %	0 %
Non effective	58.0 %	90.7 %	96.1 %

TABLE III. RESULTS FOR MULTIPLE INTERMITTENT FAULTS

% \ t_A	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	1.7 %	15.5 %	53.0 %
Latent errors	43.0 %	6.4 %	0 %
Non effective	55.3 %	78.1 %	47.0 %

Figure 3 and Figure 4 show the influence of the spatial multiplicity of faults (Single vs. Multiple faults) in the percentage of failures and latent errors. We show results for different activity times. We can see in Figure 3 that multiple faults provoke a notably bigger number of failures than single faults. This is an expected behaviour, because multiple faults affect simultaneously various physical zones of the system (in this case memory and register bench). We can observe also that the difference between $P_{Failure}$ for multiple and single faults grows with the activity time.

About latent errors, Figure 4 does not reflect a substantial difference between single and multiple faults. This implies that additional areas affected by multiple faults have provoked failures in most cases, and not latent errors. Nevertheless, we think that the dependence on the workload is very strong and it is difficult to generalize the results.

Figure 5 and Figure 6 compare the impact of intermittent faults respect to permanent and transient faults. We have also represented the influence of the spatial multiplicity (single vs. multiple faults).

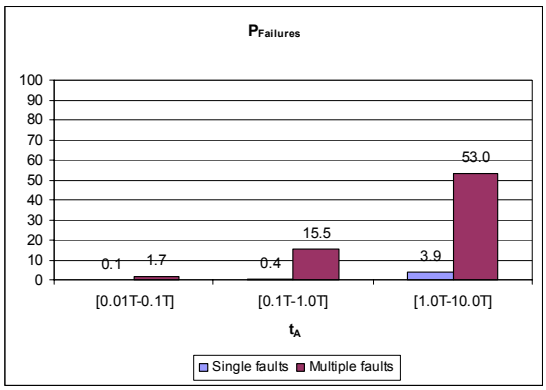


Figure 3. Influence of the spatial multiplicity of faults in the percentage of failures.

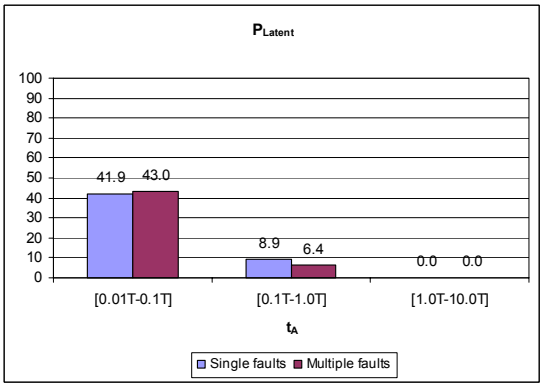


Figure 4. Influence of the spatial multiplicity of faults in the percentage of latent errors.

As expected, Figure 5 shows that permanent faults provoke a bigger percentage of failures than transient faults, because permanent faults have an indefinite duration. On the other hand, we can see that the impact of intermittent faults is lower than transient faults, even for the bigger activity time. We expected a bigger impact of intermittent faults, because they are actually a sequence of transient faults that form the burst. However, a more detailed analysis of the injected fault models and the workload allows establishing the cause of this anomalous behaviour. The characteristics of the used workload provokes a low rate of overwrite operations in the memory cells affected by transient faults (bit-flips), causing a *de facto* indefinite duration of these faults, remaining in this way stored permanently.

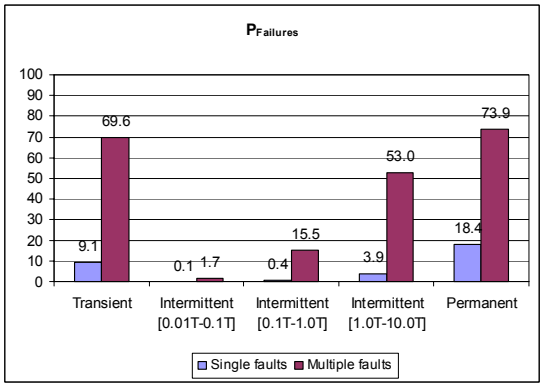


Figure 5. Comparison of the percentage of failures provoked by transient, intermittent and permanent faults.

Figure 6 shows that intermittent faults have provoked a bigger percentage of latent errors than transient and permanent faults. This type of faults propagates into memory cells or registers, but they do not provoke failures, although the system remains contaminated.

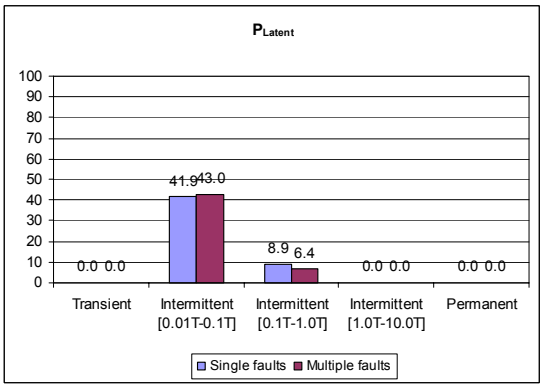


Figure 6. Comparison of the percentage of latent errors provoked by transient, intermittent and permanent faults.

Finally, Figure 7 and Figure 8 show the influence of the inactivity time, that is, the temporal separation between the pulses of the burst (see Section IV). No significant differences between single and multiple faults are observed when varying the inactivity time. The separation of the pulses of the burst does not provoke a noteworthy variation in $P_{Failure}$ or P_{Latent} (assuming that all pulses of the burst are present). However, the separation between pulses is decisive for the system to recover when detection and recovery mechanisms are introduced in the system. These mechanisms should be as fast as possible [10].

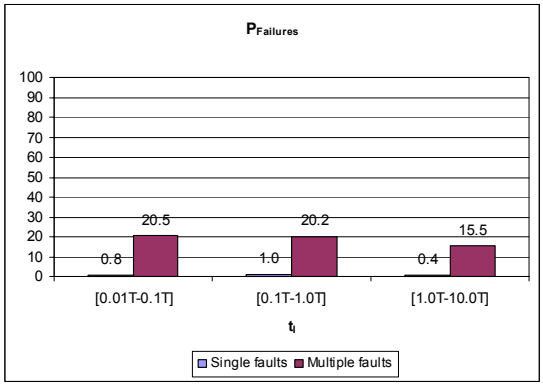


Figure 7. Percentage of failures respect to the inactivity time.

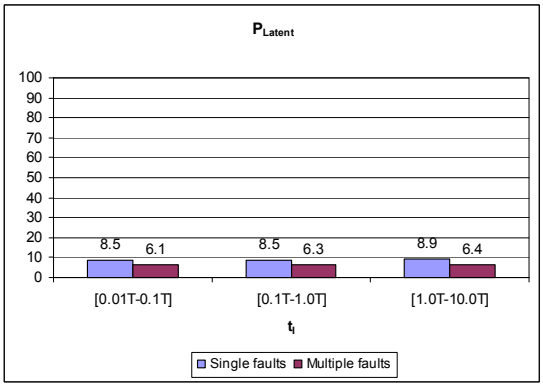


Figure 8. Percentage of latent errors respect to the inactivity time.

VI. Conclusions and Future Work

In this work, we have presented an exhaustive study of the effects of intermittent faults on the behaviour of a commercial microcontroller. The method used has been fault injection. In order to choose the intermittent fault models to inject, we have selected some faults that have been observed in real computer systems, and we have adapted their effects to a logic representative level.

We have used the VHDL-based fault injection technique to carry out the injection experiments. This technique allows a great flexibility, controllability and observability. The different injection experiments have been performed using VFIT, a VHDL-based fault injection tool developed by our research group.

In the different injection experiments, we have studied the influence of the variation of different parameters of intermittent faults. We have compared also the results obtained when injecting transient and permanent faults.

In general terms, and assuming the important influence of the workload used, it has been observed that:

- The activity time, that is, the width of the pulses is the most significant parameter of intermittent faults. We have not observed important changes respect to the separation of the burst pulses.
- The spatial multiplicity of intermittent faults presents also a notable impact in the behaviour of the system.
- Intermittent faults have provoked a lower percentage of failures than permanent and transient faults. This is caused by the specific characteristics of the used workload and the transient fault model. When using more general workloads, we expect that intermittent faults will provoke more failures than transient ones.
- Intermittent faults have provoked a bigger percentage of latent errors than transient and permanent faults. This type of faults propagates into memory cells or registers, but they do not provoke failures, although the system remains contaminated.

In the future, we want:

- To use other workloads and other microprocessors in order to try to generalize the results;
- To inject faults in other critical places, like control and data buses;
- To analyze the effect of other parameters, like the burst length and the system clock frequency;
- To inject other observed fault models in buses, like delay, short and open;
- To investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, and propose and implement new related fault models.

Acknowledgement

This work has been partially funded by the Spanish Government under the Project MCYT TEC2005-05119 “Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida”.

References

[1] C. Constantinescu, “Trends and Challenges in VLSI Circuit Reliability”, IEEE Micro, Vol. 23(4):14-19, 2003.

[2] P.J. Gil *et al.*, “Fault Representativeness”, *Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245*, 2002.

[3] A. Benso and P. Prinetto, eds., “Fault Injection Techniques and Tools for VLSI reliability evaluation”, Kluwer Academic Publishers, 2003.

[4] D. Alexandrescu, L. Anghel, and M. Nicholaidis, “Simulating SET in VDSM ICs for Ground Level Radiation”, in JETTA(20):413–421, 2004.

[5] C. Constantinescu, “Neutron SER Characterization of Microprocessors”, Procs. 2005 International Conference on Dependable Systems and Networks (DSN’05), pp. 754-759, Yokohama, Japan 2005.

[6] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[7] D. Barros Jr, F. Vargas, M.B. Santos, I.C. Teixeira, and J.P. Teixeira, “Modeling and Simulation of Time Domain Faults in Digital Systems”, Procs. 10th IEEE International On-Line Testing Symposium (IOLTS’04), pp. 5-10, Portugal, July 2004.

[8] N. Kranitis, A. Merentitis, N. Laoutaris, and G. Theodorou, “Optimal Periodic Testing of Intermittent Faults In Embedded Pipelined Processor Applications”, Design, Automation and Test in Europe (DATE’06), Vol. 1, pp. 1-6, March 2006.

[9] C. Constantinescu, “Intermittent Faults in VLSI Circuits”, 2nd Workshop on Silicon Errors in Logic - System Effects (SELSE2), April 2006.

[10] C. Constantinescu, “Impact of Intermittent Faults on Nanocomputing Devices”, in WDSN-07, Edinburgh, UK, June 2007. http://www.laas.fr/WDSN07.

[11] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell, “Fault Injection for Dependability Validation: A Methodology and Some Applications”, IEEE Transactions on Software Engineering, vol. 16(2):166–182, 1990.

[12] D. Gil, J. Gracia, J.C. Baraza, and P.J. Gil, “Study, Comparison and Application of different VHDL-Based Fault Injection Techniques for the Experimental Validation of a Fault-Tolerant System”, Microelectronics Journal, vol. 34(1):41-51, 2003.

[13] "IEEE Standard VHDL Language Reference Manual", IEEE Std 1076-1993.

[14] E.A. Amerasekera and F.N. Najm, "Failure mechanisms in semiconductor devices", John Whiley & Sons, 1997.

[15] J. Gracia, D. Gil, L. G. Lemus and P. J. Gil, “Studying Hardware Fault Representativeness with VHDL Models”, in Proc. XVII Conference on Design of Circuits and Integrated Systems (DCIS’02), pp. 33-39, Santander, Spain, November 2002.

[16] J.C. Smolens, B.T. Gold, J.C. Hoe, B. Falsafi, and K. Mai. “Detecting Emerging Wearout Faults”, 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[17] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, and V. De, “Parameter Variations and Impact on Circuits and Microarchitecture”, in Procs. 40th Conference on Design Automation (DAC 2003), pp. 338 – 342, April 2003.

[18] C. Constantinescu, “Dependability Benchmarking using Environmental Test Tools”. Procs. Reliability and Maintainability Symposium (RAMS 2005), pp. 567–571, January 2005.

[19] J.W. McPherson, “Reliability challenges for 45nm and beyond”, Procs. Conference on Design Automation (DAC 2006), pp. 176–181, July 2006.

[20] J.H. Stathis, “Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits”, IEEE Transactions On Device And Materials Reliability, Vol. 1(1):43–59, March 2001.

[21] J.C. Baraza, J. Gracia, D. Gil, and P.J. Gil, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, Journal of Systems Architecture, vol. 47(10):847–867, 2002.

[22] http://www.oregano.at.

Applying Fault Injection to Study the Effects of Intermittent Faults

L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA)
Universidad Politécnica de Valencia, Spain
e-mail: {ljsaiz, jgracia, jcbaraza, dgil, pgil}@disca.upv.es

Abstract

As new submicron technologies have allowed increasing processors performance and decreasing their size, their reliability has decreased. In this way, intermittent faults are expected to be an important challenge in modern VLSI circuits. Currently, intermittent faults due to manufacturing residuals and process variations have grown, being necessary to study their effects. Using as background faults observed in real computer systems, we have injected intermittent faults in the VHDL model of a microcontroller. The controllability and flexibility of VHDL-based fault injection technique permits to carry out a detailed analysis of the influence of some parameters of intermittent faults.

1. Introduction

The advances in integration techniques have allowed rising microprocessors operating frequency as well as reducing their size and power voltage, achieving in this way a greater performance. Nevertheless, these advances have had a negative impact on reliability. As the new submicron technologies shrink device sizes, the rate of occurrence of faults increases. Whereas the effects of permanent and transient faults have been studied in depth [1], the influence of intermittent faults is not considered very often. It is foreseen that intermittent faults (together with transient faults) will have a great impact in deep submicron technologies [2]. The complexity of the manufacturing process (that provokes residuals and process variations) and some wear out mechanisms can be the origin of such faults. Although errors induced by transient and intermittent faults manifest in a similar way, the last ones are activated repeatedly in the same place, and so they are usually grouped in bursts [3].

Our objective is to study the impact of intermittent faults, and compare their consequences to those provoked by permanent and transient faults. We have used VHDL-based fault injection due to its flexibility and the high observability and controllability of all the modelled components [4]. In addition, fault injection allows performing a systematic and exhaustive analysis of the influence of the parameters of intermittent faults on the system's behaviour.

This work is organised as follows. Section 2 describes the intermittent fault models injected. Section 3 explains the fault injection experiments, and Section 4 discusses the results obtained. Finally, Section 5 provides some conclusions and a proposal of future work.

2. Fault models

An important issue is the definition of the fault models to be injected, as they must be representative of real faults. This requires studying the fault mechanisms considering the features of the technology used. Nevertheless, and as far as we know, the causes and mechanisms of intermittent faults in new submicron technologies have not been studied so much. Intermittent faults can be related to wear out mechanisms that eventually end up in permanent faults [5]. But it is also becoming more and more necessary to keep in mind the influence of manufacturing defects, like process variations and residuals.

In this paper we have selected a set of intermittent faults frequently observed in real computer systems by means of fault logging [3]. Then, we have extrapolated their effects to fault models applicable to the logic/RT abstraction levels. In this way, we can inject and simulate these faults into VHDL models. Table 1 summarises these intermittent faults and their mechanisms and models.

Table 1. Intermittent faults mechanisms and models

Causes	Targets	Fault mechanisms	Type of fault	Fault Models
Residuals in cells	Memory and registers	Intermittent contacts	Manufacturing defect	Intermittent stuck-at
Solder joints	Buses	Intermittent contacts	Manufacturing defect	Intermittent pulses, short, open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance	Wear out-timing	Delay, intermittent short, open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing	Delay
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wear out-timing	Indetermination, delay

3. Fault injection experiments

As stated above, intermittent faults manifest in bursts. So, to inject this type of faults, the following parameters must be configured (see Figure 1): the number of activations of the fault in the burst (we will call it *burst length*, or L_{Burst}), the duration of each activation (we will refer to it as *activity time*, or t_A), and the separation between two consecutive activations (we will name it as *inactivity time*, or t_I).

The different fault injection experiments have been carried out in the VHDL model of the 8051 microcontroller running the Bubblesort sorting algorithm as workload. Fault injection experiments have been carried out using a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [6], that runs on PC computers (or compatible) under Windows®.

In order to study the effects of intermittent faults, we have calculated in all cases the percentages of *failures* provoked and *latent errors* (that is, errors that propagate to registers and memory but do not cause failures).

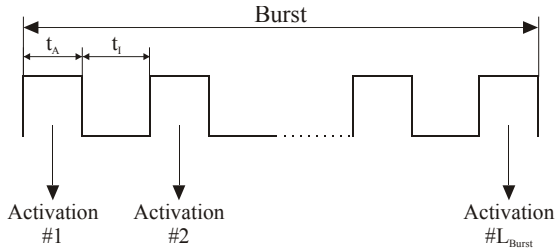


Figure 1. Description of the main elements of a burst

4. Results

Table 2 to Table 5 show the effect of intermittent faults injected in two types of targets: memory and registers (Table 2 and Table 3), and buses (Table 4 and Table 5). The fault models injected have been (see two first rows in Table 1): intermittent stuck-at (in memory and registers) and intermittent pulses (in buses). Time activity (t_A) has been generated randomly in three time intervals depending on the system clock cycle (T). Finally, single and multiple (in space domain, i.e. simultaneous faults in different targets) intermittent faults have been injected

Table 2. Results for single intermittent faults injected in memory and registers

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	0.1 %	0.4 %	3.9 %
Latent errors	41.9 %	8.9 %	0 %
Non effective	58.0 %	90.7 %	96.1 %

Table 3. Results for multiple intermittent faults injected in memory and registers

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	1.7 %	15.5 %	53.0 %
Latent errors	43.0 %	6.4 %	0 %
Non effective	55.3 %	78.1 %	47.0 %

Table 4. Results for single intermittent faults injected in buses

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	3.4%	28.2%	55.3%
Latent errors	2.5%	14.1%	16.3%
Non effective	94.1%	57.7%	28.4%

Table 5. Results for multiple intermittent faults injected in buses

Activity time (t_A)	[0.01T, 0.1T]	[0.1T, 1.0T]	[1.T, 10.0T]
Failures	13.4%	60.8%	89.4%
Latent errors	6.2%	19.8%	7.6%
Non effective	80.4%	19.4%	3%

The main observations that can be made are:

- As expected, the percentage of failures grows with the activity time, in both registers/memory and buses. We can also note that intermittent faults in buses have more impact that those in memory/registers.
- The percentage of latent errors does not show a clear trend with regard to the activity time. Whereas it decreases in registers/memory, it

rises slightly in buses. We think that this discrepancy is due to the workload.

- Multiple faults provoke a greater percentage of failures, which also was foreseeable. The difference is more pronounced in registers/memory than in buses.
- No significant differences in the percentage of latent errors are detected between single and multiple faults.

Finally, Figure 2 compares the impact of transient, intermittent and permanent faults injected in buses. In the case of transient faults, three different values of fault duration have been considered, using the same time intervals as for the activity time in intermittent faults. Some examples of transient and permanent fault models injected can be found in [6].

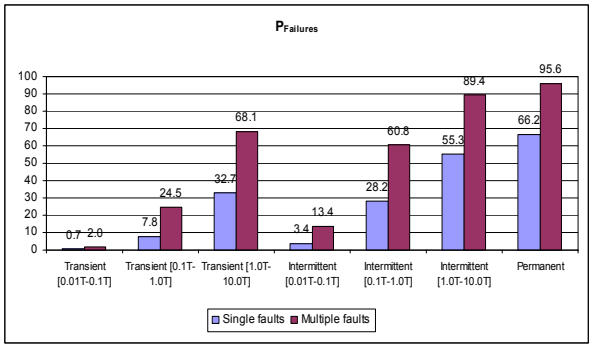


Figure 2. Comparison of the percentage of failures provoked by transient, intermittent and permanent faults in buses

From the figure, we can see that intermittent faults provoke a greater percentage of failures than transient faults. This is a logical result, as a burst of intermittent faults manifest like a sequence of transient faults in spite of having different origin. The effect of fault duration is also shown for all types of faults. Obviously, the greater impact is due to permanent faults because of their indefinite duration.

5. Conclusions and Future work

In this work, we have presented a case study of the effects of intermittent faults on the behaviour of a commercial microcontroller. The methodology used lies in VHDL-based fault injection technique, which allows a systematic and detailed analysis of the influence of different fault parameters. We have also compared the results to those obtained when injecting transient and permanent faults.

In the future, we have planned to extend this study in different aspects. First, we want to use other workloads in order to try to generalize the results. We also intend to analyse the effect of other parameters, like the burst length, the inactivity time, and the system clock frequency. Moreover, we think that it is important to inject other fault models, like delay, intermittent short and open. Finally, it is necessary to investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, in order to propose new fault models related to them.

6. Acknowledgement

This work has been partially funded by the Spanish Government under the project MCYT TEC2005-05119 “Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida”.

7. References

[1] P.J. Gil et al., “Fault Representativeness”, *Deliverable ETIE2 of Dependability Benchmarking Project*, IST-2000-25245, 2002.

[2] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[3] C. Constantinescu, “Impact of Intermittent Faults on Nanocomputing Devices”, in WDSN-07, Edinburgh, UK, June 2007, http://www.laas.fr/WDSN07.

[4] A. Benso and P. Prinetto, eds., “*Fault Injection Techniques and Tools for VLSI reliability evaluation*”, Kluwer Academic Publishers, 2003.

[5] J.C. Smolens et al, “Detecting Emerging Wearout Faults”, 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[6] J.C. Baraza et al, “A Prototype of a VHDL-Based Fault Injection Tool: Description and Application”, *Journal of Systems Architecture*, vol. 47(10):847–867, 2002.

Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers[†]

D.Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto ITACA
Universidad Politécnica de Valencia, Spain
e-mail: {dgil, ljsaiz, jgracia, jcbaraza, pgil}@disca.upv.es

Abstract

It is expected that intermittent faults will be a great challenge in modern VLSI circuits. In this work, we present a case study of the effects of intermittent faults on the behavior of a commercial microcontroller. The methodology used lies in VHDL-based fault injection technique, which allows a systematic and exhaustive analysis of the influence of different fault and system parameters. From the simulation traces, the occurrences of failures and latent errors have been logged. To extend the study, the results obtained have been compared to those got when injecting transient and permanent faults. The applied methodology can be generalized to more complex systems.

1. Introduction

It has been foreseen that intermittent faults will have a great impact in deep submicron technologies. The reduction of the feature size and the power voltage, together with the increase of the clock frequency, will raise the rate of transient faults. On the other hand, the complexity of the manufacturing process (that provokes residues and process variations) and special wearout mechanisms may increase the presence of intermittent faults [1].

Although errors induced by transient and intermittent faults manifest in a similar way, the last ones are activated repeatedly in the same place, and so they are usually grouped in bursts. On the other hand, whereas replacing the affected part eliminates an intermittent fault, transient faults cannot be fixed by repair. Additionally, intermittent faults may be activated or deactivated by temperature, voltage or frequency changes [2].

Transient and permanent faults have well established fault mechanisms and models [3]-[7]. Permanent faults occur due to irreversible physical changes. Transient faults are mainly generated by environmental conditions, like cosmic rays. On the other hand, the knowledge of intermittent faults is not so developed, as fewer observations of real faults and studies of their physical causes and mechanisms have been performed [2][8][9][10].

Related to intermittent faults, there are some questions difficult to answer: Where do intermittent faults happen? When do faults occur? How many times does a fault activate in a burst? How does the fault manifest at higher abstraction levels? etc. To answer these questions, it is important to understand the physical mechanisms that take place in deep submicron technologies. But this research subject is complex, it is technology dependent, and it is still in an early phase of evolution.

In order to study the impact of intermittent faults, we propose to use a methodology based on fault injection as a complement to the research on the *physics of fault.* Fault injection technique allows a controlled introduction of faults in the system, not being necessary to wait for a long time to log the apparition of real faults [11]. Particularly, VHDL-based fault injection can be a very suitable option due to its flexibility as well as the high observability and controllability of all the model components [12]. It allows both to make an exhaustive variation of the fault parameters and to analyze the effects of faults on the system behavior.

In previous works [13][14], we have generated fault models for intermittent faults at logic and RTL abstraction levels, and injected these faults in the VHDL model of a commercial microcontroller to study their impact on the system behavior. In these works, the influence of different fault and system parameters has been analyzed.

The objective of this work is to deepen those studies, analyzing the importance of other factors. Faults have been injected in new locations, and both

[†] This work has been partially funded by the Spanish Government under the project MCYT TEC2005-05119 "Estudio de las técnicas de inyección de fallos en modelos en VHDL: emulación de fallos en FPGA y uso de nuevas técnicas de simulación distribuida".

the burst length and the operation frequency have been modified.

This work is organized as follows. Section 2 depicts a set of intermittent fault models that can be injected. Section 3 describes the fault injection experiments, and Section 4 discusses the results obtained. Finally, Sections 5 and 6 provide some conclusions and a proposal of future work.

2. Fault Models

As stated above, whereas the models of transient and permanent faults have been traditionally well established, intermittent fault modeling is a pending issue. To obtain representative fault models for intermittent faults, it is necessary to understand the physical mechanisms that take place in deep submicron technologies. Intermittent faults occur due to unstable or marginal hardware. Manufacturing residues, process variations and special wearout processes can lead to such faults. In addition, wearout mechanisms may provoke that intermittent faults eventually end up in permanent faults.

We have selected a set of intermittent faults observed in real computer systems by means of fault logging [9][2], as well as fault mechanisms related to process variations and wearout [2][10][15][16]. Then, we have deduced a set of fault models at logic (gate) and RTL abstraction levels which can be simulated into VHDL models [13][14]. Table 1 shows some examples of intermittent faults and the associated fault models.

Fault models applied in this work have been emphasized in bold. Bursts of Single Bit Errors (or SBEs) have been observed in the memory of a set of monitored servers [9]. Analysing the failures, it was found that polymer residues led to an intermittent contact. The fault model chosen to represent a SBE in memory and register cells is *intermittent stuck-at*, which represents that the value of the cell changes intermittently between '0' and '1'. The origin of the fault lies in an unstable hardware, and it is not related to an external environmental effect (e.g. cosmic radiation). So, it is not an *intermittent bit-flip*, and we have called this fault model *intermittent stuck-at*.

Bursts of SBEs have also been observed in data buses [9]. The analysis of the failures revealed that the source of errors were solder intermittent contacts. We have associated this mechanism to the following fault models: *intermittent pulse*, *intermittent short* and *intermittent open*, appliable to bus lines [4].

3. Fault injection experiments

The different fault injection experiments were carried out on the VHDL model of the 8051 microcontroller [17] running the Bubblesort sorting algorithm as workload. Although the chosen system has not a deep submicron technology, the methodology used can be generalized to more complex microprocessors/microcontrollers.

Fault injection experiments were performed using a fault injection tool called VFIT (VHDL-based Fault Injection Tool) [18], that runs on PC computers (or compatible) under Windows®. VFIT can apply different fault injection techniques on VHDL models (see Figure 1) at diverse abstraction levels. In this work we used the technique based on *simulator commands*, because it is not necessary to modify the VHDL code and it is easy to apply. This technique consists on using the commands of the simulator to modify the value of the model signals and variables at simulation time. Nevertheless, it has limitations to inject some complex fault models [12].

Next we will set the main injection parameters. Some of them are closely connected with the questions posed in Section I.

Table 1. Some intermittent faults mechanisms and models [14]

Causes	Places	Fault mechanisms	Type of fault	Fault Models
Residues in cells	**Memory and registers**	Intermittent contacts	Manufacturing defect	**Intermittent stuck-at**
Solder joints	**Buses**	Intermittent contacts	Manufacturing defect	**Intermittent pulse**, intermittent short, intermittent open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance	Wearout-timing	Delay, intermittent short, intermittent open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing Voltage perturbation	Delay,intermittent pulse
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wearout-timing	Delay, intermittent indetermination

In bold, fault models injected in this work

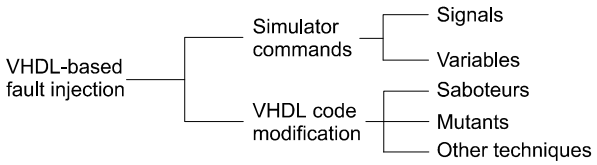


Figure 1. Fault-injection techniques for VHDL models [12]

Where are faults injected? Intermittent faults were injected in two types of places (targets) of the microcontroller: memory elements (the register file and the internal RAM), and the buses. We chose these targets because in real systems, intermittent faults have been logged in similar locations, as it was indicated in Section II. Figure 2 shows the structure and the injection targets of the microcontroller. There are other locations potentially sensitive to intermittent faults, such as the combinational logic (arithmetic and control circuits), that will be analyzed in a future work.

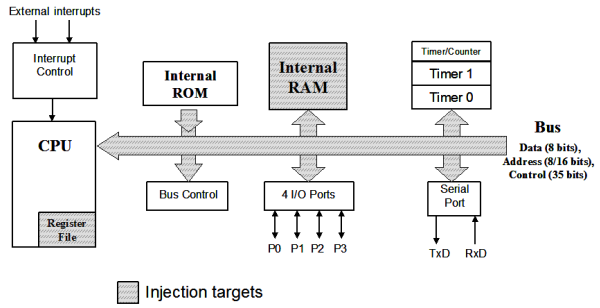


Figure 2. Injection targets

Depending on the number of places perturbed in each injection, faults can be classified as *single* (one cell or bus line) or *multiple* (various cells or bus lines). As stated in Section I, it is expected that there will be a rise of the manufacturing defects in deep submicron technologies. This may imply the presence of multiple faults in the system, and so it is interesting to take into account this issue in the injection experiments.

What type of faults are injected? The fault models to inject depend strongly on the injection place. According to Table 1, and considering the capabilities of the injection technique chosen, the intermittent fault models selected are: 1) *Intermittent stuck-at*, for memory and registers, and 2) *Intermittent pulse*, for buses. These fault models are emphasized in bold in Table 1.

When are the faults injected? The fault injection instant has been selected randomly along the workload duration, according to a Uniform distribution. In real systems, other fault distributions have been observed, such as Exponential or Weibull [8]. For instance, a Weibull distribution with increasing fault rate can be used to emulate a wearout process.

Which are the parameters of the burst? As stated above, intermittent faults manifest in bursts. So, to inject this type of faults the following parameters must be configured (see Figure 3):

- The number of activations of the fault in the burst (we will call it *burst length*, or L_{Burst}).
- The duration of each activation (we will refer to it as *activity time*, or t_A).
- The separation between two consecutive activations (we will name it as *inactivity time*, or t_I).

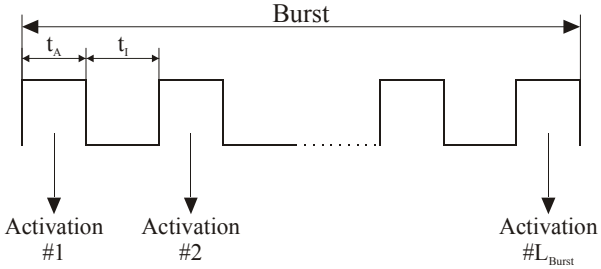


Figure 3. Main elements of a burst

The burst length (L_{Burst}) was varied randomly between 1 and 10. Activity time (t_A) and inactivity time (t_I) were generated randomly according to a Uniform distribution function in three time intervals, depending on the system clock cycle (T): [0.01T – 0.1T], [0.1T – 1.0T], and [1.0T – 10.0T].

Nevertheless, in wearout processes it is expected that during lifetime, L_{Burst} and t_A will increase and t_I will decrease. At last, the intermittent fault may become a permanent fault. Hence, we will use in the future other distribution functions, such as Weibull.

In order to measure the impact of intermittent faults, we have calculated in all cases the percentages of provoked *failures* and *latent errors*:

- Percentage of failures:

$$P_{Failures} = \frac{N_{Failures}}{N_{Injected}} \times 100$$

- Percentage of latent errors:

where:

- $N_{Injected}$ = number of faults injected (1000 per injection experiment).
- $N_{Failures}$ = Number of failures, defined as the number of propagated errors (that is, the injected faults which propagate to the memory and registers) that provoke failures. A failure is produced when the result is erroneous at the end of the simulation time. Similar definitions of failure can be found in [8] [19].
- N_{Latent} = Number of latent errors. A latent error is a propagated error that has not provoked a failure.

Figure 4 summarizes the syndrome of faults and the calculated data.

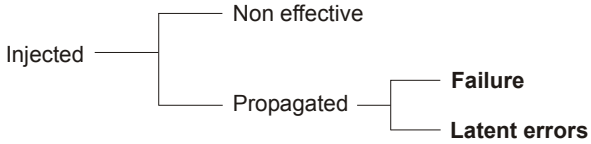


Figure 4. Syndrome of faults

4. Results

According to the injection parameters whose incidence is analyzed, we have divided the injection experiments in two groups. First, we show the study of the influence of the burst parameters. Next, we present the study of the influence of other parameters, such as the injection place and the system clock frequency.

In both studies, the importance of the fault multiplicity has been considered.

4.1. Influence of burst parameters

As explained in Section 3, the main parameters of an intermittent fault burst are the number of activations (L_{Burst}), the duration of the activations (t_A), and the separation between activations (t_I).

4.1.1. Activity time. Figures 5 and 7 show that the percentage of failures grows with the activity time (t_A in Figure 3), both in memory/registers and buses. Depending on the target and the fault multiplicity, the growth seems to fit logarithmic or linear functions (note that the t_A scale is logarithmic).

We can also note that intermittent faults have more impact on buses than on memory/registers, showing that buses are more sensitive to intermittent faults.

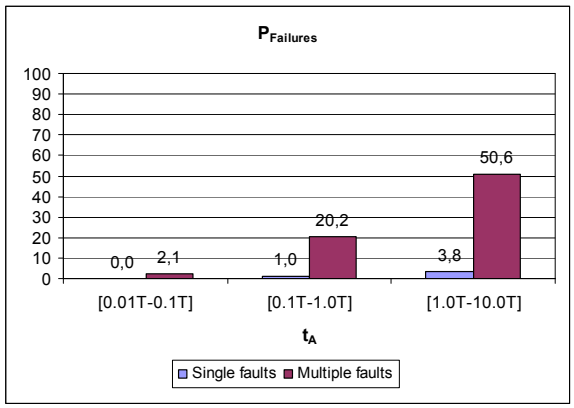


Figure 5. Influence of t_A in $P_{Failures}$ (Target: memory and registers)

Figures 6 and 8 reflect that the percentage of latent errors does not show a clear trend with regard to the activity time. Whereas it decreases in memory/registers, it rises slightly in buses. This discrepancy can be due to the workload.

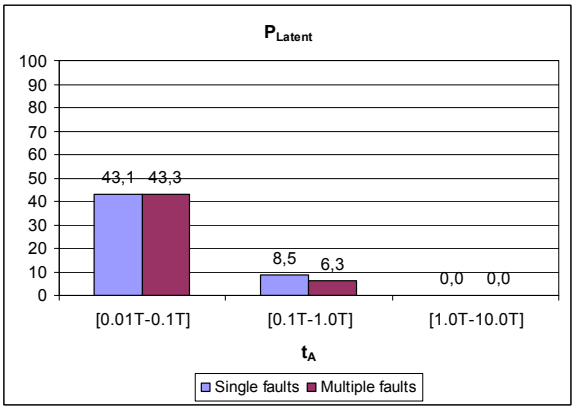


Figure 6. Influence of t_A in P_{Latent} (Target: memory and registers)

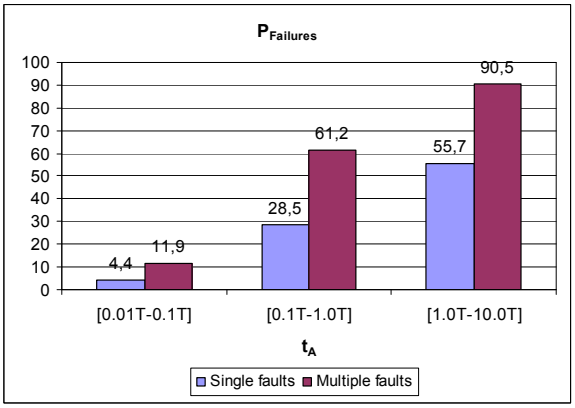


Figure 7. Influence of t_A in $P_{Failures}$ (Target: buses)

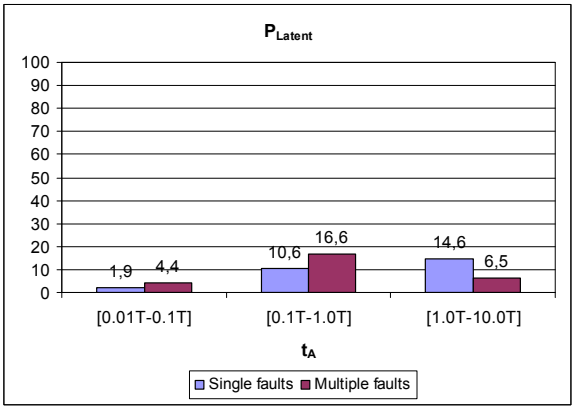


Figure 8. Influence of t_A in P_{Latent} (Target: buses)

Regarding the spatial multiplicity, from Figures 5 to 8, it can be seen that multiple faults provoke a greater percentage of failures than single faults. This is a predictable behaviour, because multiple faults affect simultaneously various physical locations of the system. Comparing the $P_{Failures(multiple)}/P_{Failures(single)}$ ratio,

values of up to 20 have been observed. It is noticeable that higher values of this ratio have been detected for lower values of t_A, and specially in registers/memory targets.

Finally, no significant differences in the percentage of latent errors are detected between single and multiple faults.

4.1.2. Separation between activations. Figures 9 and 10 show the influence of the temporal separation between the pulses of the burst (inactivity time or t_I in Figure 3) for intermittent faults injected in registers and memory. No significant differences are observed when varying the inactivity time. The separation of the activations of the burst does not provoke a noteworthy variation in $P_{Failures}$ or P_{Latent} (for a fixed value of L_{Burst}). A similar behavior has been observed when injecting in buses.

4.1.3. Burst length. The number of activations of the burst (L_{Burst} in Figure 3) has been varied from 1 to 10. In wearout processes, the trend is expected to be a raise of the length of the burst during the life time, until the fault becomes a permanent fault.

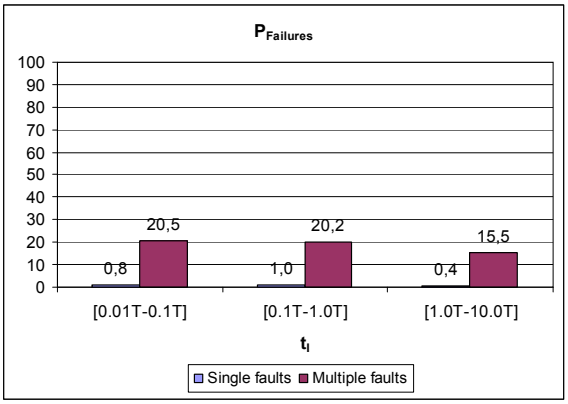


Figure 9. Influence of t_I in $P_{Failures}$ (Target: memory and registers)

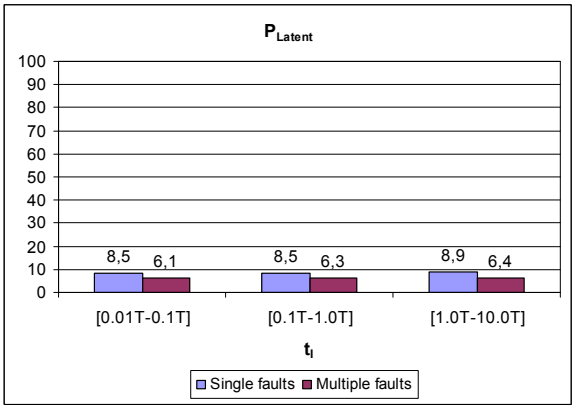


Figure 10. Influence of t_I in P_{Latent} (Target: memory and registers)

Figure 11, corresponding to faults injected in memory and registers, shows that $P_{Failures}$ rises asymptotically up to 27.5%. From $L_{Burst} \approx 9$ $P_{Failures}$ gets stable. On the other hand, P_{Latent} has small variations with the length of the burst, with values between 5% and 8%.

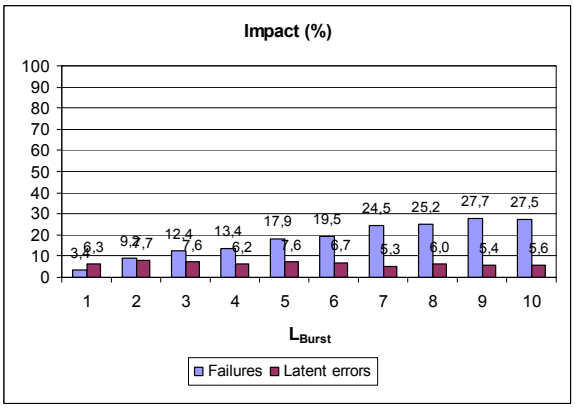


Figure 11. Influence of the L_{Burst} in $P_{Failures}$ and P_{Latent} (Target: memory and registers)

Figure 12, corresponding to faults injected in buses, shows the same trend that Figure 11, although the values of $P_{Failures}$ and P_{Latent} are notably higher. $P_{Failures}$ rises asymptotically up to 75%, and stabilizes from $L_{Burst} \approx 9$. Instead, P_{Latent} shows fewer variations, varying between 13% and 21%.

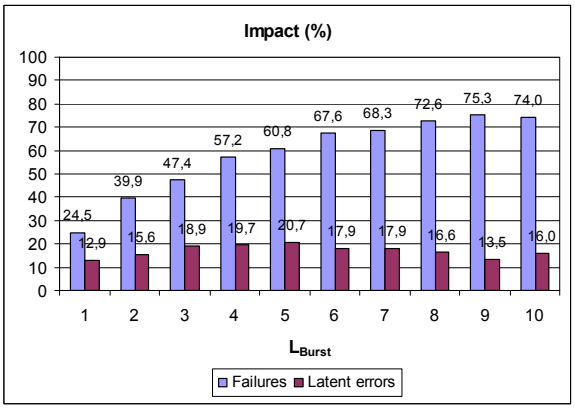


Figure 12. Influence of L_{Burst} in $P_{Failures}$ and P_{Latent} (Target: buses)

4.2. Influence of other parameters

4.2.1. Injection place. In previous results we have observed that intermittent faults in buses are more harmful than in memory/registers. One reason can be the difference in the area occupied by the two targets. The size (in bits) of the registers plus memory is quite bigger than that of the buses. This implies that the probability of sensitizing the workload is lower. In

addition, buses are a bottle-neck in the fetch and execution phases.

This suggests the need of adding mitigation techniques to detect and tolerate intermittent faults in buses. For instance, ECC implemented in hardware can deliver fast error detection and correction of errors occurring at a high rate. In general, hardware implemented error handling techniques are likely to provide the best solutions to diminish the effects of intermittent faults [2].

4.2.2. Clock frequency of the system. Figures 13 and 14 show $P_{Failures}$ and P_{Latent} respect to the frequency, for single and multiple intermittent faults in buses. In both cases, $P_{Failures}$ rises asymptotically, similarly to as with the burst length.

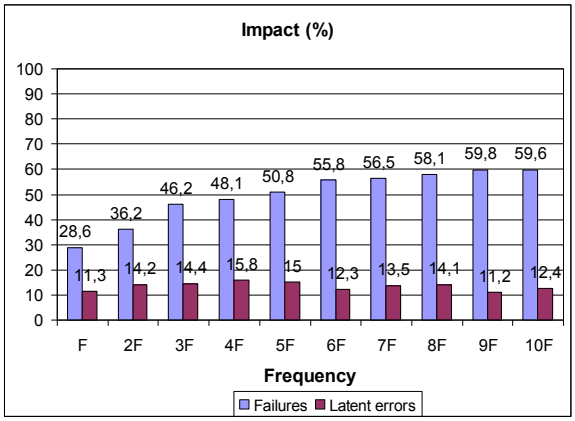


Figure 13. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: buses. Single faults)

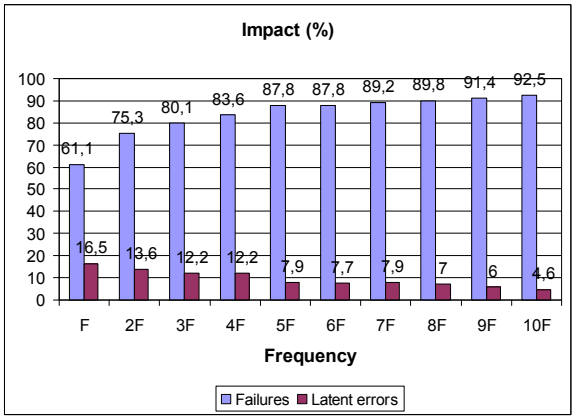


Figure 14. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: buses. Multiple faults)

When injecting single faults, $P_{Failures}$ tends to 60%, while with multiple faults it stabilizes at about 90% from 7×F (F = 10 MHz). The explanation of the strong frequency influence lies in the fact that at higher frequencies, the probability of capturing an error in active edges of synchronous components (like memory and registers) rises. It is important to remark that the frequency increase has been a trend in VLSI technologies.

On the other hand, P_{Latent} does not reflect a clear dependency. In Figure 14 it seems to show a smooth negative exponential function, whereas in Figure 13 we do not observe a defined trend. Until 4×F, P_{Latent} raises to 16%. Since then, it holds around 11%-15%.

When injecting in memory and registers, we have observed a similar asymptotic trend of $P_{Failures}$, as Figures 15 and 16 show. Nevertheless, the values are notably lower than in buses case. About P_{Latent} values, a sharp negative exponential behaviour is observed.

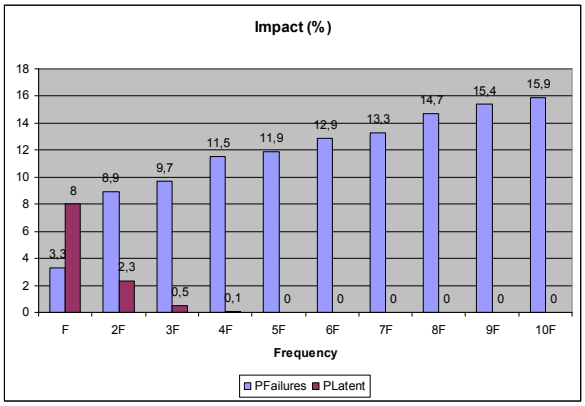


Figure 15. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: memory and registers. Single faults)

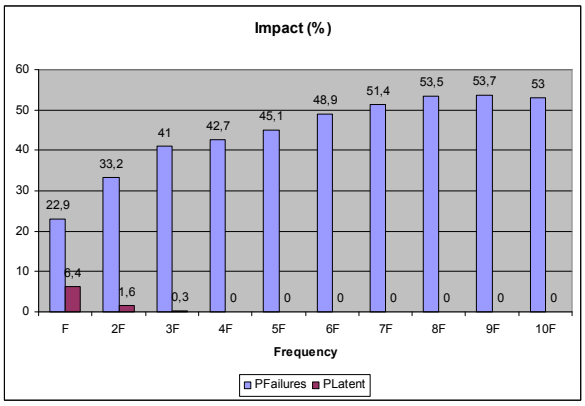


Figure 16. Influence of the clock frequency in $P_{Failures}$ and P_{Latent} (Target: memory and registers. Multiple faults)

4.3. Comparison with transient and permanent faults

Finally, Figures 17 and 18 compare the impact of transient, intermittent and permanent faults for the two types of targets. Transient fault models injected have been *bit-flips* in memory/registers (to emulate Single Event Upsets, or SEUs) and *pulses* in buses (to emulate

Single Event Transients, or SETs). For permanent faults, the fault models injected have been *stuck-at(0,1)*, *open*, and *indetermination* [12][18]. In the case of transient faults in buses, three different values of fault duration have been considered, using the same time intervals as for the activity/inactivity times in intermittent faults. Due to their nature, when injecting bit-flips, it does not make any sense to set any duration.

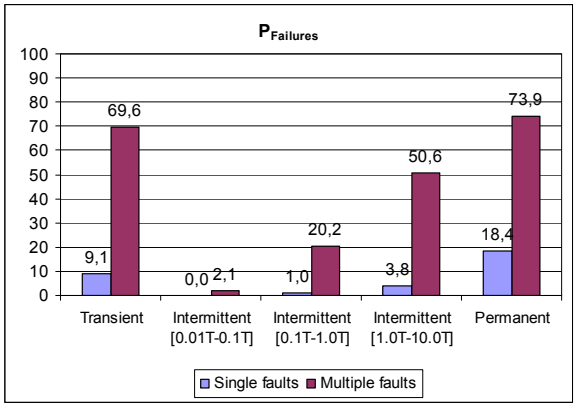


Figure 17. Influence of the fault type in $P_{Failures}$ (Target: memory and registers)

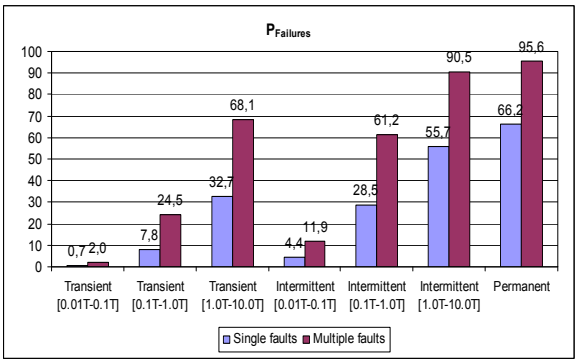


Figure 18. Influence of the fault type in $P_{Failures}$ (Target: buses)

In Figure 18 (buses) we can see that intermittent faults provoke a greater percentage of failures than transient faults. This is a logical result, as a burst of intermittent faults manifest like a sequence of transient faults in spite of having different origin. Calculating the $P_{Failures(intermittent)}/P_{Failures(transient)}$ ratio, values between 1.5 and 6 are observed. Values are higher for single faults.

However, Figure 17 (memory/registers) shows an unexpected trend. Intermittent faults have a lower impact than transient faults, even for the biggest activity time. A more detailed analysis of the injected fault models and the workload allowed establishing the cause of this anomalous behavior. The characteristics of the workload used provoke a low rate of overwrite operations in the memory cells affected by transient faults (bit-flips), causing a *de facto* infinite duration of these faults, thus remaining stored permanently.

Obviously, in both figures, the greatest impact corresponds to permanent faults because of their infinite duration, independently of the type of target.

5. Conclusions

In this work, we have presented a case study of the effects of intermittent faults on the behavior of a commercial microcontroller (8051). The methodology used lies in VHDL-based fault injection technique, which allows a systematic and exhaustive analysis of the influence of different fault parameters and factors. We have also compared the results to those obtained when injecting transient and permanent faults. This methodology can be applied to other microcontrollers and/or workloads.

In general terms, and assuming the important influence of the microcontroller and the workload used, some results can be summarized.

About the influence of burst parameters:

- The activity time is a quite critical parameter. Increasing the duration of the activations provokes the rise of the percentage of failures. Linear or logarithmic growths have been observed, depending on the target and/or the fault multiplicity.
- The burst length has also a notable impact. Nevertheless, when increasing the burst length, the percentage of failures grows towards a horizontal asymptote. From a given value of burst length, the percentage of failures stabilizes and remains constant.
- Respect to the inactivity time, no important changes have been observed. It seems that the separation between activations does not modify the percentage of failures (assuming a fixed number of activations).

During lifetime, it is expected that the trend in wearout fault mechanisms will be a raise of both the activity time and the burst length, and a decrease of the separation between activations. This must be taken into account in future fault injection experiments in order to characterize the burst in a more realistic way.

About other factors studied:

- The *spatial multiplicity* of intermittent faults presents a significant impact on the behavior of the system. Multiple faults provoke a greater percentage of failures than single faults. The complexity of the manufacturing process in deep submicron technologies may favor the presence of multiple intermittent faults.

- The injection *target (place)* leads also to important differences. We have observed that intermittent faults in buses are more harmful than in memory/registers. This fact suggests the suitability of adding mitigation techniques to deliver fast error detection and correction of intermittent errors in buses. For instance, applying ECC implemented in hardware.
- The rise of the *clock frequency* increases the impact of intermittent faults. This is due to the higher probability of capturing an error in active edges of synchronous components. The percentage of failures presents an asymptotical rising trend, similar to the burst length dependency. The increase of the clock frequency has been a trend in deep submicron technologies.

It is necessary to point out that the percentage of latent errors (errors that propagate to the memory and registers, but do not provoke failures) has not shown a definite trend with regard to most parameters. Factors related with the workload execution may be involved in their behavior and need to be more deeply studied.

Finally, the impact of the intermittent faults has been compared to that of transient and permanent faults. As a burst of intermittent faults manifest like a sequence of transient faults (in spite of having a different origin), intermittent faults provoke a greater percentage of failures than transient faults. Obviously, the greatest impact is caused by permanent faults because of their infinite duration.

6. Future work

In the future, we have planned to extend this study in different aspects. First, we intend to use other workloads in order to try to generalize the results. We also want to analyze the effect of intermittent faults on other microprocessors or microcontrollers. Moreover, we think that it is important to inject other representative fault models, like delay, intermittent short and intermittent open. It is also necessary to investigate more deeply the causes and mechanisms of intermittent faults related to new submicron technologies, in order to propose new fault models related to them. Particularly, it would be interesting to identify intermittent fault models for combinational circuits. Finally, we want to study mitigation techniques for fast detection and recovery of intermittent faults.

7. References

[1]. C. Constantinescu, "Impact of Deep Submicron Technology on Dependability of VLSI Circuits", in Procs. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[2]. C. Constantinescu, "Impact of Intermittent Faults on Nanocomputing Devices", in Procs. WDSN-07, Edinburgh, UK, June 2007, http://www.laas.fr/WDSN07.

[3]. E.A. Amerasekera and F.N. Najm, "Failure mechanisms in semiconductor devices", John Whiley & Sons, 1997.

[4]. P.J. Gil *et al.*, "Fault Representativeness", *Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245*, 2002.

[5]. J. Segura, A. Keshavarzi, C.F. Hawkins, "CMOS IC nanometer technology failure mechanisms", IEEE Custom Integrated Circuits Conference, 2003.

[6]. J. Segura, C.F. Hawkins, "CMOS Electronics. How it works, how it fails". 2004.

[7]. N. Weste, D. Harris, "CMOS VLSI design", Addison-Wesley, 2005.

[8]. D.P. Siewiorek, R. S. Schwarz: *Reliable computer systems: Design and evaluation* (2nd. Edition). Ed. Digital Press, 1992.

[9]. C. Constantinescu, "Dependability Benchmarking using Environmental Test Tools". Procs. Reliability and Maintainability Symposium (RAMS'05), pp. 567–571, January 2005.

[10]. J.C. Smolens, B.T. Gold, J.C. Hoe, B. Falsafi, and K. Mai. "Detecting Emerging Wearout Faults", 3rd Workshop on Silicon Errors in Logic - System Effects (SELSE3), April 2007.

[11]. J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", IEEE Transactions on Software Engineering, vol. 16(2):166–182, 1990.

[12]. D. Gil, J. C. Baraza, J. Gracia, and P. J. Gil, "VHDL simulation-based fault injection techniques", Ch. 4.1 in *Fault Injection Techniques and Tools for VLSI Reliability Evaluation*, A. Benso and P. Prinetto, Eds. Dordrecht, The Nederlands: Kluwer Academic Publishers, 2003, pp. 159–176.

[13.]. J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil, "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), pp. 80-85, Bratislava, SlovaKia, April 2008

[14]. L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Applying Fault Injection to Study the Effects of Intermittent Faults", 7th European Dependable Computing Conference (EDCC-7), Supplemental Volume, pp.67-69, Kaunas, Lithuania, May 2008.

[15]. S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitecture", in Procs. 40th Conference on Design Automation (DAC'03), pp. 338 – 342, April 2003.

[16]. J.W. McPherson, "Reliability challenges for 45nm and beyond", Procs. Conference on Design Automation (DAC'06), pp. 176–181, July 2006.

[17]. http://www.oregano.at

[18]. J.C. Baraza et al, "A Prototype of a VHDL-Based Fault Injection Tool: Description and Application", *Journal of Systems Architecture*, vol. 47(10):847–867, 2002.

[19]. J.C. Laprie, "Dependability: basic concepts and terminology", *Sringer-Verlag*, 1992.

Intermittent Faults: Analysis of Causes and Effects, New Fault Models, and Mitigation Techniques

Luis-J. Saiz-Adalid
Grupo de Sistemas Tolerantes a Fallos (GSTF) – Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas – Universidad Politécnica de Valencia
Camino de Vera, s/n, 46022, Valencia, Spain
ljsaiz@disca.upv.es

Abstract

It is expected that dealing with intermittent faults will be a great challenge in modern VLSI circuits. As new submicron technologies have increased processors performance and reduced their size, their reliability has decreased. The objective of my research work is to study intermittent faults, to analyze their causes and effects and to propose new fault models and mitigation techniques. In order to check these new proposals, it is planned to develop the VHDL model of a fault tolerant system including different mitigation techniques. These techniques will be validated by injecting the intermittent fault models proposed.

1. Introduction

The advances in integration techniques have allowed rising microprocessors operating frequency, as well as reducing their size and power voltage, achieving a greater performance. Nevertheless, these advances have had a negative impact on their reliability, and the rate of occurrence of faults has increased [1]. Permanent and transient faults have been extensively studied, but the analysis of the causes and mechanisms of intermittent faults is a pending issue and an open line of research.

It is foreseen that intermittent faults will have a great impact in deep submicron technologies [2]. Usually intermittent faults have been assumed to be the prelude of permanent faults. Currently, intermittent faults due to process variations and manufacturing defects have grown.

Errors induced by transient and intermittent faults manifest in a similar way, but the last ones are activated repeatedly in the same place, and they are usually grouped in bursts. On the other hand, replacing the affected part eliminates an intermittent fault, while transient faults cannot be fixed by repair. Additionally, intermittent faults may be deactivated or activated by temperature, voltage and frequency changes [3].

Related to intermittent faults, there are some questions difficult to answer: Where do intermittent faults happen? When do faults occur? How many times does a fault activate in a burst? How does the fault manifest at higher abstraction levels? … To answer these questions, it is important to understand the physical mechanisms that take place in deep submicron technologies. But this research subject is complex, it is technology dependent, and it is still in an early phase of evolution.

In order to study the impact of intermittent faults, a methodology based on fault simulation is proposed. Fault injection allows a controlled introduction of faults in the system, not being necessary to wait for a long time to log the apparition of real faults [4]. Particularly, VHDL-based fault injection can be a very suitable option due to its flexibility and high observability and controllability of all the model components [5]. It allows to carry out an exhaustive variation of the fault parameters as well as to analyze the effects of faults on the system behavior.

My research work is enclosed in the GSTF (Fault-Tolerant Systems) research group. The group has a wide experience in fault injection techniques and dependability research. This paper introduces the research line for my PhD thesis.

Following this research line, first results and conclusions obtained by the group studying the effects of intermittent faults can be found in [6].

This work is organized as follows. Section 2 describes the intermittent fault models studied. Section 3 introduces the fault injection environment. Section 4 presents a brief analysis of mitigation techniques. Finally, Section 5 provides proposals for future work.

2. Intermittent Fault Models

Hardware fault representativeness for transient and permanent faults has been extensively studied and they are well established [7]. As far as I know, the causes and mechanisms of intermittent faults in deep submicron technologies have not been studied so much. To obtain representative fault models for intermittent faults, it is necessary to understand their physical mechanisms.

The first option is to use fault models similar to those used previously for permanent faults, changing their temporal behavior. In fact, in a wearout process faults initially may appear intermittently, but eventually end up in permanent faults [3].

However, the introduction of new submicron technologies makes necessary to keep in mind the influence of manufacturing defects. Process variations, residues and special wearout processes can lead to new intermittent faults [8]. A deeper study is necessary, being this an open line of research.

To define a set of intermittent fault models, the effects of intermittent faults observed in real computer systems by means of fault logging [2], as well as fault mechanisms related to process variations and wearout [2][8] have been extrapolated. In previous works, some intermittent fault models at logic (gate) and RT abstraction levels have already been deduced [9].

Bursts of Single Bit Errors (or SBEs) have been observed in memory [2]. The analysis of failures found that polymer residues can lead to an intermittent contact. The origin of the fault lies in an unstable hardware, and it is not related to an external environmental effect. The value of a bit storage cell changes intermittently to a fixed value. It is not a repeated bit-flip. Instead, it is similar to a stuck-at, but with intermittent temporal behavior. The fault model deduced to represent bursts of SBEs in storage elements (registers and memory) has been called *intermittent stuck-at*.

Bursts of SBEs have also been observed in data buses [10]. Failure analysis revealed that the source of errors was intermittent solder contacts. It is expected that control lines would have the same problem, although this kind of errors are more difficult to log. We have associated this mechanism to *intermittent pulse*, *intermittent short* and *intermittent open* fault models in bus and control lines.

Also, timing failures may occur due to propagation delays over the interconnection lines. For instance, barrier layer delamination and electromigration lead to a higher resistance and, as a result, to timing violations. Electromigration can also induce voids [3] that can provoke short and open faults in metal lines. Process, voltage and temperature (PVT) variations tend to amplify crosstalk delays. They may take place when adjacent signals switch in opposite directions, and are related to capacitive effects in adjacent metallic lines (Miller effect). Even more, crosstalk speed-ups may occur when adjacent signals switch in the same direction [11], provoking that signals switch faster than expected. The fault models proposed to these mechanisms are *intermittent delay*, *intermittent speed-up*, *intermittent short* and *intermittent open* in interconnection lines and buses.

Another type of causes and mechanisms of intermittent faults are due to oxide defects and wearout. Soft breakdown belongs to this class. In this case, the leakage current of the gate oxide fluctuates in time, without inducing the thermal damage, so it does not produce a permanent breakdown [12]. This provokes erratic fluctuations of the minimum supply voltage. Sensitivity to this parameter is expected to increase with scaling. This fluctuation may lead to voltage values not associated to a logic value and can be modeled with the *intermittent indetermination* fault model.

Table 1 summarizes the intermittent fault mechanisms and models described.

In the future, it is planned to carry out a deeper study of the fault mechanisms related with wearout processes and process variations in new submicron technologies.

Table 1. Some intermittent faults mechanisms and models

Causes	Targets	Fault mechanisms	Type of fault	Fault Models
Residuals in cells	Memory and registers	Intermittent contacts	Manufacturing defect	Intermittent stuck-at
Solder joints	Buses	Intermittent contacts	Manufacturing defect	Intermittent pulse, intermittent short, intermittent open
Electromigration Barrier layer delamination	Buses I/O connections	Variation of metal resistance and/or capacity Voids	Wearout/Timing	Intermittent delay, intermittent speed up, intermittent short, intermittent open
Crosstalk	Buses I/O connections	Electromagnetic interference	Timing Voltage perturbation	Intermittent delay, intermittent pulse
Soft breakdown	Power supply voltage	Leakage current fluctuation	Wearout/Timing	Intermittent delay, intermittent indetermination

3. Fault injection environment

The GSTF have developed VFIT (VHDL-based Fault Injection Tool) [13], an automatic fault injection tool to inject faults in VHDL models. VFIT is model-independent and admits VHDL models at any abstraction level, although it has been mainly used on models at logic and RT levels. VFIT has been used for years in different research projects, and hundreds of fault injection experiments have been carried out [7][14][15].

VFIT can inject permanent, transient and intermittent faults, in both single and multiple locations. It allows to inject a wide set of fault models that try to be representative of deep submicron technologies.

The output of an injection experiment depends on the objective: it can perform an error syndrome analysis or a dependability validation campaign. In the first case, it calculates propagation latencies and percentages of propagated errors and failures. In the second one, detection and recovery coverages and latencies are obtained.

As mentioned before, intermittent faults manifest in bursts. That is, it activates and deactivates repeatedly several times in the same place. So, to inject intermittent faults, the following parameters must be configured (see Figure 1): the number of activations in the burst (burst length, or L_{Burst}), the duration of each activation (activity time, or t_A), and the separation between two consecutive activations (inactivity time, or t_I). In VFIT, the values for these parameters can be selected randomly, according to a selectable distribution function. In a burst, each activation or separation times may have different durations.

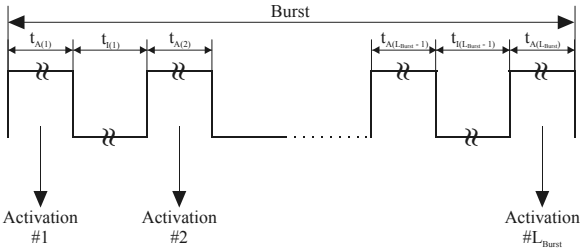


Figure 1. Parameters of a burst

Several intermittent fault injection experiments have been carried out into the VHDL model of the 8051 microcontroller [16] running different workloads: the Arithmetic Series, the Bubblesort algorithm and a matrix multiplication.

The objective of these experiments is to study the influence of different parameters of intermittent faults in the system. Due to lack of space, results are omitted here. Anyway, a detailed explanation of some of these experiments, results and conclusions can be found in [6][9][17].

4. Mitigation techniques

One objective of my research work is to study different mitigation techniques, trying to make them suitable for intermittent faults, or even finding a novel approach. Finally, in order to test the solutions proposed, the idea is to develop the VHDL model of a microprocessor or microcontroller system including them.

In this way, the most vulnerable hardware structures that may need protection should be identified. As stated previously, errors can be found in both storage elements and buses. For this reason, the information coded here must be protected. The first approach could be to use error detecting and correcting codes such as low-density parity check codes [18][19], linear codes (Hamming, Reed-Solomon, convolutional, …) [20], etc. combined with other techniques like scrubbing [21].

Elements like combinational circuits can be protected with different techniques: TMR, duplication and comparison, time redundancy, reconfiguration, etc.

The key question to tolerate intermittent faults is to detect faulty hardware, and either mask it or reconfigure the circuit. Possible techniques are:

- Detection: encoding, duplication+comparison, self-checking, etc.
- Masking: TMR, instruction-retry, etc.
- Reconfiguration when the fault/error is near a permanent fault/error.

Finally, other techniques like redundant execution, watchdog processors, software detection and recovery, etc. can be used.

5. Future work

There is a lot of pending work. Firstly, it is important a better understanding of intermittent fault causes and mechanisms, in order to get representative fault models. Secondly, and using these fault models, it will be necessary to carry out new fault injection experiments into the VHDL model of various microprocessors or microcontrollers running different workloads. Thus, it will be possible to deepen in the study of the influence of the parameters of intermittent faults. Thirdly, a complete study of mitigation techniques is needed. The objective is to get a set of techniques suitable for intermittent fault tolerance. Finally, the VHDL model has to be modified to

include the selected mitigation techniques, and the fault injection experiments should be repeated into the hardened model to check the validity of the proposed scheme.

As a conclusion, the main objective of my PhD research is to study intermittent faults: analyze their causes and mechanisms, deduce representative fault models, examine their effects using fault simulation and find solutions to mitigate them.

6. References

[1] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability", *IEEE Micro*, Vol. 23(4):14-19, 2003.

[2] C. Constantinescu, "Impact of Deep Submicron Technology on Dependability of VLSI Circuits", in Proc. DSN 2002, pp. 205-209, Washington D.C., June 2002.

[3] C. Constantinescu, "Impact of Intermittent Faults on Nanocomputing Devices", in WDSN-07, Edinburgh, UK, June 2007. http://www.laas.fr/WDSN07.

[4] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, vol. 16(2):166–182, 1990.

[5] D. Gil, J. C. Baraza, J. Gracia, and P. J. Gil, "VHDL simulation-based fault injection techniques", Ch. 4.1 in *Fault Injection Techniques and Tools for VLSI Reliability Evaluation*, A. Benso and P. Prinetto, Eds. Dordrecht, The Nederlands: Kluwer Academic Publishers, 2003, pp. 159–176.

[6] D. Gil, L.J. Saiz, J. Gracia, J.C. Baraza, P.J. Gil, "Injecting Intermittent Faults for the Dependability Validation of Commercial Microcontrollers", HLDVT'08, Incline Village, NV, USA, November, 2008, pp. 177-184.

[7] P.J. Gil et al., "Fault Representativeness", Deliverable ETIE2 of Dependability Benchmarking Project, IST-2000-25245, 2002.

[8] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshkavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitecture", in Procs. 40th Conference on Design Automation (DAC 2003), pp. 338 – 342, April 2003.

[9] J. Gracia, L.J. Saiz, J.C. Baraza, D. Gil, P.J. Gil, "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), pp. 80-85, Bratislava, Slovakia, April 2008

[10] C. Constantinescu, "Dependability Benchmarking using Environmental Test Tools". Procs. Reliability and Maintainability Symposium (RAMS'05), pp. 567–571, January 2005.

[11] W. Moore, G. Gronthoud, K. Baker, M. Lousberg, "Delay-fault testing and defects in deep sub-micron ICs-does critical resistance really mean anything?", Procs. IEEE International Test Conference 2000, Atlantic City, NJ, USA, October 2000, pp. 95-104.

[12] J.H. Stathis, "Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits", IEEE Trans. on Device and Materials Reliability, Vol. 1(1):43–59, March 2001.

[13] J.C. Baraza, J. Gracia, D. Gil, and P.J. Gil, "A Prototype of a VHDL-Based Fault Injection Tool: Description and Application", Journal of Systems Architecture, vol. 47(10):847–867, 2002.

[14] J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Using VHDL-Based Fault Injection to Exercise Error Detection Mechanisms in the Time-Triggered Architecture", Procs. of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC2002), pp. 316-320, Tsukuba, Japan, December 2002.

[15] D. Gil, J. Gracia, J.C. Baraza, P.J. Gil, "Impact of Faults in Combinational Logic of Commercial Microcontrollers", Lecture Notes in Computer Science, Dependable Computing – EDCC-5, Springer-Verlag GMBH, Vol.3463/2005, pp. 379-390 April 2005.

[16] http://www.oregano.at

[17] L.J. Saiz, J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Applying Fault Injection to Study the Effects of Intermittent Faults", 7th European Dependable Computing Conference (EDCC-7), Supplemental Volume, pp.67-69, Kaunas, Lithuania, May 2008.

[18] R.G. Gallager, *Low Density Parity Check Codes* Cambridge, MA: M.I.T. Press, 1963.

[19] S.K. Chilappagari, B.V. Vasic, "Fault Tolerant Memories Based on Expander Graphs", ITW'07, IEEE, Tahoe City, CA, USA, September 2007, pp. 126-131.

[20] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[21] A.M. Saleh, J.J. Serrano, J.H. Patel, "Reliability of Scrubbing Recovery-Techniques for Memory Systems", IEEE Transactions on Reliability, Vol. 39, Nº 1, pp. 114-122, April 1990.