UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

VALIDACIÓN DE LA ARQUITECTURA TTA MEDIANTE INYECCIÓN FÍSICA DE FALLOS A NIVEL DE PIN

Tesis Doctoral

Presentada por:

Dña. Sara Blanc Clavero

Dirigida por:

Dr. D. Pedro Joaquín Gil Vicente

Valencia, 2004

Cuando entre la sabiduría en tu corazón
y la ciencia sea dulce para tu alma
velará sobre ti la reflexión
y te guardará la prudencia

Proverbio

A los míos,
gracias de todo corazón

Agradecimientos:

Al Doctor D. Pedro Joaquín Gil Vicente, director de esta tesis, por su orientación y su constante ayuda.

A los directores y promotores del proyecto *Fault Injection for TTA*, el Doctor Herbert Grünbacher, del Instituto Técnico de Carinthia, y el Doctor Herman Kopetz, de la Universidad Técnica de Viena, y en general, a todos los miembros del proyecto.

A todos los compañeros del Departamento de Informática de Sistemas y Computadores (D.I.S.C.A.), y en especial a los integrantes del Grupo de Sistemas Tolerantes a Fallos al cual pertenezco.

A mi familia, porque sin ellos no hubiera podido llegar hasta aquí. A mi marido, Eduardo, por su apoyo incondicional y su eterna paciencia.

A todos, gracias.

Abstract

Dependability has so far been used as a required characteristic in order to evaluate complex or critical systems, especially those in which a failure means a risk for human life or high economical losses. Nowadays, the development of embedded systems has increased in all areas, from industrial environments to household uses. Due to the commercial expansion of embedded systems and market competitiveness, many system designers take dependability into account. Dependability evaluation and system validation has to be carried out before the functional-live phase of the product. Since an "in situ" work may require a long time because of the low failure rate of components in modern circuits, it is useful to resort to a experimental validation that generates faulty events forcing the system to deal with them according to design specifications. Fault Injection is an experimental validation method with increasing acceptance based on the realization of controlled experiments where the observation of the system behaviour in present of faults is explicitly induced by the deliberate introduction (injection) of faults into the system.

The effect of physical faults on current semiconductors, with their high operation frequency and integration density, is more aggressive than the effect obtained on devices of less advanced technologies. It can no longer be justified that a single fault only causes a single error. Consequently, it is necessary to deal with multiple errors. It has been also observed with Single Events Upsets on static and dynamic RAM memories. Moreover, thinking on the short distance existing between pads, it would be reasonable to validate the tolerance of the system against multiple faults in physically neighbouring lines.

There are many fault injection techniques and tools, among them, *Physical fault injection at pin level* is applied externally to the system and it can fulfil the requirement of not causing overhead or alteration in the execution of the code. Thus, this technique is suitable to validate complex fault-tolerant real-time distributed embedded systems. The strong temporal requirements of a real-time system, and its own condition of distributed, force us to look for non-overhead solutions that solve both, a runtime integration of the injection tool, where the system under test is never halted or delayed, and a dynamic reading of all the system events that take place at the same time but in different units.

The Time-Triggered Architecture TTA is aimed to the development of safety-critical real-time distributed embedded applications. In the TTA, fail-silence is a main concern in two domains, the time and the value domains. Fail-silence in the time domain should be guarantied by the TTP communication protocol. Fail-silence in value domain guarantees the correctness of the delivered message. This work details an important part of the experiments carried out in the course of the EU-funded IST project “Fault Injection for TTA”. It is focused on analysing the effect of faults at pin level on the TTP^{TM}/C communication controller based on the Time-Triggered Architecture (TTA), revealing weaknesses and encouraging to improve the error detection mechanisms to reach the objective of dependability.

Keywords: Dependability, Experimental Validation, Fault Injection, Time-Triggered Architecture, Physical Faults, Time-Triggered Protocol.

Resum

Els sistemes informàtics es troben presents en molts àmbits, des dels relacionats amb la indústria fins als aparells domèstics. Cada vegada amb més freqüència, un dels requisits principals a l'hora de dissenyar sistemes informàtics és que presenten un alt grau de confiabilitat, especialment aquells considerats com a crítics, ja que el seu mal funcionament pot posar en perill la integritat de les persones o pot ocasionar grans pèrdues econòmiques. A més, la confiabilitat també pot ser un factor important en la seua expansió i competitivitat en el mercat. La confiabilitat permet a l'usuari depositar una confiança justificada en el funcionament del producte, i ha de ser avaluada abans de la seua fase operacional per mitjà de la verificació i validació del comportament del sistema segons el servei especificat, tant en condicions normals com en presència de fallades. No obstant la taxa de fallades en un sistema informàtic sol ser baixa i aleshores és necessari recórrer a tècniques de validació experimental com la injecció de fallades que acceleren la validació per mitjà de la introducció deliberada i controlada de fallades en el sistema.

En general, l'efecte de les fallades físiques en els semiconductors actuals, on l'increment de la freqüència de funcionament i la densitat d'integració són notables, és més important que l'observat amb tecnologies menys avançades. Ja no és justificable suposar que una fallada simple només genera un error simple, sent necessari validar el sistema davant d'errors múltiples causats no sols per fallades localitzades en memòria , sinó també en la lògica combinacional o en soldadures i metal·litzacions, acrescuts estos últims per la reducció de la distància entre pistes.

Hi ha diverses tècniques i ferramentes d'injecció de fallades, entre elles, la *Injecció física a nivell de pin*. Un dels principals avantatges d'esta tècnica és la seua aplicació externa, no generant sobrecàrrega addicional en el sistema ni pertorbant l'execució normal de les seues tasques, la qual cosa la fa adequada per a validar sistemes encastats i distribuïts de temps real. Els estrictes requisits temporals d'estos sistemes i la seua condició de distribuïts forcen la recerca i desenvolupament de solucions que resolguen tant una integració no agressiva de les ferramentes d'injecció en l'entorn de validació, com la lectura dinàmica dels esdeveniments que ocorren en les diferents parts del sistema al mateix temps, encara que estos es troven físicament distanciats.

L'Arquitectura de Disparament per Temps (TTA, Time Triggered Architecture) és una nova arquitectura que guanya acceptació en sectors industrials tan forts com l'aeronàutica o la indústria de l'automòbil. TTA està orientada al disseny de sistemes distribuïts de temps real crítics, sent important validar els productes basats en esta arquitectura i la confiabilitat dels quals ha de garantir el protocol de comunicacions TTP. Este treball de tesi detalla una part important de la investigació realitzada durant el projecte de col·laboració Europea IST "Fault Injection for TTA". El treball se centra en l'anàlisi de l'arquitectura TTA i la validació del controlador de comunicacions TTP^{TM}/C davant de fallades físiques a nivell de pin.

Resumen

Los sistemas informáticos se encuentran presentes en muchos ámbitos, desde los relacionados con la industria hasta el hogar. Cada vez con más frecuencia, uno de los requisitos principales a la hora de diseñar sistemas informáticos es que presenten un alto grado de confiabilidad, especialmente aquellos considerados como críticos, ya que su mal funcionamiento puede poner el peligro la integridad de las personas o puede ocasionar grandes pérdidas económicas. Además, la confiabilidad también puede ser un factor importante en su expansión y competitividad en el mercado. La confiabilidad permite al usuario depositar una confianza justificada en el funcionamiento del producto y debe ser evaluada antes de su fase operacional mediante la verificación y validación del comportamiento del sistema según el servicio especificado tanto en condiciones normales como en presencia de fallos. Sin embargo la tasa de fallos en un sistema informático suele ser baja, siendo necesario recurrir a técnicas de validación experimental como la Inyección de Fallos que aceleran la validación mediante la introducción deliberada y controlada de fallos en el sistema.

En general, el efecto de los fallos físicos en los semiconductores actuales, donde el incremento de la frecuencia de funcionamiento y la densidad de integración son notables, es más importante que el observado con tecnologías menos avanzadas. Ya no es justificable el asumir que un fallo simple sólo genera un error simple, siendo necesario validar el sistema ante errores múltiples causados no sólo por fallos localizados en memoria, sino también en la lógica combinacional o en soldaduras y metalizaciones, acrecentados estos últimos por la reducción de la distancia entre pistas.

Existen diversas técnicas y herramientas de inyección de fallos, entre ellas, la *Inyección física a nivel de pin*. Una de sus principales ventajas es su aplicación externa, no generando sobrecarga adicional en el sistema o perturbando la ejecución normal de sus tareas, lo que la hace adecuada para validar sistemas empotrados y distribuidos de tiempo real. Los estrictos requisitos temporales de estos sistemas y su condición de distribuidos fuerzan la búsqueda y desarrollo de soluciones que resuelvan tanto una integración no agresiva de las herramientas de inyección en el entorno de validación, como la lectura dinámica de los eventos que ocurren en las diferentes partes del sistema al mismo tiempo, aunque en emplazamientos físicamente distanciados.

La Arquitectura de Disparo por Tiempo o TTA es una arquitectura novel que gana aceptación en sectores industriales tan fuertes como la aeronáutica o la industria del automóvil. TTA está orientada al diseño de sistemas distribuidos de tiempo real críticos, siendo importante validar los productos basados en esta arquitectura y cuya confiabilidad debe garantizar el protocolo de comunicaciones TTP. Este trabajo de tesis detalla una parte importante de la investigación realizada durante el proyecto de colaboración Europea IST FIT “Fault Injection for TTA”. El trabajo se centra en el análisis de la arquitectura TTA y la validación del controlador de comunicaciones TTP^{TM}/C frente a fallos físicos a nivel de pin.

Índice

Lista de figuras

Lista de gráficas

Lista de tablas

Glosario de Términos y Acrónimos

Abreviaturas y Acrónimos

AFIT Advanced Fault Injection Tool

BIST Built-In-Self Mechanisms

CAN Control Area Network

CDE Código de Detección de Errores

CDEV Código de Detección de Errores Verticales

CNI Controller Network Interface

COTS Commercial-Off-The-Self

CRC Cyclic Redundant Code

DICOS Distributed Control System

EDA Electronic Design Automation

EMI Electromagnetic Interferences

ET Event-Triggered Architecture

FIT Fault Injection for TTA

FPGA Field Programmable Gate Array

FTU Fault Tolerant Unit

GB Guardian Bus

HWIFI Hardware implemented Fault Injection

IFG Interframe Grap

MBE Multiple Bit Error

MBU Multiple Bit Upset

MEDL Message Description List

MT - mT MacroTick - microTick

NGU Never Give Up

ODC Orthogonal Defect Classification

PCU Protocol Control Unit

RTL Register Transfer Logic

SAE Society of Automotive Engineers

SEB Single Event Burnouts

SEE Single Event Effect

SEL Single Event Lachup

SER Single Error Rate

SEU Single Event Upset

SoC System-on-Chip

SOFI Software Fault Injector

SOS Slightly-Off-Specifications

STF Sistema Tolerante a Fallos

SWIFI Software implemented Fault Injection

TDMA Time Division Media Access

TID Total Ionization Doze

TMR Triple Modular Redundant

TTA Time-Triggered Architecture

TTP Time-Triggered Protocol

VFIT VHDL Fault Injection Tool

VHDL Very High Speed Integrated Circuit Hardware Description Language

VLSI Very Large Scale Integration

Diccionario de términos

Active state – Estado activo.

Arbitrary or inconsistent failure – Avería asimétrica.

Attributes – Atributos de la confiabilidad.

Availability – Disponibilidad.

Babbling-idiot failure – Transmisiones espurias.

Backward error recovery – Técnica de vuelta atrás.

Bit-flip – Inversión del valor de un bit.

Bit-per-byte parity – Paridad bit por byte.

Bit-per-chip parity – Paridad bit por chip.

Bit-per-multiple-chip parity – Paridad bit por múltiples chips.

Bit-per-word parity – Paridad bit por palabra.

Bridging – Puente entre dos líneas.

Built-In-Self error – Error interno detectado por ciertos mecanismos implementados a tal efecto.

Byzantine failure – Avería Bizantina.

Clique avoidance – Segregación del sistema en grupos.

Cold start state – Arranque en frío.

Commission or impromptu failure – Servicio inesperado.

Common-mode failures – Averías en modo común.

Configuration charge fault – Fallo de configuración.

Conformace test – Test de conformidad.

Control flow checking – Control de flujo del programa.

Crash failure – Avería con parada.

Crosstalk – Acoplamiento de señales.

Delay – Retardo en una señal.

Dependability – Confiabilidad.

Deterministic testing – Pruebas deterministas .

Distributed System – Sistema distribuido.

Dormant fault – Fallo dormido.

Error – Error.

Event-Triggered Systems – Sistema de disparo por evento.

Failure – Avería.

Fault – Fallo.

Fault containment – Detección, diagnosis y aislamiento de errores dentro de un nodo, evitando su propagación a otros nodos.

Fault forecasting – Predicción de fallos.

Fault hipótesis – Hipótesis de fallos.

Fault masking – Enmascaramiento de fallos.

Fault prevention – Prevención de fallos.

Fault removal – Eliminación de fallos.

Fault tolerance – Tolerancia a fallos.

Fault Tolerant System – Sistema tolerante a fallos.

Fault-silent behaviour – Comportamiento silencioso.

Firewall – Barrera de contención.

Forward error recovery – Técnica de recuperación hacia delante.

Freeze state – Estado congelado.

Functional test – Test funcional.

Golden run – Traza libre de fallos.

Hard error – Error no reversible.

Hardware-implemented fault injection – Inyección de fallos implementada por hardware.

Heavy-ion radiation – Inyección de fallos mediante fuentes de radiación.

Host error – Error relacionado con el nivel de aplicación.

Impairments – Impedimentos de la confiabilidad.

Indetermination – Indeterminación en el valor de la señal.

Init state – Estado de inicio.

Interlaced parity – Paridad entrelazada.

Interleaving – Técnica de entrelazado.

Latent error – Error latente.

Link failure – Pérdida de conexión.

Listen state – Estado de escucha.

Logic bomb – Bomba lógica (fallo malicioso).

Maintainability – Mantenibilidad.

Masquerading failure – Suplantación de la identidad.

Means – Medios para alcanzar la confiabilidad.

Mechatronic system – Componentes electrónicos para el control mecánico.

Membership service – Servicio de pertenencia.

Open-line – Línea abierta.

Passive behaviour – Comportamiento pasivo.

Passive state – Estado pasivo.

Periodic srubbing – Técnica de rastreo.

Physic fault injection – Inyección física.

Physical fault injection at pin level – Inyección física de fallos a nivel de pin.

Protocol error – Error en el protocolo de comunicaciones.

Pulse – Pulso en una señal.

Random testing – Pruebas aleatorias.

Reliability – Fiabilidad.

Safety – Seguridad-Inocuidad.

Scan chain-based fault injection – Inyección de fallos mediante cadenas de exploración.

Security – Seguridad-Confidencialidad.

Short – Puente entre dos líneas.

Simulation-based fault injection – Inyección de fallos basada en técnicas de simulación de modelos.

Slightly-Off-Specifications failure – Avería SOS (tipo de avería asimétrica).

Soft error – Error reversible.

Software-implemented fault injection – Inyección de fallos implementada por software.

Statistical testing – Pruebas estadísticas.

Stress-based injection - Inyección basada en el estrés.

Stuck-at – Pegado de la línea a un valor.

Stuck-open – Pegados en nodos en alta impedancia.

Symmetric or consistent failure – Avería simétrica.

Time redundancy – Redundancia temporal.

Time-Triggered Architecture – Arquitectura de disparo por tiempo.

Time-Triggered Systems – Sistemas de disparo por tiempo.

Trapdoor – Puerta falsa (fallo malicioso).

Trojan horse – Caballo de Troya (fallo malicioso).

VHDL-based fault injection – Inyección de fallos en modelos VHDL.

Watchdog processor – Procesador de guardia.

Watchdog timer – Temporizador de guardia.

Worm – Gusano (fallo malicioso).

X-by-wire – Sistema guiado por cable.

Capítulo 1. Introducción

1.1. Fundamentos y Motivación

En los últimos años se ha producido un gran avance en los sistemas informáticos. Cada vez más complejos, estos sistemas se encuentran presentes en muchas áreas relacionadas con la industria, la banca, los transportes, las telecomunicaciones, las instalaciones civiles, e incluso con el hogar. Algunos de ellos son críticos, ya que su mal funcionamiento puede poner en riesgo la integridad de las personas o puede ocasionar grandes pérdidas económicas. Por tanto, es necesario garantizar que siempre cumplan correctamente su función, tanto en condiciones normales como en situaciones especiales provocadas por la aparición de errores o averías. Un sistema que permita al usuario depositar una confianza justificada en su funcionamiento es un sistema confiable, y la forma de alcanzar un alto grado de confianza es mediante el diseño de Sistemas Tolerantes a Fallos. La evaluación de la confiabilidad de un sistema y la verificación de su correcto funcionamiento ante cualquier situación forman parte de la validación del sistema.

La validación de un sistema se puede realizar teórica o experimentalmente. La validación teórica requiere la resolución de un modelo teórico del sistema. La dificultad que presentan dichos modelos es la de obtener ciertos parámetros como son los coeficientes de cobertura de detección de errores de los mecanismos implementados a tal efecto y cobertura de recuperación del sistema tras la detección del error, así como los tiempos de latencia asociados. La validación experimental supera esta dificultad observando el comportamiento del sistema tanto en condiciones normales como en presencia de fallos que puedan ocasionarle errores o averías. Sin embargo, la tasa de fallos en un sistema informático suele ser baja, dilatando en exceso el periodo de observación. Las técnicas de Inyección de Fallos aceleran la validación experimental mediante la introducción deliberada y controlada de fallos en el sistema.

En general, las técnicas de inyección de fallos pueden dividirse en tres categorías: Inyección de fallos física o implementada por hardware, Inyección de fallos implementada por software e Inyección de fallos basada en simulación de modelos. Las dos primeras se aplican sobre prototipos del sistema, mientras que la simulación de modelos no precisa de su construcción. El Grupo de Sistemas Tolerantes a Fallos del Departamento de Informática de Sistemas y Computadoras de la Universidad Politécnica de Valencia, al que la autora de esta tesis doctoral pertenece, ha desarrollado herramientas de inyección de fallos en las tres categorías con el objetivo de participar en la validación de Sistemas Tolerantes a Fallos críticos. Las herramientas desarrolladas son AFIT [Gil 1992, Gil *et al*. 1997, Martínez *et al*. 1999 y Blanc 1998] en inyección física de fallos a nivel de pin, SOFI [Campelo 1999] e INERTE [Yuste *et al*. 2003 (a) y Yuste 2004] en

inyección de fallos implementada por software y VFIT [Baraza *et al*. 2002 y Baraza 2003] en inyección de fallos basada en simulación de modelos.

Uno de los productos comerciales sobre el que se ha trabajado es el controlador de comunicaciones TTP^{TM}/C basado en la arquitectura de disparo por tiempo TTA. Se trata de una arquitectura novel que está ganando aceptación en sectores industriales tan fuertes como son la aeronáutica o la industria del automóvil. TTA está especialmente diseñada para sistemas distribuidos de tiempo real, y debido a su utilización en el diseño de componentes electrónicos cuyo funcionamiento puede ser crítico en un automóvil, barco o avión, la confiabilidad de la arquitectura TTA debe demostrar un alto grado de confianza, siendo importante validar sus posibles implementaciones, como es el controlador de comunicaciones TTP^{TM}/C, antes de poder ser introducidos definitivamente en el mercado. En este sentido, el proyecto de colaboración europea FIT ("Fault Injection for TTA") ha coordinado el trabajo realizado por varias universidades europeas, bajo la supervisión de empresas relacionadas con el sector, para la validación del controlador TTP^{TM}/C, y en general de la arquitectura TTA, mediante diferentes técnicas de inyección de fallos. Una de las técnicas utilizadas durante el proyecto fue la Inyección física de fallos a nivel de pin, cuya valoración y análisis de la validación experimental realizada fundamentan este trabajo de tesis.

1.2. Objetivos

Este trabajo se enmarca dentro del proyecto europeo "Fault Injection for TTA" (IST-1999-10748) en el que participó el Grupo de Sistemas Tolerantes a Fallos del Departamento de Informática de Sistemas y Computadoras de la Universidad Politécnica de Valencia. El objetivo principal del proyecto es el de validar experimentalmente el controlador de comunicaciones TTP^{TM}/C, así como evaluar la confiabilidad de la arquitectura TTA ("Time-Triggered Architecture") y descubrir aquellas debilidades de la arquitectura que puedan poner en peligro el correcto funcionamiento del sistema que la utilice como soporte. La validación experimental se realiza mediante diferentes técnicas de inyección de fallos, llevándose un análisis transversal a la validación del controlador relacionado con el alcance y representatividad de los fallos inyectados con cada técnica. Este análisis permitirá establecer una comparativa entre herramientas basadas en técnicas de inyección de fallos y del tipo de resultados que se obtienen.

Los objetivos concretos relacionados con la validación experimental presentada son:

- Profundizar sobre el alcance e impacto de los fallos físicos, especialmente, de la representatividad de los fallos inyectados a nivel de pin. Se trata de revisar las causas que perturban la funcionalidad de los semiconductores y los efectos que tendrán sobre el sistema, identificando cuáles de estos efectos son reproducibles mediante inyección física de fallos a nivel de pin y comparar el alcance y daño de estos fallos con otras técnicas de inyección.
- Proponer una herramienta de inyección física de fallos que permita validar sistemas distribuidos empotrados. Debido a las restricciones de funcionamiento que estos sistemas de tiempo real presentan, y a su propia condición de distribuidos, son necesarias nuevas soluciones dentro de la validación experimental que no impliquen una sobrecarga en la función del sistema y que resuelvan los problemas relacionados con la integración de la herramienta en el entorno de validación. También es necesario resolver la lectura dinámica de los distintos eventos que ocurren durante un mismo intervalo de tiempo, aunque en emplazamientos físicamente distanciados.
- Análisis de la confiabilidad de la arquitectura TTA y validación del protocolo de comunicaciones TTP implementado en el controlador TTP^{TM}/C. La validación se

propone en dos dominios, el domino del tiempo y el dominio del valor. La validación de un sistema distribuido definido bajo estrictos parámetros temporales relativos a los tiempos de envío y de recepción de la información, no sólo debe asegurar que cada miembro del sistema que comparte el canal de comunicaciones cumple con las especificaciones de diseño incluso en presencia de fallos, sino también que la integración del sistema es segura garantizando que su composición final al unir sus diferentes partes no degrada o invalida ninguna propiedad que individualmente mantenga cada miembro. Por tanto, la confiabilidad de un sistema distribuido no se basa sólo en la tolerancia a fallos individual, sino también deben asegurar la continuidad del servicio en situaciones de avería ocasionadas por errores que afecten a la funcionalidad de un miembro o fallos en el propio canal de comunicaciones. Por otro lado, el dominio del valor analiza aquellas situaciones que pueden llevar a un miembro del sistema a enviar información incorrecta cuyo impacto se refleja en las acciones derivadas de los algoritmos de control soportados por cada componente electrónico. Enviar información inapropiada, siendo imposible detectar posibles errores contenidos en el mensaje, es un ejemplo de la propagación de errores a través del canal de comunicaciones que atenta contra la confiabilidad del sistema.

- Proponer modificaciones o nuevos mecanismos de tolerancia a fallos en la arquitectura TTA para corregir defectos o carencias en el diseño del protocolo de comunicaciones TTP.

1.3. Desarrollo

A partir de los objetivos mencionados, el desarrollo en capítulos del presente trabajo de tesis es el siguiente:

El capítulo 2 introduce los conceptos y términos generales relacionados con el campo de la confiabilidad en sistemas tolerantes a fallos bajo el que se enmarca el presente trabajo de tesis. Una parte relevante del capítulo está dedicada a sistemas distribuidos tolerantes a fallos, ya que son la base y motivación de la investigación realizada.

El capítulo 3 profundiza en el alcance e impacto de los fallos físicos y en su modelado mediante técnicas de inyección de fallos. Este capítulo revisa las técnicas de inyección de fallos y los trabajos de investigación más importantes publicados hasta el momento, tratando de valorar aquellas propiedades que puedan resultar orientativas en la comparación de las distintas técnicas y herramientas de inyección.

El capítulo 4 está dedicado a la arquitectura TTA (“Time-Triggered Architecture”) y los protocolos de comunicaciones basados en ella. Entre los protocolos analizados se encuentra el TTP, bajo el que se diseña el controlador de comunicaciones TTP^{TM}/C, objetivo de la validación experimental descrita y analizada en los siguientes capítulos.

El capítulo 5 describe la herramienta de inyección física de fallos a nivel de pin AFIT (“Advanced Fault Injection Tool”). Esta herramienta ha sufrido mejoras y adaptaciones desde su primera versión en 1992. La última versión de la herramienta, presentada en este trabajo, está especialmente adaptada para sistemas distribuidos.

El capítulo 6 se centra en la validación experimental del protocolo de comunicaciones TTP llevada a cabo mediante inyección física de fallos a nivel de pin sobre los dos prototipos comerciales del controlador: TTP/C-C1 y TTP/C-C2. El análisis de los resultados está enfocado a la valoración de la arquitectura TTA en el domino del

tiempo y las debilidades a tener en cuenta en cualquier protocolo de comunicaciones basado en esta arquitectura.

El capítulo 7 analiza los problemas en el dominio del valor que se pueden identificar en la implementación actual del controlador TTP^{TM}/C y propone soluciones para mejorar la cobertura de detección de errores contenidos en los mensajes enviados por los miembros del sistema.

Finalmente, en el capítulo 8 se resumen las conclusiones más importantes extraídas durante el desarrollo de esta tesis doctoral y las posibles líneas de trabajo futuras.

Capítulo 2. Conceptos Básicos

Este primer capítulo introduce los conceptos y términos generales relacionados con el campo de la confiabilidad en sistemas tolerantes a fallos bajo el que se enmarca el presente trabajo de tesis. Una parte relevante del capítulo está dedicada a sistemas distribuidos tolerantes a fallos, ya que son la base y motivación de la investigación realizada.

2.1. Confiabilidad

Definimos como **sistema** aquel conjunto de componentes de diversa naturaleza, que forma un único elemento completo, capaz de realizar por sí mismo las funciones que se le atribuyen y de interactuar o interferir con otras entidades, es decir, con otros sistemas. Un **Sistema Tolerante a Fallos**, o STF, es aquel que posee la capacidad de preservar la ejecución correcta de sus tareas, a pesar de la ocurrencia de fallos.

La **confiabilidad**[1] de un sistema es "*la propiedad que permite a sus usuarios depositar una confianza justificada en el servicio que proporciona*" [Laprie 1992, Laprie 1998 y Avizienis *et al*. 2000]. El **servicio** proporcionado es el comportamiento percibido por los usuarios, mientras que el **usuario** es otro sistema físico o humano que interacciona con el primero. La confiabilidad se caracteriza mediante **atributos**, **impedimentos** y **medios**[2].

2.1.1. Atributos

Fiabilidad, **disponibilidad**, **seguridad-inocuidad**, **seguridad-confidencialidad** y **mantenibilidad**[3] son atributos de la Confiabilidad.

Fiabilidad R (t) es confiabilidad respecto a la continuidad del servicio. La fiabilidad es la probabilidad de que el sistema tenga un servicio continuado hasta un tiempo *t* desde un tiempo inicial t_0. Si un sistema tiene una tasa de averías por hora λ, la fiabilidad en un momento determinado *t*, suponiendo una ley de fallos exponencial, será:

$$R(t) = 1 - \frac{1}{e^{\lambda(t-t_0)}}$$

La preparación para obtener ese servicio da lugar a la *disponibilidad*. La no ocurrencia de consecuencias catastróficas en el entorno da lugar a la *seguridad-inocuidad*. La no ocurrencia de revelaciones no autorizadas de información da lugar a la *confidencialidad*. La no ocurrencia de alteraciones indebidas de información da lugar a la *integridad*. La asociación de la integridad y disponibilidad respecto a acciones

[1] Dependability
[2] Attributes, impairments and means
[3] Reliability, Availability, Safety, Security and Maintainability

autorizadas, junto con la confidencialidad, da lugar a la *seguridad-confidencialidad*. La capacidad para someterse a reparaciones y evoluciones da lugar a la *mantenibilidad*.

2.1.2. Impedimentos

Los impedimentos para alcanzar la confiabilidad deseada son los **fallos**, los **errores** y las **averías**[4]. Un *fallo* es una imperfección física en el hardware o el en el software del sistema. Un *error* es un estado interno incorrecto del sistema. Es consecuencia de un fallo y puede dar lugar a una avería. Una *avería* ocurre cuando el servicio entregado por el sistema no es el especificado.

2.1.2.1. Fallos

Los fallos y las fuentes de fallo son sumamente diversas. Se pueden clasificar de acuerdo a su origen, naturaleza, momento de aparición, causa, valor o persistencia [LIS 1996, Gil 1992, Gil 1996, Ors *et al*. 2001].

Según su ***origen*** diferenciamos entre **fallos físicos** y **fallos de origen humano**. Los *fallos físicos* afectan directamente al hardware del sistema e indirectamente al software. Las causas de estos fallos las encontramos en el entorno de trabajo, por ejemplo, perturbaciones electromagnéticas, radiación, degradación de los componentes, temperatura, presión, etc. Normalmente aparecen cuando el sistema ya está en funcionamiento, pero también pueden deberse a fallos en su fabricación, debido a imperfecciones materiales. A diferencia de los fallos físicos, los *fallos de origen humano*, definidos por primera vez en [Avizienis 1978, Avizienis *et al*. 2000], tienen su origen en las "imperfecciones" humanas; se producen en la especificación, implementación o mantenimiento del sistema.

Según su ***naturaleza*** diferenciamos entre **fallos accidentales**, que aparecen de manera fortuita, o **fallos intencionados**, que aparecen intencionadamente en el sistema. Existe un grupo de fallos intencionados al que se le conoce como **fallos maliciosos**. Entre ellos están las bombas lógicas, el caballo de Troya, la puerta falsa, virus y gusanos[5] [Landwher *et al*. 1994]. Las *bombas lógicas* son partes destructivas del propio programa y sólo se activan si ocurre un evento predeterminado; el *caballo de Troya* consiste en un programa que realiza acciones ilegales aunque aparenta lo contrario y suele atentar contra mecanismos de seguridad y confidencialidad; la *puerta falsa* consiste en un acceso escondido que puede violar la seguridad del sistema; los *virus* son segmentos de programa que se adhieren a la ejecución de otro programa; y finalmente los *gusanos*, en forma de programas completos que se ejecutan en vez del programa esperado.

Según el ***momento de aparición*** encontramos **fallos de diseño o implementación** producidos durante el diseño y construcción del sistema. Son fallos humanos, aunque no intencionados. La extensión del fallo es interna puesto que se activa debido a la actividad computacional produciendo un error. Cuando el sistema ya está en funcionamiento se observan **fallos operacionales** con origen en fallos humanos, ya sean intencionados o no. Su extensión puede tener parte de interna y parte de externa, ya que pueden deberse a una mala utilización del sistema (externa) cuya posibilidad no fue contemplada en el diseño (interna). Los **fallos de configuración**[6] tienen su origen en las acciones de mantenimiento del sistema [Wood 1994] donde el servicio se ve degradado debido a alguna acción adaptativa de mantenimiento.

[4] Faults (fallos), errors (errores) and failures (averías)
[5] Logic bomb, Trojan horse, trapdoor, virus and worm
[6] Configuration change faults

La ***causa*** del fallo puede ser **externa** o **interna** al propio sistema. Un *fallo externo* se activa en el momento de su aparición. Si causa un error, el fallo ha sido efectivo. Un *fallo interno*, que se encontraba previamente **dormido**[7], puede activarse por el proceso de computación o por un fallo externo. La mayoría de fallos internos oscilan entre sus estados dormido y activo.

Según el ***valor*** diferenciamos entre **fallos determinados**, cuando el estado del sistema no cambia a través del tiempo, o **fallos indeterminados**, donde el estado del sistema sí cambia a través del tiempo.

La ***persistencia*** temporal de los fallos conduce a distinguir entre **fallos permanentes**, cuya presencia no está ligada a condiciones puntuales, sean internas (actividad computacional) o externas (entorno), y **fallos temporales**, cuya presencia está ligada a estas condiciones y, por tanto, son de duración limitada.

La noción de fallo temporal merece algunos comentarios. Los fallos temporales externos, que son originados por el entorno físico, son denominados a menudo **fallos transitorios**. Por otra parte, los fallos temporales internos son a menudo llamados **fallos intermitentes**, resultado de la presencia de combinaciones o condiciones que raramente se dan. Ejemplos de éstos son los fallos *sensibles a patrones* [LIS 1996] en las memorias semiconductoras y los procesadores. Sin embargo, los fallos intermitentes también pueden deberse al entorno de trabajo o a la degradación de los componentes hardware, donde un fallo comienza siendo intermitente para terminar en un fallo permanente [Constantinescu 2002].

Con frecuencia se encuentran situaciones que implican múltiples fallos. El tener en cuenta estos fallos lleva a distinguir entre **fallos independientes**, que son atribuidos a diferentes causas, y **fallos conexos**, atribuidos a una causa común [Avizienis y Laprie 1986]. Los fallos conexos se manifiestan con errores similares que causan **averías en modo común**[8] [Lala 1994, Kaufman *et al*. 2000, Mitra *et al*. 2000].

2.1.2.2. Errores

Los errores dependen de tres factores [LIS 1996]:

> De la *redundancia del sistema*, ya sea **redundancia intencionada** o extrínseca, que consiste en la replicación de elementos destinada explícitamente a evitar que un error derive en una avería, o **redundancia no intencionada** o intrínseca, que puede tener el mismo efecto, aunque no de forma intencionada.
>
> De la *actividad del sistema*, pudiendo existir un **error latente**[9] que nunca sea activado o por el contrario, cuya probabilidad de activación sea muy alta.
>
> De la *definición de avería* desde el punto de vista del usuario. La gravedad de una avería depende del usuario al que afecta.

La activación del error no siempre es inmediata. Un error puede permanecer *latente* cuando no se ha reconocido aún como tal. Dependiendo de los tres factores anteriores llegará a activarse, será enmascarado por la redundancia del sistema o permanecerá

[7] Dormant fault
[8] Common-mode failures
[9] Latent error

latente de forma indefinida. En un STF se espera que un error activado sea **detectado** siendo posible llevar a cabo un **diagnóstico** del daño causado y la posible **recuperación** de la parte del sistema afectada. Un error puede propagarse y generar nuevos errores antes de que sea detectado. Un STF debe garantizar que los errores no se propagarán causando una avería.

Se define como **latencia de activación** del error al tiempo transcurrido desde que se produce el error hasta que se activa. De forma análoga se puede definir la **latencia de detección** del error como el tiempo que pasa desde que el error se activa hasta que se detecta.

2.1.2.3. Averías

Las formas en las que un sistema puede averiarse se denominan **modos de avería**, que pueden describirse según los puntos de vista del dominio de la avería, la percepción por los usuarios del sistema y las consecuencias en el entorno del sistema.

Desde el punto de vista del ***dominio de la avería*** se distingue entre **averías de valor**, en las que el valor del servicio entregado no cumple con la función del sistema, y **averías de tiempo**, donde el tiempo de entrega del servicio no cumple con las especificaciones temporales esperadas [FIT 2002]. En ambos casos el servicio del sistema no es el esperado, definiendo como **función del sistema** lo que debe realizar según las especificaciones de diseño y su **servicio** como el comportamiento percibido por el usuario.

> *Las averías de valor* en sistemas distribuidos se pueden dividir entre **sintácticas** y **semánticas**. Una *avería sintáctica* implica un formato no válido en el valor enviado o recibido. Sin embargo, una *avería semántica* proporciona un formato válido pero con un valor inválido, por ejemplo, una avería sintáctica no cumpliría con las reglas de codificación, mientras que una avería semántica presentaría una codificación correcta, aunque el contenido o valor del mensaje sería erróneo desde el punto de vista de la función del sistema.
>
> *Las averías de tiempo* en sistemas distribuidos son aquellas en las que el servicio se entrega fuera del margen temporal establecido. Podemos distinguir tres posibilidades: que la entrega sea **demasiado pronto**, que sea **demasiado tarde** o que sea infinitamente tarde, identificando una entrega que **nunca** llega a ser efectiva. Existe una cuarta posibilidad denominada como **servicio inesperado**[10] y es aquella en la que el servicio entregado es un servicio inesperado o no planeado [Burns y Wellings 2001].

Desde el punto de vista de la ***percepción de la avería por los usuarios***, distinguimos entre **averías simétricas**[11], en las que todos los usuarios del sistema tienen la misma percepción de la avería, y **averías asimétricas**[12], en las que varios usuarios del sistema pueden percibir de forma diferente una avería dada. Las averías asimétricas son a menudo calificadas como arbitrarias. Cuando un sistema se diseña para tolerar averías arbitrarias no se asume ningún comportamiento específico ante un servicio entregado incorrectamente, sino que se demuestra que el sistema es capaz de tolerar una avería arbitraria probando que siempre se alcanza un estado del sistema seguro. Si por el contrario se puede llegar a alcanzar un estado no seguro, a partir del cual no es predecible

[10] Commission or impromptu failure
[11] Symmetric or consistent failures
[12] Arbitrary or inconsistent failures

el comportamiento del sistema, se dice que el sistema es tolerante a fallos asumiendo cierta hipótesis de fallos [Rushby 2001 (b)].

La gravedad de la avería es el resultado de clasificar sus ***consecuencias en el entorno del sistema***. Se puede llevar a cabo una ordenación de los modos de avería según diferentes niveles de gravedad. A cada nivel se le asocia una probabilidad de ocurrencia máxima aceptable. 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 la arquitectura del sistema. 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 derivadas de dicha avería. El primero de ellos corresponde a las **averías benignas**, donde las consecuencias son de un orden de magnitud menor o igual al beneficio obtenido por el servicio entregado. El segundo nivel corresponde a **averías catastróficas**, donde las consecuencias son inconmensurablemente superiores al beneficio obtenido por el servicio entregado. En sistemas críticos sólo se consideran como averías benignas aquellas que mantienen al sistema en un **estado seguro**, libre de inconsistencias.

2.1.3. Medios

Los medios para alcanzar una alta confiabilidad son [LIS 1996][13]:

- La **prevención de fallos**, o cómo prevenir la ocurrencia o la introducción de fallos.
- La **tolerancia a fallos**, o cómo proporcionar, a pesar de los fallos, un servicio que cumpla con las especificaciones.
- La **eliminación de fallos**, o cómo reducir la presencia (número y gravedad) de los fallos.
- La **predicción de fallos**, o cómo estimar el número actual, la incidencia futura y las consecuencias de los fallos.

Tanto la ***prevención de fallos*** como la ***tolerancia a fallos*** deben realizarse durante la fase de diseño del sistema. Los métodos de diseño y las reglas de construcción definen la *prevención de fallos*, mientras que la *tolerancia a fallos* especifica los mecanismos necesarios para evitar que un fallo en el sistema termine produciendo una avería. ***Eliminación de fallos*** y ***predicción de fallos*** son medios directamente relacionados con la validación del sistema. La *eliminación de fallos* se define como la verificación, diagnosis y mejora de los mecanismos de tolerancia a fallos implementados en el sistema (análisis cualitativo) y está orientada a detectar y reducir defectos o carencias en el diseño e implementación (construcción del sistema). La *predicción de fallos* se define como la evaluación de la eficacia del comportamiento operacional de los mecanismos de tolerancia a fallos implementados en el sistema (análisis cuantitativo) orientada a caracterizar sus **coeficientes de cobertura**, así como los **tiempos de latencia** [Constantinescu 1994, Powell *et al*. 1995, Powell *et al*. 1997] necesarios para la detección del error, localización, aislamiento y recuperación de la parte del sistema afectada, además de otras medidas globales relativas a la fiabilidad, seguridad, etc.

Tanto la *eliminación de fallos* como la *predicción de fallos* son objetivos de la validación del sistema y se pueden plantear desde el inicio de su diseño hasta su fabricación. La eliminación de fallos determina la validez que tiene el sistema en un momento dado y suele ser más útil durante los últimos pasos de la fase de diseño, dando

[13] Fault prevention (prevención de fallos), fault tolerance (tolerancia a fallos), fault removal (eliminación de fallos) and fault forecasting (predicción de fallos)

sentido al concepto de *realimentación* o vuelta a atrás. La *predicción de fallos* determina la validez que tendrá el sistema durante cierto periodo de tiempo y se aplica a la evaluación del diseño terminado. Al final de la validación ambos objetivos deben haberse cumplido, permitiendo sólo acciones de mantenimiento.

2.2. Tolerancia a Fallos en Sistemas Físicos

La Tolerancia a Fallos es "*la capacidad interna de un sistema de preservar su funcionamiento correctamente en presencia de un determinado conjunto de fallos*" [Avizienis 1976, Avizienis *et al*. 2000]. A un sistema dotado de dicha capacidad se le denomina Sistema Tolerante a Fallos o STF. Estos sistemas implementan técnicas específicas para detectar errores y diagnosticar su efecto, así como para alcanzar una posible recuperación.

2.2.1. Detección de errores

Las ***técnicas de detección de errores*** se pueden descomponer entre los **códigos de detección de errores**, la detección de errores basada en **estrategias de almacenamiento**, las **estrategias de comprobación periódicas** y las **estrategias funcionales**. Un mecanismo de detección basado en alguna de estas técnicas se implementa durante la fase de diseño como parte del propio sistema, se mantiene siempre activado o alerta y es transparente al usuario.

2.2.1.1. Códigos de detección de errores

Son códigos orientados a detectar errores en la información que se intercambia entre dos componentes. Por tanto, son mecanismos de protección ante posibles averías de valor. En función de la información que se desea intercambiar se genera un código que se agrega como parte de la información, diferenciando entre **códigos separables** o **no separables**. Un *código separable* no modifica los bits de información, sino que añade los bits correspondientes al código, mientras que un *código no separable* genera nuevos bits de información codificados o palabra de código, la cual es transmitida en lugar de los bits originales.

Un concepto frecuentemente utilizado en la implementación de códigos de detección es **la distancia mínima de Hamming** [Hamming 1950 y Wakerly 2001], que identifica el número mínimo de diferencias que deben existir entre dos combinaciones cualesquiera que cumplan las mismas reglas de codificación. En función de la distancia mínima se obtiene el número de bits erróneos que dicho código es capaz de detectar. Entre los códigos de detección de errores cabe destacar los códigos de paridad, los códigos de detección de errores unidireccionales o errores aleatorios múltiples, los códigos bidimensionales y los códigos aritméticos [Pradhan *ed*. 1986, Pradhan *ed*. 1996, Sieviorek y Swarz 1982, Sieviorek y Swarz 1992, Johnson *ed*. 1989, Iyer y Kalbarczyk 2003].

A. Códigos de paridad

Se trata de códigos separables de fácil implementación basados en identificar el tipo de paridad de los bits de información asegurando una distancia mínima de Hamming de 2. Se construye una **paridad par** si hay un número par de unos entre los bits de información. La **paridad impar** implica un número par de ceros entre los bits de información.

Como ejemplos se encuentran[14] la **paridad bit por palabra**, donde se añade un bit de paridad a los bits de información para formar la palabra codificada y la **paridad bit por byte**, donde se divide la información en dos segmentos, cada uno con un bit de paridad diferente.

Para transmisiones en paralelo de múltiples chips encontramos: la **paridad bit por chip**, donde se añade un bit de paridad por cada transmisión efectuada por un chip, resultando tantos bits de paridad por transmisión como chips. La **paridad bit por múltiples chips**, donde se añade un bit de paridad por cada conjunto de bits, perteneciendo cada bit a un chip distinto, resultando tantos bits de paridad como bits fueran transmitidos por un chip (suponiendo tamaños iguales en todos los chips). La **paridad entrelazada** para buses es igual que la paridad bit por múltiples chips pero aplicada a buses; se divide el bus en conjuntos de bits no adyacentes y se añade un bit de paridad por cada conjunto. La paridad entrelazada permite la detección de cortocircuitos entre líneas adyacentes del bus. Finalmente, la **paridad superpuesta** para la detección y corrección del error. El código de paridad está formado por más de un bit de paridad. Cada bit de información se utiliza para calcular uno o varios de estos bits de paridad. La mayoría de los códigos de paridad superpuesta aplican alguna distancia mínima de Hamming para la detección y/o corrección de errores y cuya complejidad estriba en la multiplicidad de los errores a detectar y a corregir. El más utilizado es el código **SEC-DED Hamming** que detecta errores dobles y corrige errores simples [Pradhan *ed*. 1996].

B. Detección de errores unidireccionales

Se define así a la ocurrencia de errores en múltiples bits donde todos los bits afectados cambian a un mismo valor lógico (0 o 1). Los dos recursos más utilizados en la detección de errores unidireccionales son los **códigos m-de-n** y el **código Berger**. Un *código m-de-n* es un código no separable donde existen *n* bits en la palabra de código de los cuales *m* están a uno. El *código Berger* es un código separable basado en la implementación *m-de-n*. El código Berger se calcula complementando la cuenta en binario de unos de los bits de información [Berger 1961].

C. Detección de errores aleatorios múltiples

Se define así a la ocurrencia de errores en múltiples bits donde no existe un patrón de cambio entre los bits afectados. Algunos de los códigos utilizados en su detección son los basados en la suma (o **checksum**). Su aplicación está normalmente ligada al almacenamiento en memoria o transmisiones en paralelo ya que, para construir el código, la secuencia completa de bits se divide previamente por segmentos de longitud preestablecida. El código de detección se construye mediante la suma aritmética de los segmentos (un segmento igual a un sumando).

Como ejemplos más destacables encontramos el **checksum de simple precisión**, o suma en módulo 2^n siendo n el número de bits, y el **checksum de doble precisión**, o suma en módulo 2^{2n} siendo n el número de bits. El **checksum de Honeywell** es una modificación del de doble precisión que incrementa la cobertura de detección, haciendo que un error en un bit afecte a dos bits del código de detección. El procedimiento realiza dos sumas, una con los segmentos pares y otra con los segmentos impares. Después, los dos resultados se suman formando el código de detección. El **checksum de residuo** deriva del de simple precisión. Los segmentos se suman de dos en dos. Suma los dos

[14] Bit-per-word parity (paridad bit por palabra), bit-per-byte parity (paridad bit por byte), bit-per-chip parity (paridad bit por chip), bit-per-multiple-chips parity (paridad bit por múltiples chips), interlaced parity (paridad entrelazada), overlapping parity (paridad superpuesta)

primeros segmentos y al resultado se le suma el acarreo generado en la operación. Este nuevo valor se utiliza como sumando junto con el tercer segmento, repitiendo el proceso. Finalmente, el **checksum de precisión extendida**, donde el tamaño del segmento se extiende para incluir los posibles bits del acarreo generados por la suma de todos los segmentos.

Los códigos cíclicos redundantes **CRC** se utilizan con frecuencia para la detección de errores múltiples en secuencias de bits. Sin embargo, su cobertura está orientada a la detección de errores múltiples en bits adyacentes, razón por la que se aplica con bastante éxito a la transferencia en serie de datos y no a la transferencia en paralelo. En una *Comprobación de la Redundancia Cíclica*, dada una secuencia de *k* bits, se genera un código de *n*-bits denominado secuencia de comprobación. La secuencia resultante de *k* + *n* bits debe ser divisible por algún número predeterminado. Para comprobar que no hay ningún error en la secuencia, ésta se divide por dicho número y, si el resto de la división es cero, se supone que no contiene errores. La multiplicidad de los errores a detectar dependerá de cómo se lleve a cabo la selección de la secuencia de comprobación [Stallings 2001].

D. Códigos bidimensionales

Si además de la detección de errores múltiples, ya sean unidireccionales o aleatorios, también se requiere de su corrección, el código resultante puede llegar a ser muy complejo. Los códigos bidimensionales simplifican la localización del bit o bits erróneos, resultando más fácil su corrección [Pradhan 1980, Wakerly 2001].

Para poder aplicar un código bidimensional es necesario que la secuencia completa de bits de información se encuentre dividida en segmentos de longitud preestablecida. En la figura 2.1. se representa un bloque de 6 bytes dispuestos en una matriz bidimensional. Con las filas de la matriz se calculan los **códigos horizontales**, mientras que con las columnas de la matriz se calculan los **códigos verticales**. Existe además cierta redundancia, ya que se pueden introducir códigos de verificación de los propios códigos horizontales y verticales.

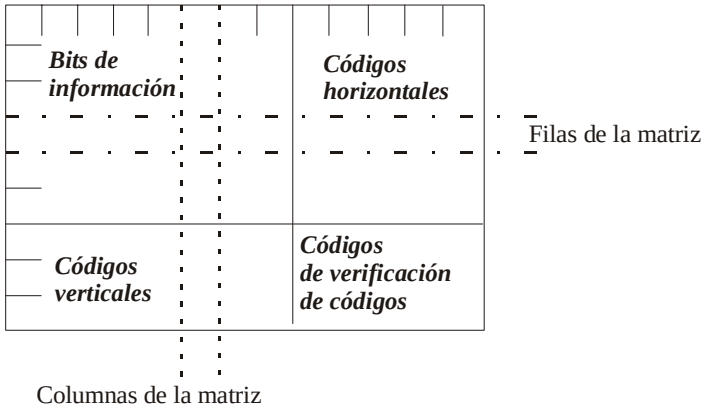


Figura 2.1. Distribución de códigos horizontales y verticales en una matriz bidimensional

El procedimiento de corrección es directo. Si se detecta un error en un código horizontal no se sabe a priori cuál es el bit erróneo, sin embargo, suponiendo que sólo hubiera una fila errónea, es posible reconstruirla utilizando la información obtenida en la comprobación de los códigos verticales. En caso de que los bits erróneos se encuentren localizados en la misma columna, podemos aplicar el mismo procedimiento a la inversa.

E. Códigos aritméticos

Los códigos aritméticos son utilizados para comprobar circuitos que realizan operaciones aritméticas. Entre éstos encontramos los **códigos AN** y **códigos de residuo**. Un *código AN* no separable es aquél que se forma multiplicando el dato original *N* por una constante *A*, distinta de cualquier potencia de la base en la que está codificado el dato. Cuando se realiza la operación aritmética con dos datos codificados, se obtiene un resultado que también debe seguir la codificación impuesta. En caso contrario se detecta un error. Dependiendo de la complejidad de la codificación es posible detectar errores múltiples, e incluso corregir errores simples.

Los *códigos de residuo* son separables y se forman con el resto que se obtiene de realizar la división entera de un número por otro número entero. Existen diferentes implementaciones dependiendo del coste máximo computacional admisible o de la multiplicidad de los errores a detectar.

2.2.1.2. Detección de errores mediante estrategias de almacenamiento

Las estrategias de almacenamiento de datos, especialmente en memorias, se pueden aplicar a diferentes niveles, mejorando la detección de errores aparecidos durante el proceso de almacenamiento o durante el propio intervalo de tiempo en el que los datos permanecen en memoria.

Nivel de estructura del sistema: Se basan en el conocimiento de que los fallos físicos que se producen en dos elementos físicamente distintos son independientes. Por lo tanto, en vez de una sola memoria se utilizan varias memorias más pequeñas. La ventaja es que la información contenida en una memoria se puede proteger con un código SEC-DED de Hamming sin que éste resulte excesivamente complejo.

Nivel de estructura del componente: Se basa en la subdivisión de la estructura interna de la propia memoria, como si de varias memorias se tratara. En principio esta técnica es menos costosa que la anterior, sin embargo, un mismo fallo físico sí puede afectar a dos subdivisiones al mismo tiempo. Esta técnica se complementa con códigos de detección bidimensionales que delimitan las subdivisiones, mejorando la independencia de éstas ante posibles errores múltiples.

Nivel de estructura de la memoria: Son métodos de almacenamiento en memoria, aunque dicha memoria fuese una subdivisión de una memoria mayor. Definen cómo almacenar los datos en memoria y se basan en el conocimiento de que un fallo físico que provoca errores múltiples siempre lo hace sobre celdas adyacentes. Entre las técnicas implementadas por software, la más conocida es la de **entrelazado**[15] [Bianchini *et al*. 1994, Baumann *et al*. 2001]. Consiste en almacenar datos consecutivos en posiciones distantes. Si un grupo de bits se protege con el mismo código y dichos bits no se encuentran en posiciones adyacentes de memoria, la única posibilidad de que exista un error múltiple que afecte a este grupo de bits es que hayan ocurrido múltiples fallos independientes. Entre las técnicas implementadas por hardware se utiliza el "**scrambled**" [McCluskey *et al*. 2001]. Los bits que lógicamente son adyacentes por pertenecer al mismo dato, o a dos datos consecutivos, no lo son en memoria (figura 2.2.). Se puede aplicar en horizontal o en vertical, dependiendo de la implementación del código de detección utilizado.

[15] Interleaving

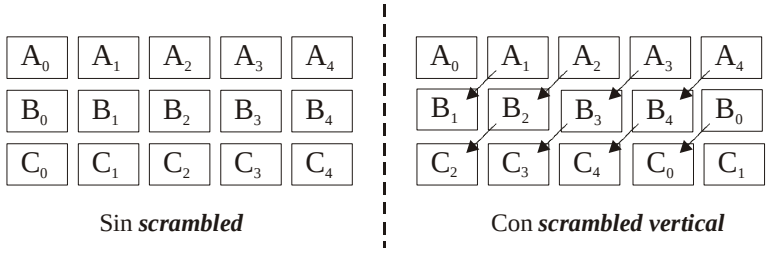


Figura 2.2. "Scrambled" vertical

2.2.1.3. Estrategias de comprobación periódicas

Se utilizan para comprobar la secuencia de ejecución de instrucciones. Podemos diferenciar entre **temporizadores de guardia**, **procesadores de guardia** y **técnicas de rastreo**[16].

Los *temporizadores de guardia* detectan errores en la ejecución de procesos a partir de la definición de un tiempo de ejecución máximo. Se puede aplicar para:

El control del tiempo de ejecución de uno o varios procesos, si a priori se conoce el tiempo máximo de ejecución del proceso y se contabiliza la duración real, se obtiene una desviación temporal. Si la desviación es mayor que el margen establecido, se detecta el error.

El control del funcionamiento del sistema, donde se genera una interrupción cada cierto tiempo que pone en marcha un proceso de comprobación general del sistema. El temporizador de guardia establece el tiempo máximo entre dos activaciones de la interrupción. Si este tiempo se sobrepasa, se detecta el error.

Los procesadores de guardia son coprocesadores que se diseñan para la detección de errores concurrentes mediante la comparación del comportamiento observado con el esperado [Serrano 1987, Mahmood y McCluskey 1982, Mahmood y McCluskey 1995].

Una de las utilidades de los procesadores de guardia es la detección de errores mediante el **control del flujo del programa**[17] [algunos ejemplos: Michel *et al*. 1991, Gil 1992, Oh *et al*. 2002 (a), Benso *et al*. 2003]. Un ejemplo es la representación del flujo del programa como un grafo de control, donde a cada nodo del grafo se le asigna un firma. El procesador de guardia debe ser informado tanto del valor de las firmas como del orden en el que se le deben entregar. Durante la ejecución del programa, el procesador compara las firmas recibidas con las esperadas diferenciando dos métodos, el de **firma asignada** y el de **firma derivada** u obtenida.

Firma asignada: A cada nodo del grafo se le asocia una firma arbitrariamente. El procesador de guardia compara la firma recibida con la esperada y en caso de ser diferentes detecta el error.

Firma derivada: Es un método más complejo, ya que el procesador de guardia tiene que calcular la firma y compararla con la recibida. Podemos diferenciar entre firmas insertadas o no insertadas. Si las firmas son insertadas, el programa se modifica para incluir las firmas. Si no lo son, ambos procesadores deben ejecutar de forma

[16] Watchdog timer (temporizador de guardia), watchdog processor (procesador de guardia) y periodic scrubbing (rastreo)
[17] Control flow checking

sincronizada el mismo diagrama de flujo, de lo contrario, el procesador de guardia detectará un error.

Las *técnicas de rastreo* periódico se complementan con códigos de detección de errores [Saleh *et al*. 1990 y McCluskey *et al*. 2001] o de detección y corrección. Cada palabra o dato almacenado en memoria (o cada bloque de datos) está protegido por un código de detección. Periódicamente se ejecuta una tarea que comprueba el código de un rango delimitado de direcciones. El objetivo de esta técnica es obtener una probabilidad de errores mínima en las instrucciones a decodificar. Las instrucciones se ejecutan sin ninguna comprobación adicional, reduciendo el coste temporal de ejecución.

2.2.1.4. Estrategias funcionales

Las estrategias funcionales están orientadas a la detección de errores en el código del programa mediante la comprobación del resultado de la ejecución de un bloque de instrucciones. Existen dos niveles de implementación, a **nivel de decodificación de instrucciones** y a **nivel de ejecución de instrucciones**.

A *nivel de decodificación de instrucciones* se comprueba que la instrucción en curso sea correcta y acorde con el juego de instrucciones. Algunos procesadores y controladores incorporan mecanismos internos capaces de detectar decodificaciones incorrectas, como por ejemplo, divisiones por cero, violaciones de segmentos de memoria protegidos, instrucciones no válidas, direcciones de memoria no válidas, etc. El tratamiento de las excepciones define las pautas en la recuperación tras la detección del error.

A *nivel de ejecución de instrucciones* se diseñan estrategias para ratificar el resultado de un proceso. Son estrategias de **redundancia temporal**[18] [Damm 1986, Johnson *ed*. 1989, Oh y McCluskey 2001 y Oh *et al*. 2002 (b)], aplicadas a secciones críticas dentro del código del programa. La redundancia temporal consiste en repetir la ejecución de estas secciones y comparar los resultados. Destaca la **doble ejecución**, donde el código seleccionado se ejecuta dos veces y se comparan los resultados de ambas ejecuciones. Si son diferentes se detecta un error. Esta estrategia sólo es válida para detectar errores transitorios, ya que si el error es permanente, ambas ejecuciones obtendrán el mismo resultado erróneo. Con la **doble ejecución con referencia** el código seleccionado se ejecuta dos veces y se comparan los resultados de ambas ejecuciones. En caso de ser diferentes, se interpreta que ha ocurrido un error transitorio. Si ambos resultados son iguales, se lleva a cabo una tercera ejecución utilizando unas constantes predefinidas. El resultado de esta tercera ejecución es conocido a priori. Si el resultado de la tercera ejecución es el esperado, se acepta que no existe error. Si el resultado de la tercera ejecución no es el esperado, se interpreta que existe un error permanente en el código. En la **triple ejecución con votación** [Schuette *et al*. 1986] el código seleccionado se ejecuta tres veces y el resultado mayoritario es el que se toma como válido. Sin embargo, no detecta errores permanentes. Esta última estrategia funcional se ha presentado en algunos trabajos, de forma más elaborada, como **redundancia software N-modular** o de **N-versiones** [Avizienis 1995 (b), Tsai 1998].

2.2.2. Procesamiento de errores

El procesamiento de errores puede llevarse a cabo de dos formas: mediante **técnicas de recuperación** o mediante **compensación**.

[18] Time-redundancy

2.2.2.1. Técnicas de recuperación de errores

Las técnicas de recuperación más conocidas son la **vuelta atrás** y la **recuperación hacia delante**[19] [LIS 1996]. La técnica de *vuelta atrás* es quizá la más utilizada y se basa en almacenar periódicamente el estado del sistema, para que, tras la ocurrencia de un error, el sistema retorne al último estado seguro que se almacenó. La *recuperación hacia delante* es una alternativa, normalmente complementaria a la técnica de vuelta atrás. Con esta alternativa se busca, si es posible, un nuevo estado para el sistema que sea seguro y que le posibilite continuar la ejecución de sus tareas.

Para poder implementar una técnica de recuperación, un paso previo es definir alguna técnica de detección de errores, para aplicar al sistema posteriormente el proceso de recuperación.

2.2.2.2. Compensación

En este caso el estado erróneo posee la suficiente redundancia como para proporcionar un servicio correcto. Las técnicas basadas en compensación utilizan la información derivada de la detección del error para eliminarlo sin alterar el estado del sistema. Podemos destacar la **redundancia activa** y el **enmascaramiento de fallos**[20].

A pesar del coste de diseño que implica replicar explícitamente ciertos elementos críticos, la *redundancia activa* es eficaz y bastante utilizada por su simplicidad [Johnson *ed*. 1989 y Pradhan *ed*. 1986, Pradhan *ed*. 1996]. Cuando se utilizan dos elementos replicados deben ser independientes para que un fallo físico no pueda afectar a ambos a la vez [Mitra y McCluskey 2001]. Los elementos replicados realizan las mismas tareas, con mecanismos de detección de errores locales a cada elemento. Si se detecta un error en un elemento, el proceso de recuperación consiste en reemplazar la función del elemento erróneo por otro sin error. Los sistemas con dos elementos realizando la misma función se denominan **sistemas duplicados con reconfiguración** si ambos están funcionando a la vez ó **sistemas duplicados con repuesto** en caso de que solamente funcione uno de los dos, actuando el otro de repuesto si se detectara un error en el primero.

El *enmascaramiento de fallos* utiliza una compensación sistemática, incluso es ausencia de fallos. Un ejemplo son los sistemas N-modular redundantes con voto por mayoría [Johnson *ed*. 1989, Siewiorek y Swartz 1992]. El problema surge cuando la mayoría es errónea. Como solución, se pueden incorporar, previo a la votación, mecanismos de detección de errores locales a cada elemento implicado en la votación. A los sistemas que utilizan tanto detección de errores (también llamados sistemas con redundancia dinámica) como votación mayoritaria (también llamados sistemas con redundancia estática) se les denomina sistemas con **redundancia híbrida**.

2.3. Validación de Sistemas Tolerantes a Fallos

A. Avizienis denomina en 1986 como **paradigma de diseño** a la abstracción y refinamiento del diseño de un sistema [Avizienis 1995 (a)]. Su finalidad es minimizar las posibles redundancias, inconsistencias y errores en las pruebas realizadas para verificar y evaluar un sistema. El paradigma se divide en tres partes: la **especificación**, el **diseño** y la **verificación-evaluación**.

[19] Backward error recovery, forward error recovery.
[20] Fault masking

La *especificación* comienza con la definición detallada de los requisitos del sistema. Es posible dividir estos requisitos en funcionales y en relacionados con la tolerancia a fallos. Los primeros se refieren a las necesidades de funcionamiento del sistema en un estado normal, libre de fallos, mientras que los relacionados con la tolerancia a fallos se refieren al comportamiento del sistema en presencia de fallos.

El *diseño* se entiende como la partición del sistema, el diseño de los subsistemas obtenidos y finalmente, su integración. El sistema se divide en subsistemas, dependiendo de los requisitos funcionales y de cuál pueda ser el efecto de un fallo y su repercusión en el comportamiento del subsistema. Durante el diseño se definen los mecanismos de tolerancia a fallos específicos para cada subsistema y durante la integración se analizan los requisitos globales y la unión de las partes.

La *verificación-evaluación* equivale a la eliminación y predicción de fallos como medios directamente relacionados con la validación del sistema. El objetivo de la eliminación de fallos es conseguir explícitamente la reducción, mediante verificación, de fallos de diseño e implementación. Se trata de identificar las debilidades en la tolerancia a fallos del sistema y corregirlas, mientras que el objetivo de la predicción de fallos es medir, mediante evaluación, la eficiencia de los mecanismos de tolerancia a fallos implementados.

2.3.1. Eliminación de fallos

La eliminación de fallos implica tanto la verificación del diseño y su implementación, como diagnosticar y corregir los defectos y debilidades identificadas. Podría considerarse como un proceso de depuración. Se observa el comportamiento del sistema bajo ciertas condiciones y se analiza si su respuesta es la adecuada. Las condiciones a las que se le somete se fuerzan a través del entorno en el que debe funcionar. Por ejemplo, mediante la generación de unos datos de entrada determinados o de unas condiciones físicas en el límite de los rangos de funcionamiento recomendados (temperatura, presión, etc.).

Las técnicas de eliminación de fallos se pueden clasificar según su propósito. Se denominan **tests de conformidad**[21] o verificación a aquellos cuyo propósito es determinar si el sistema cumple con las especificaciones. Este tipo de pruebas se realizan al final de la fase de diseño y durante la implementación del prototipo. En un STF implicarían la depuración de los mecanismos de tolerancia a fallos implementados para detectar errores. Sin embargo, el propósito también puede ser simplemente el de detectar defectos de fabricación. Es el caso de los **tests funcionales**[22] que se aplican durante el proceso de fabricación y no se consideran dentro del área de la validación de sistemas tolerantes a fallos.

Los parámetros que describen el método utilizado durante la verificación pueden ser cualitativos o cuantitativos. **Verificación cualitativa** significa la aserción, por parte el sistema, de los requisitos especificados en su definición. Se realiza mediante pruebas deterministas[23], donde se seleccionan los datos de entrada del sistema y las condiciones de entorno bajo cierto criterio. Estos datos de entrada fuerzan un flujo conocido. La respuesta final del sistema se compara con la esperada, verificando así las funciones y mecanismos implicados durante el proceso. **Verificación cuantitativa** significa la aplicación de criterios cuantitativos. Normalmente, estos criterios se describen en términos de valores máximos y mínimos para una evaluación positiva o negativa del

[21] Conformance tests
[22] Functional tests
[23] Deterministic testing

sistema. La verificación cuantitativa se realiza mediante pruebas aleatorias o estadísticas[24], donde los datos de entrada y las condiciones de entorno se generan aleatoriamente, de acuerdo con alguna distribución probabilistica [Thévenod-Fosse 1991 (a), Thévenod-Fosse 1991 (b), David y Thévenod-Fosse 1995].

2.3.2. Predicción de fallos

Consiste en evaluar los atributos de confiabilidad del sistema. También, podemos diferenciar entre **evaluación cualitativa** o determinista y **evaluación cuantitativa** o probabilística. La *evaluación cualitativa* es aquella destinada en primer lugar a identificar, clasificar y ordenar los modos de avería, y en segundo, a identificar los eventos (datos de entrada y condiciones de entorno) que dan lugar a situaciones no deseadas. La *evaluación cuantitativa* es aquella en la que se especifica la probabilidad de que cierto mecanismo detecte un error con un intervalo de confianza conocido.

La evaluación del sistema precisa de las nociones de **servicio correcto** y **servicio incorrecto**. *Servicio correcto* es aquel donde el servicio entregado cumple con la función del sistema. *Servicio incorrecto* es aquel donde el servicio entregado no cumple con la función del sistema. Una avería, es por tanto, la transición entre servicio correcto a incorrecto. A la transición entre servicio incorrecto a correcto se le denomina *reparación*. La fiabilidad, disponibilidad y mantenibilidad, atributos sobre la confiabilidad, se pueden entender mediante la cuantificación de la alternancia entre servicio correcto e incorrecto:

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

Disponibilidad: Medida de entrega de un servicio correcto considerando la alternancia entre servicio correcto e incorrecto.

Mantenibilidad: Medida del tiempo de reparación después de la última avería, o de manera equivalente, de la entrega continua de un servicio incorrecto.

En la predicción de la fiabilidad de un sistema podemos tener en consideración los términos establecidos en [Laprie 1998] sobre **fiabilidad estable** o **fiabilidad creciente**. Obtenemos una *fiabilidad estable* cuando se mantiene la capacidad del sistema para entregar un servicio correcto, por ejemplo, la tasa de averías constante que tiene el hardware. En una *fiabilidad creciente* se mejora la aptitud del sistema frente a la entrega del servicio correcto (crecimiento estocástico de los tiempos hasta la avería), por ejemplo, las mejoras introducidas en el software.

2.4. La Tolerancia a Fallos en Sistemas Distribuidos

Definimos por sistema distribuido a aquel que se compone de varias unidades de computación o **nodos** interconectados a través de un mismo canal de comunicaciones. Existe, por tanto, una lejanía física o separación entre los nodos que se encuentran distribuidos en un espacio más o menos amplio, dependiendo del tipo y características del sistema. Los datos se transmiten por el canal de comunicaciones, encapsulados en una trama según un formato predefinido en las especificaciones. A esta trama se le denomina comúnmente **mensaje**.

En un STF distribuido, cada nodo es una unidad tolerante a fallos. La probabilidad

[24] Random testing (aleatoria), statistical testing (estadística)

de que un mismo fallo físico afecte a dos nodos de la misma manera es muy baja. Por tanto, el error o errores derivados de un fallo deben **contenerse**[25] dentro del nodo. Esto es, deben detectarse, diagnosticarse y aislarse dentro del propio nodo, evitando su propagación a otros nodos.

El comportamiento del nodo ante un fallo dependerá de su gravedad. Sin embargo, durante el tiempo que el nodo permanezca dedicado a las tareas de diagnosis y contención del error, así como a la recuperación de su funcionalidad, el sistema debe continuar el servicio sin que el usuario perciba el problema. Por tanto, existe una capa de tolerancia a fallos superior a la de cada nodo en particular, la que define, en un sistema distribuido, cómo son de tolerantes a fallos tanto cada uno de los componentes individualmente como el sistema en conjunto.

2.4.1. Comportamiento funcional

Cuando un nodo perteneciente a un STF detecta un error y las tareas de diagnosis, aislamiento y recuperación no le permiten una reintegración inmediata en el sistema, la percepción sobre el estado del nodo que deben tener el resto de los nodos del sistema debe ser de **comportamiento pasivo** o en **silencio**[26], definidos por primera vez en [Powell *et al*. 1988]. En la primera, el nodo mantiene un *comportamiento pasivo*, con sus salidas a un valor constante, pudiendo ser el último valor reconocido como correcto o un valor predeterminado. En la segunda, el nodo se mantiene en *silencio*. No fuerza ningún valor en sus salidas, ni transmite ningún mensaje por el canal de comunicaciones. Ambos comportamientos evitan que el nodo afectado por el error acometa una transmisión incorrecta, en cualquiera de los dominios de tiempo o valor, generando un conflicto en el sistema mayor que la pérdida momentánea de la función del nodo.

El máximo exponente de un nodo perteneciente a un STF es poder contener cualquier error, evitando su propagación a otros nodos a través del canal de comunicaciones. Sin embargo, esto no es siempre posible. Primero, porque los fallos físicos también pueden afectar directamente al canal de transmisión o a los componentes pasivos del nivel físico de comunicaciones. Segundo, porque no podemos definir como sistema seguro a aquél que sólo ejerce tareas de tolerancia a fallos localmente en cada nodo. Por tanto, es necesario la especificación de mecanismos de tolerancia a fallos tanto en el nodo, para la contención y no propagación de errores, como en la integración del sistema, ya que independientemente de que una avería afecte a un componente, el sistema deberá cumplir correctamente su función manteniendo los atributos que lo definen como sistema confiable.

2.4.2. Hipótesis de fallos

Un STF presenta siempre una hipótesis que define explícitamente el tipo de fallos que es capaz de tolerar, especificando su tipo, número y frecuencia de ocurrencia. El sistema será seguro, según su hipótesis de fallos, si cualquier avería que en él ocurra se puede clasificar como benigna y nunca como catastrófica. Una avería catastrófica deriva de una **inconsistencia** o estado no seguro del sistema, ya que no es predecible cuál será el comportamiento de los nodos a partir de ese estado.

De la misma forma, la hipótesis incluye el comportamiento del nodo en presencia de fallos. Por ejemplo, si el nodo tiene un *comportamiento silencioso* ante fallos, significa

[25] Fault containment
[26] Passive behaviour (pasivo), fault-silent behaviour (silencio)

que sólo enviará mensajes correctos o permanecerá en silencio, considerándose como mensaje correcto aquel que sea correcto tanto en el dominio del tiempo como en el del valor. En caso contrario estaremos ante una **violación de la hipótesis de fallos**.

La hipótesis de fallos puede tener en cuenta comportamientos distintos del nodo ante un fallo, dependiendo del modo de funcionamiento en el que se encuentre el sistema. Así distinguimos entre un **modo normal de funcionamiento**, donde el fallo ocurre cuando todos los nodos se encuentran libres de fallos y un **modo degradado de funcionamiento**, donde el fallo ocurre cuando uno o más nodos se encuentran recuperándose de una avería previa.

2.4.3. Modos de avería

La clasificación de las averías que ocurren a nivel de nodo definen los **modos de avería** del sistema. Una avería localizada en un nodo puede desencadenar fallos en otros nodos, que si no se controlan a tiempo acabará en una avería general. Lo que es lo mismo, el sistema será incapaz de completar su servicio correctamente, generando un bloqueo, desconexión o una reiniciación completa, alternativas nada optimistas en sistemas clasificados como críticos.

La división más exacta de los *modos de avería* en sistemas distribuidos es la que se realiza según los dominios del tiempo o valor. Sin embargo, a veces es difícil clasificar una avería dentro de uno de estos grupos puesto que, según qué punto de vista, podría incluirse en ambos. Es más realista clasificar los modos de avería según el efecto que se observa sobre el resto de los nodos del sistema, diferenciando [Kopetz 2002] entre[27] las *averías con* **parada y reintegración**, las **averías bizantinas**, las **averías SOS**, la **suplantación de otra identidad**, las **transmisiones espurias**, o la **pérdida de conexión**.

2.4.3.1. Averías con parada y reintegración

Las *averías con parada* son el tipo de avería más común en un sistema distribuido. Ocurre cuando un nodo se desconecta del canal de comunicaciones debido a una avería interna. En el momento que el nodo vuelva a ser funcional, tratará de reintegrarse en el sistema como nodo activo para la transmisión y recepción de mensajes.

2.4.3.2. Averías bizantinas

Se observan como tales las transmisiones erróneas de un nodo. Si en la hipótesis de fallos se especifica que los nodos deben tener un comportamiento pasivo o silencioso ante fallos, las *averías bizantinas* suponen una clara violación de esta hipótesis.

Se denominan *averías bizantinas* [Pease *et al*. 1980 y Lamport *et al*. 1982] por la Armada Bizantina acampada con sus tropas asediando una ciudad enemiga. Se necesita un plan común de batalla que depende principalmente de la buena coordinación de los batallones. Los batallones se comunican mediante mensajes que se entregan por orden de los generales. Sin embargo, uno o más generales pueden ser traidores y confundir con sus mensajes. La batalla se ganará sólo si la mayoría de los generales se ponen de acuerdo para atacar.

[27] Crash failures (parada), byzantine failures (averías bizantinas), Slightly-Off-Specifications failures (SOS), masquerading failures (suplantación de otra identidad), babbling-idiot failures (transmisiones espurias), link failures (pérdidas de conexión)

Una avería bizantina puede darse tanto en el dominio del tiempo como en el dominio del valor, sin embargo siempre se consideran como tales aquellas transmisiones que son sintácticamente correctas. Por tanto, existen dos posibles tipos de transmisiones bizantinas (ver la figura 2.3. basada en [Blough y Torii 1997]): transmisiones incorrectas en el tiempo pero sintácticamente correctas y transmisiones correctas en el tiempo pero semánticamente incorrectas. Además, una avería bizantina pueden ser causa de otra **avería arbitraria** cuando exista una división en la interpretación del estado del sistema por parte de los nodos. Esta división puede generar un estado inconsistente a partir del cual sea impredecible la evolución del servicio.

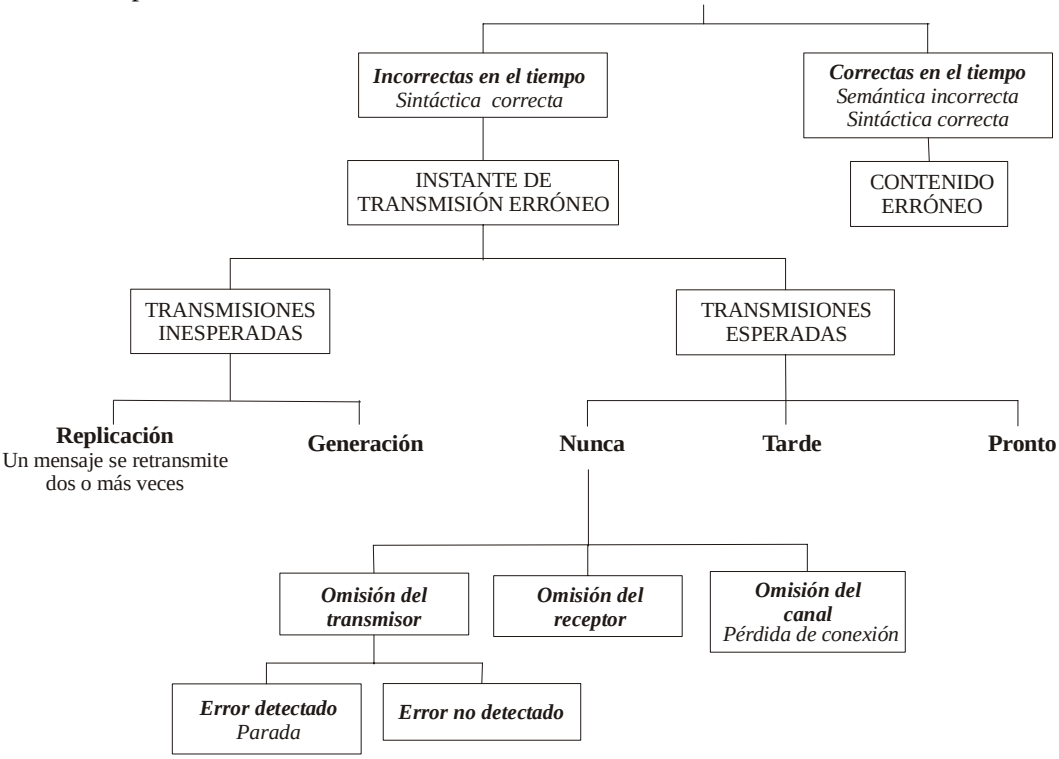


Figura 2.3. Transmisiones bizantinas

2.4.3.3. Averías SOS

Se conocen por su abreviatura *SOS* y es un caso particular de averías asimétricas. Estas averías se producen cuando la recepción del mensaje está delimitada por un patrón de tiempo o de valor.

Si existe un patrón temporal definido en la recepción, donde se especifica una **ventana de tiempo**[28] para cada mensaje, cabe la posibilidad de que esta ventana esté ligeramente desviada en uno o más nodos. Este desajuste temporal puede ser causa de una inconsistencia. Por ejemplo, un mensaje que se transmite demasiado pronto cabrá dentro de la ventana de tiempo de los nodos más rápidos, que aceptarán la transmisión como correcta. Sin embargo, los nodos más lentos abrirán su ventana con la transmisión en curso, rechazándola. Esta transmisión en sí no es errónea, y si lo fuera, debería ser reconocida como tal por todos los nodos. El problema radica en la falta de sincronismo.

[28] Una ventana de tiempo normalmente define tres valores: **apertura de la ventana** o instante a partir del cual se puede recibir una nueva trama; **momento de recepción** o instante de la ventana en el que se espera el comienzo de la recepción; y **cierre de la ventana** o instante en el que cualquier recepción debe haber terminado.

En general, una **avería SOS en el dominio del tiempo** es aquella que se origina debido a una percepción asimétrica del instante de recepción de la trama, independientemente de cuál sea el valor contenido en ella.

De la misma manera, la conversión de valores analógicos en digitales también puede provocar el mismo efecto. Por ejemplo, en el caso de una señal que llegue atenuada debido a alguna causa física. Si se recibe tan atenuada como para presentar un valor indeterminado, la conversión de analógica a digital en cada nodo puede ser distinta. En este caso, el nodo transmisor no es la causa del error, aunque algunos nodos lo marcarán como erróneo, produciéndose una división inconsistente de la interpretación del estado del sistema. En general una **avería SOS en el dominio del valor** es aquella que se origina debido a una conversión asimétrica del valor de las señales analógicas a digitales y viceversa.

La diferencia entre una avería bizantina y una avería SOS es su origen. Las bizantinas son consecuencia de un nodo averiado que lleva a cabo una transmisión incorrecta, mientras que en una avería SOS la transmisión puede ser correcta o incorrecta, ya que la avería se produce por una interpretación asimétrica de la transmisión.

Las averías SOS no se pueden considerar como una violación de la hipótesis de fallos por parte del nodo que lleva a cabo la transmisión. Para preservar la consistencia del sistema ante este tipo de averías se debe especificar alguna **estrategia NGU** ("Never Give Up"), que puede consistir, por ejemplo en que el grupo más grande de nodos con la misma percepción del problema sea el que mantenga el servicio del sistema, mientras que el resto de los nodos pasen a un estado pasivo o silencioso (estrategia definida en el TTP^{TM}/C [Kopetz y Bauer 2002]).

2.4.3.4. Suplantación de otra identidad

La *suplantación de otra identidad* implica que un nodo del sistema, que comparte el canal de comunicaciones, asume ilegalmente la identidad de otro en el envío de sus tramas. Este tipo de averías es factible especialmente en sistemas que utilizan mensajes donde se reserva un campo para el identificador del nodo transmisor.

2.4.3.5. Transmisiones espurias (o "babbling idiot")

Se caracteriza por ser una transmisión arbitraria de un mensaje en un instante aleatorio. Podemos dividirlas en dos tipos:

Consideradas dentro de las averías bizantinas: Son los mensajes sintácticamente correctos, pero erróneos en el dominio del tiempo.

No consideradas dentro de las averías bizantinas: Son los mensajes sintácticamente incorrectos, aunque correctos en el dominio del tiempo y los mensajes incorrectos en ambos dominios, que se atribuyen a fallos en el propio canal o en los componentes pasivos del nivel físico.

2.4.3.6. Pérdida de conexión

Las *pérdidas de conexión* son averías propias del canal de comunicaciones. Si un mensaje no llega a su destino, se atribuye la avería al transmisor, mientras que ésta puede ser debida a que el canal de comunicaciones está cortado temporal o permanentemente. Por otro lado, si lo que llega es un mensaje incorrecto, también pudiera ser debido a una

desconexión temporal del canal.

2.4.4. Consistencia

Un estado inconsistente en un sistema distribuido implica diferencias en la interpretación de alguna característica propia del sistema o de algún evento entre los nodos. Aunque las inconsistencias se puede definir a diferentes niveles de la arquitectura, normalmente nos referimos a inconsistencias en el protocolo de comunicaciones. Las principales causas de inconsistencia son debidas a la falta de sincronización de la base de tiempos, a las decisiones no consensuadas y a la división del sistema en los llamados grupos de comunicación.

2.4.4.1. Sincronización de la base de tiempos

Una de las características de los sistemas distribuidos es que cada procesador suelen utilizar un reloj u oscilador independiente. Esto complica la coordinación y sincronización de los nodos ante el envío de un mensaje. Si no existe una base de tiempos común, no existe posibilidad de comunicarse. La frecuencia de muestreo del canal debe ser la misma para cualquier nodo. Pero esta regla básica no es la única. Una base de tiempos global también es necesaria si existe una tarea periódica que deban realizar varios nodos a la vez. La implementación de tareas sincronizadas es muy común en sistemas distribuidos, donde los problemas a resolver son, por ejemplo, la existencia de nodos replicados que realizan lecturas sobre sensores compartidos, el acceso al canal de comunicaciones mediante una secuencia de asignación de tiempos, definición de tiempos máximos de espera en la recepción de un mensaje, etc.

Se puede conseguir una base de tiempos global sincronizando los relojes de todos los nodos. Esta sincronización puede ser interna o externa [Kopetz 1997]. En una **sincronización interna**, cada nodo lee periódicamente los relojes del resto de los nodos y aplica alguna función de corrección sobre sí mismo. En una **sincronización externa** se necesita un servidor específico cuya base de tiempos se toma siempre como referencia.

2.4.4.2. Consenso

La toma de decisiones consensuadas es fundamental en los sistemas distribuidos tolerantes a fallos. El consenso[29] consiste básicamente en que todos los nodos tengan la misma visión del estado del sistema y tomen decisiones concordantes [Kopetz y Ochsenreiter 1987, Cristian 1988, Mostefaoui *et al*. 2000, Nguyen 2001]. En el caso de que un nodo deba tomar una decisión, como por ejemplo, excluir a otro nodo por considerarlo averiado, es de esperar que todos los nodos tomen la misma decisión manteniendo una consistencia.

En el trabajo presentado por J. Gray [Gray 1978] se prueba que esta consistencia no es posible ante la pérdida de mensajes, y en [Fischer *et al*. 1985] expone las dificultades de establecer un consenso cuando no se puede delimitar una ventana de recepción conocida a priori para cada mensaje transmitido. Si el mensaje se pierde totalmente, no hay constancia de que se fuera a recibir un mensaje. Si la recepción del mensaje no se puede delimitar en una ventana de tiempo, no se puede saber si el mensaje ha llegado tarde o demasiado pronto. Este argumento es rebatido en [Schmid *et al.* 2002], donde defiende que, siempre que estas pérdidas sean debidas a pérdidas de conexión transitorias, es posible definir un algoritmo en el que interviene más de un nodo, para consensuar la

[29] Algunos autores utilizan la expresión “byzantine agreement”

correcta transmisión o la posible pérdida del mensaje.

En general, para evitar situaciones de inconsistencia ante cualquier evento no deseado, una solución es la incorporación de un patrón de tiempos que sincronice la transmisión y recepción de mensajes. Por esta razón, los llamados **sistemas distribuidos de disparo por tiempo**[30] son una alternativa, cada vez más utilizada, para la implementación de aplicaciones críticas de tiempo real.

Se denomina *sistema distribuido de disparo por tiempo* a aquel sistema, basado en una arquitectura de disparo por tiempo[31] [Kopetz y Bauer 2002], en el que el instante de transmisión de cualquier mensaje se establece a priori, y al menos es conocido tanto por el transmisor como por los posibles receptores. Esta información se amplía en el capítulo 4.

2.4.4.3. Grupos de comunicación

El envío de un mensaje conlleva una acción dinámica. Determinar los destinatarios del mensaje, la ruta empleada y cuál va a ser el mecanismo utilizado para garantizar al emisor que el mensaje ha sido recibido y aceptado.

Las transmisiones que envían un mensaje a todos los posibles destinatarios son las denominadas "**broadcast**". Si los destinatarios son un subconjunto de todos los posibles, se denominan "**multicast**". Sobre estas últimas se implementan el servicio de pertenencia, que permite el agrupamiento dinámico de los nodos en subgrupos. Todos los nodos pertenecientes a un subgrupo son destinatarios de un tipo determinado de mensajes.

La habilidad de definir la ruta de un mensaje simultáneamente a varios destinos es una característica en las redes de comunicación. Por ejemplo, con el algoritmo Multicast Backbone (MBone), realizar un envío a múltiples destinos es una realidad en Internet [Macedonia y Brutzman 1994]. Sin embargo, los algoritmos de envío a múltiples destinatarios sólo pueden garantizar que el mensaje es recibido por el/los destinos, pero no garantizan que su recepción sea consistente.

Un protocolo que garantice que *a*) todos los destinatarios reciben el mismo mensaje, no habiendo inconsistencias en el dominio del valor, *b*) todos los mensajes se tramiten y *c*) que no se transmiten mensajes espurios, no habiendo inconsistencias en el dominio del tiempo, es un protocolo fiable [Hadzilacos y Toueg 1993].

2.4.5. Técnicas para alcanzar la tolerancia a fallos

En un sistema distribuido tolerante a fallos, cada nodo es de por sí una unidad funcional tolerante a fallos. Sin embargo, ante una avería, dicha unidad no siempre es capaz de llevar a cabo un restablecimiento inmediato y en solitario de su función y por tanto no se garantiza la continuidad de servicio del sistema. Por este motivo, en sistemas distribuidos se aplican configuraciones destinadas a garantizar la continuidad del servicio ante fallos, así como una recuperación libre de averías de la parte del sistema afectada. En este apartado destacamos dos configuraciones, la recuperación distribuida y la replicación.

[30] Time-Triggered System
[31] Time-Triggered Architecture

2.4.5.1. Recuperación distribuida

Para que un nodo pueda llevar a cabo cualquier tipo de recuperación, debe ser capaz de responder en caso de una avería. Si el nodo queda bloqueado, no pudiendo forzar un restablecimiento o por lo menos una reiniciación, es que no es un nodo estable.

Tras una **recuperación local**, el nodo debe reestablecer una visión consistente del sistema (igual que la del resto de nodos), iniciando las variables utilizadas en las operaciones con valores apropiados. Si la visión que tiene un nodo del sistema tras su recuperación no es consistente, puede ocasionar que su restablecimiento implique la avería de otro nodo. Por otro lado, si los valores de las variables no se restablecen adecuadamente, podría ocasionar una discrepancia entre dos nodos replicados si uno de dos nodos es el restablecido.

Más práctica que la anterior, la **recuperación distribuida** consiste en que el reestablecimiento del nodo se lleva a cabo de una forma consensuada entre todos los nodos del sistema. El protocolo de comunicaciones establece unos mecanismos de recuperación que podrían entenderse como una conversación entre el nodo que pretende reestablecer su comunicación y los nodos que están utilizando activamente el canal. Una alternativa es que el nodo intente establecer una conversación mediante el envío de mensajes a otros nodos. Otra posibilidad es que los mensajes que se envían normalmente estén dotados de información, no sólo relativa a la aplicación, sino relacionada con el estado del sistema. Esta información es leída por el nodo que trata de recuperarse, aportándole los detalles que necesita para reestablecer su comunicación.

2.4.5.2. Replicación

Cuando no se puede suponer que los nodos sean estables tras cualquier tipo de avería, o que el proceso de recuperación de un nodo no es suficientemente rápido y eficiente como para asegurar la continuidad de servicio del sistema, la alternativa es la replicación.

Podemos diferenciar entre **replicación pasiva**, **replicación activa** o **replicación semi-activa** [Powell 1994]. En la *replicación pasiva*, el nodo replicado se mantiene a la sombra. El nodo *sombra* no recibe ni procesa la información transferida por el canal. Tampoco envía mensajes. Se limita a realizar actualizaciones periódicas de su estado en función del estado de su nodo réplica. En la *replicación activa* ambos nodos son parte activa e integrante del sistema. Ambos procesan la misma información y acceden al canal para acciones de envío y recepción de mensajes. Un nodo que recibe mensajes transmitidos desde dos nodos replicados debe implementar alguna estrategia para seleccionar sólo uno de los dos. Esta técnica es capaz de tolerar averías arbitrarias si se seleccionan los mensajes recibidos mediante votación mayoritaria [Chérèque *et al*. 1992]. En la *replicación semi-activa*, el nodo replicado recibe mensajes y procesa la información recibida. Lo único que lo diferencia de una replica activa es que no llega a enviar mensaje alguno.

2.5. La Tolerancia a Fallos y Validación Experimental

La necesidad de la validación experimental de los sistemas tolerantes a fallos viene motivada por dos aspectos [Arlat 1990 y Gil 1992]. En primer lugar, la novedad que supone cualquier diseño, aunque éste sea una nueva versión de un diseño anterior, tanto por los componentes que incorpora como por las aplicaciones destinadas al mismo. En cuanto a los componentes, el avance de la tecnología hace que tengan una validez temporal limitada. Las experiencias obtenidas sobre un componente sirven de poco para

diseños posteriores que incorporan componentes de nueva generación. En cuanto a las aplicaciones, se pueden observar situaciones similares, pero raramente son las mismas. Es más, ante cualquier cambio en la arquitectura del sistema, una misma aplicación puede observar comportamientos diferentes en presencia de fallos. En segundo lugar, las incertidumbres relativas a la patología de los fallos. A causa de la complejidad de los diseños, 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.

En cuanto a la cuantificación de la influencia de los fallos en la confiabilidad, en un STF se enfatiza sobre la confianza en el comportamiento apropiado de los mecanismos que contribuyen a dicha tolerancia, es decir, sobre la **validación de la cobertura**. La *validación de la cobertura* concierne principalmente a la validación del producto, y por tanto, de los mecanismos de tolerancia a fallos integrados. Dos tipos de parámetros nos permiten cuantificar la eficacia de dichos mecanismos: el **factor de cobertura** y la **latencia de tratamiento** [Laprie 1998]. Así se definen cuatro términos:

La *cobertura de detección* de un mecanismo es la probabilidad de que este mecanismo detecte un tipo determinado de errores.

La *cobertura de recuperación* de un componente es la probabilidad de que dicho componente se recupere tras una avería.

La *latencia de detección* es el intervalo de tiempo que separa la activación del error con su detección.

La *latencia de recuperación* de un componente es el tiempo que separa la detección del error con la recuperación del sistema.

La validación experimental puede llevarse a cabo de dos formas. Ya sea mediante **experiencias no controladas**, observando el comportamiento en fase operativa de uno o varios prototipos del sistema en presencia de fallos, o mediante **experiencias controladas**, analizando el comportamiento del sistema en presencia de fallos introducidos deliberadamente.

El primer caso, las *experiencias no controladas*, se puede considerar más realista en cuanto a coeficientes de cobertura, número de averías y coste temporal de las operaciones de mantenimiento. Sin embargo presenta dos inconvenientes que lo hacen impracticable en la mayoría de los casos. Primero, la baja probabilidad de ocurrencia de fallos. Sería necesario un tiempo inadmisiblemente alto para observar todos los posible fallos y obtener una muestra con un margen de confianza adecuado. Segundo, la validación del sistema también incluye la verificación de los mecanismos de tolerancia a fallos. Las técnicas de retroalimentación y rediseño, para la mejora o adecuación de los mecanismos de tolerancia ante un tipo de fallos determinados, son impracticables mediante experiencias no controladas.

Estos inconvenientes hacen que el segundo caso, con *experiencias controladas*, denominado **inyección de fallos**, sea más adecuado para la validación de sistemas tolerantes a fallos. La *inyección de fallos* la define con precisión J. Arlat en [Arlat 1990] como:

"Inyección de fallos es la técnica de validación de la confiabilidad en sistemas tolerantes a fallos consistente en la realización de experimentos controlados, donde la observación del comportamiento del sistema en presencia de fallos es inducida explícitamente por la

introducción (inyección) deliberada de fallos en el sistema”.

2.6. Resumen y Conclusiones del Capítulo

Este capítulo introduce los conceptos y términos generales relacionados con el campo de la confiabilidad y que van a ser de uso frecuente a lo largo del trabajo. Se estructura en cuatro partes. La primera parte está dedicada a la caracterización de la confiabilidad en sistemas tolerantes a fallos. La segunda parte trata la tolerancia a fallos en sistemas físicos, mientras que en la tercera parte se profundiza en un clase concreta de sistemas físicos: los sistemas distribuidos. Finalmente, la cuarta parte presenta una reflexión sobre la necesidad de validar experimentalmente el grado de confiabilidad de los sistemas tolerantes a fallos.

La inyección de fallos es un modo de validación experimental muy adecuado para todo tipo de sistemas tolerantes a fallos. Entre ellos, el efecto de un fallo en un sistema distribuido puede tener una doble consecuencia. Primero, relativa al comportamiento del nodo directamente afectado por el fallo, y segunda, cómo afecta ese comportamiento al resto de los nodos, y por tanto, al servicio del sistema.

Aunque un nodo sea capaz de enmascarar el fallo, o de detectar y contener el error con un tiempo de latencia suficientemente bajo, sin que altere su función, esta previsión es muy optimista. Un STF distribuido se diseña para hacer frente a comportamientos individuales menos optimistas, donde se precisa que el sistema mantenga siempre un servicio correcto. Éste es el caso de los STF distribuidos basados en una arquitectura de disparo por tiempo o TTA, especialmente orientados a aplicaciones críticas. La inyección de fallos puede resultar de gran valor para validar experimentalmente estos sistemas, siendo esta la base del presente trabajo de tesis.

Capítulo 3. Técnicas de Inyección de Fallos

Tal como se concluyó en el capítulo anterior, la validación experimental de sistemas tolerantes a fallos mediante experiencias controladas resulta de especial interés para la verificación y evaluación de la confiabilidad de un producto antes de que llegue al mercado, y por tanto, antes de su fase operacional. La inyección de fallos engloba aquellas técnicas de validación experimental basadas en la introducción (inyección) deliberada de fallos en un sistema, orientadas tanto a verificar el correcto servicio del sistema como a evaluar y mejorar sus atributos revelando posibles defectos y carencias a lo largo de todas las fases de su ciclo de vida.

La exposición de cualquier sistema durante su fase operacional a posibles fallos físicos hace que estos fallos tengan un especial interés en la validación experimental. Las causas que pueden provocar un fallo físico en un circuito integrado son diversas. Por eso, este capítulo revisa las causas que favorecen la aparición fallos físicos en los circuitos integrados y cómo se puede modelar su efecto mediante técnicas de inyección.

3.1. Fallos Físicos: Causas y Modelos

Los fallos físicos se pueden deber a causas externas como, por ejemplo, interferencias electromagnéticas (EMI), variaciones en la tensión de la fuente de alimentación, radiación o el efecto de las condiciones ambientales de trabajo. También se pueden deber a causas internas como el desgaste o los residuos acumulados en el proceso de fabricación. La mayoría de los diseños prevén la aparición de algunos de estos problemas. Así por ejemplo, la transmisión diferencial mejora la inmunidad de la señal transmitida al ruido, o la utilización de fuentes de alimentación independientes mejora la efectividad de las barreras de contención que evitan la propagación de errores. Sin embargo, con respecto a la radiación o las condiciones ambientales de trabajo, las propias características de los circuitos integrados de nueva generación los hacen menos inmunes y más susceptibles a los fallos.

Por tanto, cabe asumir que en un circuito integrado van a ocurrir fallos físicos a lo largo de su vida operacional con casi total seguridad. La ocurrencia de dichos fallos es inevitable, pero no que dichos fallos, al transformarse en errores, lleguen a provocar averías, o lo que es lo mismo, situaciones no deseables. Cerciorar que en un diseño tolerante a fallos el porcentaje de averías esté por debajo de un umbral conocido es objetivo de la validación experimental llevada a cabo mediante técnicas de inyección.

Pero para una correcta utilización de las técnica de inyección es necesario un doble análisis. Primero, entender cuál es el efecto de las causas de fallo consideradas sobre los circuitos integrados que se fabrican actualmente. Segundo, definir modelos de fallos que sean representativos a un nivel apropiado. A este nivel se aplicará la inyección de fallos.

3.1.1. Causas de los fallos físicos en los circuitos integrados

Los circuitos integrados presentan en la actualidad una densidad de integración que prácticamente se está duplicando cada dos años[32] [Constantinescu 2002]. Este incremento en la densidad de integración viene acompañado de un aumento importante en las frecuencias de trabajo. Ambos parámetros son ventajosos a la hora de diseñar nuevos circuitos más potentes y pequeños, pero también hacen que sea necesario reducir la tensión de alimentación para aliviar el calentamiento y consumo del integrado. Todos estos factores incrementan la vulnerabilidad de los circuitos a los efectos de la radiación (neutrones, protones y partículas α) a cualquier altitud, incluso a nivel del mar [Hazucha y Svensson 2000, Constantinescu 2002, Shivakumar *et al*. 2002].

Otro factor importante resulta de las condiciones de trabajo. A medida que se puedan desarrollar circuitos más potentes, podrán utilizarse en aplicaciones más complejas. Por ejemplo, en automoción, aviónica, transportes en general, industria espacial, etc. donde las condiciones ambientales a las que quedan expuestos los circuitos son muy adversas (cambios bruscos de temperatura y humedad, mayor exposición a radiación cósmica, ruido, riesgo de desconexión de los cables debido al movimiento o a golpes, mayor desgaste, etc.).

En esta sección se desglosan los efectos derivados de diferentes causas de fallos según su localización en un circuito integrado. Este desglose resulta muy útil si queremos entender mejor cómo se debe aplicar un modelo de fallos en el análisis de una parte concreta del circuito.

3.1.1.1. Efectos de la radiación en los semiconductores

Cuando una partícula con carga positiva incide en un semiconductor, se produce una **derivación** de los electrones de valencia en la zona ionizada. Este movimiento permite a los electrones de niveles de energía inferiores alcanzar niveles superiores en el substrato, generando a su vez huecos. Esta generación produce nuevas recombinaciones electrón-hueco hasta que se alcanza una estabilidad. Este segundo proceso es conocido como **difusión** de electrones en el substrato, proceso más lento que la derivación.

Ambos fenómenos de derivación y difusión conllevan que algunos electrones alcancen la banda de conducción del semiconductor, provocando un aumento o pico de corriente inesperado. Si el transistor está conduciendo, se invierte el flujo de electrones, provocando un efecto contrario al deseado. Los semiconductores más sensibles son los de puerta flotante (DRAM, SRAM y biestables), mientras que las memorias “flash” son inmunes a los efectos de la radiación directa [Baumann *et al*. 2001].

Por ejemplo, en una DRAM, si el 0 lógico implica la no conducción (transistor no excitado), el efecto de la radiación sólo provoca el cambio de 1 a 0 y no de 0 a 1, aunque esta regla es aplicable tan solo al transistor de almacenamiento. Sobre el transistor de selección, el efecto puede ser cualquiera, de 0 a 1 o de 1 a 0. Con frecuencias inferiores a 66MHz predomina el efecto sobre el transistor de almacenamiento. A medida que se incrementa la frecuencia, se produce un incremento equitativo de la frecuencia de refresco, incrementando a su vez el tiempo total en el que el transmisor de selección actúa. Por tanto, su sensibilidad es mayor a altas frecuencias [Baumann *et al*. 2001]. En una SRAM, la estructura de una celda con 6 transistores o 6T es inmune a la radiación durante operaciones de escritura, a menos que afecte directamente al circuito de

[32] Tal como afirmó G. Moore en 1965 (Ley de Moore)

selección. Pero no lo es durante el almacenamiento, donde la celda es muy vulnerable [Palau *et al*. 2001].

Algunos conceptos interesantes relacionados con la incidencia de la radiación en los semiconductores son:

TID [LaBel 2000] ("Total Ionization Dose"): Es el factor utilizado para delimitar la vida operacional de los circuitos integrados expuestos a radiación. Este factor se calcula en función del nivel de radiación que debe acumular el semiconductor para producir una degradación irreversible del componente.

SEE [LaBel 2000] ("Single Event Effect"): Se denomina así a la observación de un cambio en el semiconductor debido al impacto de una partícula o ion. Dependiendo del efecto que causan, estos eventos se clasifican en:

- **SEU** ("Single Event Upset"): Si una partícula, por ejemplo, un protón, incide sobre un semiconductor, altera la posición de los electrones de valencia del substrato pudiendo provocar un cambio en el nivel lógico del transistor afectado.
- **SEL** ("Single Event Latchup"): Estos eventos se producen en tecnología CMOS debido a la existencia de parejas de transistores (tipo *p* – tipo *n*) no utilizados en las memorias y registros. Son aquellos transistores que en condiciones normales permanecen inactivos. Un pulso espurio derivado del impacto de una partícula puede activar una pareja de estos transistores, generando un cortocircuito entre las tensiones de alimentación y referencia. Si la corriente está limitada se evitarán daños permanentes y el efecto desaparecerá desconectado la alimentación del componente.
- **SEB** ("Single Event Burnouts"): Es el efecto de un SEL cuando la corriente no está limitada. El daño puede ser irreversible.

Mientras el efecto de la radiación desaparezca y la celda no quede dañada, estaremos ante un error reversible. Sin embargo, no todos los errores son reversibles. En [Koga *et al*. 2001] se describe la posibilidad de provocar pegados permanentes de las celdas de memoria que no son efecto de un SEL o SEB. Debido a una exposición continua a la radiación, una celda puede resultar dañada irreversiblemente. El primer impacto de un ion sobre una celda de memoria no provoca un error permanente, pero sí la acumulación de cierta dosis de radiación.

Respecto a la aparición de fallos permanentes debidos al proceso de fabricación, en general la calidad de los diseños y de las técnicas de fabricación han reducido la ocurrencia de estos fallos sobre memorias DRAM y SRAM en la última década, tal como muestra el ejemplo de la figura 3.1. obtenida de *Telcordia Technologies, Reliability Procedure for Electronic Equipment* [Constantinescu 2002].

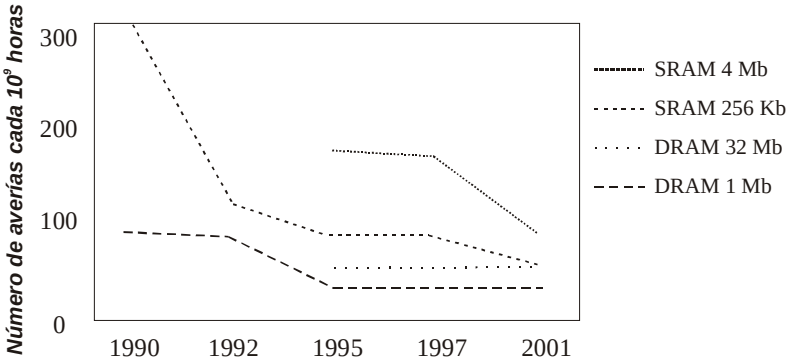


Figura 3.1. Ejemplo comparativo de fallos permanentes en DRAM y SRAM derivados del proceso de fabricación

La radiación de partículas en los semiconductores puede ser debida a tres motivos, los emisores naturales de partículas alfa, las radiaciones cósmicas y algunos isótopos con comportamientos especiales.

A. Emisores naturales de partículas α

Los isótopos radiactivos naturales son emisores de partículas α. El ^{232}Th, ^{238}U, ^{210}Po, etc. forman parte de las impurezas de los semiconductores. Una partícula α es un núcleo de helio con una energía de entre 4-9MeV. Esta energía hace que la partícula se pueda mover por el substrato de silicio una distancia de entre 10-100μm. La incidencia de la partícula en el substrato crea una zona ionizada a lo largo de su trayectoria, originando el fenómeno de derivación y la consiguiente difusión [Baumann *et al*. 2001]. La forma de evitar las consecuencias de estos isótopos es mejorar el proceso de fabricación para reducir las emisiones por debajo de los 0,002 átomos/cm^2-hr.

Un efecto de especial interés para los encapsulados de tipo CDA ("Direct Chip Attach"), CSP ("Chip Scale Package") y BGA ("Ball Grip Array") es el del isótopo radiactivo ^{210}Pb que difícilmente puede ser separado del isótopo estable ^{206}Pb. Este isótopo es causa de problemas en la superficie de contacto entre la soldadura y el metalizado del encapsulado: UBM ("Under Bump Metalization"). El isótopo radiactivo es inestable y en su cadena de reacciones, hasta estabilizarse en ^{206}Pb, se observa la emisión de una partícula α de 5.3MeV [Baumann *et al*. 2001]:

$$^{210}\text{Pb} \xrightarrow{\beta^-} {}^{210}\text{Bi} \xrightarrow{\beta^-} {}^{210}\text{Po} \xrightarrow{\alpha} {}^{206}\text{Pb}$$

B. Radiaciones cósmicas

Los iones pesados basan su incidencia sobre los semiconductores debido a las radiaciones cósmicas. Se consideran de importancia aquellos que inciden con una energía superior a 1MeV.

Por ejemplo, un neutrón que colisione contra el núcleo del átomo de silicio ^{28}Si, con una energía superior a 1MeV, provoca un desplazamiento del núcleo. Los electrones del átomo de silicio son "expulsados" de sus posiciones obligadas generando los correspondientes pares electrón-hueco. El fenómeno que se produce es el de difusión [Baumann *et al*. 2001]. Es más, dependiendo de la energía con la que el neutrón incide en el semiconductor, el ^{28}Si + n puede ser fuente de nuevas **partículas secundarias**: átomos de otros componentes + protones, neutrones o partículas α. Por ejemplo:

$^{25}Mg + \alpha$; $^{21}Ne + 2\alpha$; $^{28}Al + p$; $^{27}Al + d$ (protón + neutrón); $^{24}Mg + n + \alpha$; etc.

El ángulo de incidencia y trayectoria de las partículas secundarias es independiente de la tecnología de diseño del semiconductor. Los transistores de tipo *n* son normalmente más sensibles a la derivación que los de tipo *p*, aunque para ciertas trayectorias resultan igualmente sensibles [Castellani-Coulié *et al*. 2001]. El efecto de los neutrones y el de los protones es muy similar [Wrobel *et al*. 2001].

En la radiación cósmica, el encapsulado y estructura del componente juega un papel muy importante como medio aislante. Algunas empresas, como Honeywell [Kaakani 2001], están teniendo en cuenta la necesidad de esta protección.

C. Isótopos especiales: La fisión del Boro

Es isótopo de ^{10}B es una impureza que se encuentra en los semiconductores de tipo *p*. Aunque normalmente la concentración de ^{10}B es baja en comparación con ^{11}B, no puede descartarse. La fisión del ^{10}B es de interés por la alta probabilidad del isótopo a aceptar un neutrón. Es más, el neutrón no requiere una energía mayor de 1MeV (incluso menor) [Baumann *et al*. 2001]. La incidencia de un neutrón de baja energía sobre el núcleo de ^{10}B degenera en una nueva combinación de $^{7}Li + \alpha$.

3.1.1.2. Efectos de la radiación en memorias SRAM

Analizamos los efectos de la radiación en memorias SRAM ya que es el tipo de memoria que encontraremos en nuestro prototipo del TTP^{TM}/C.

Los efectos de la radiación en la memoria SRAM se agravan a medida que se aumenta la densidad de integración y se reduce la tensión de alimentación. Los nuevos diseños tienen mayor sensibilidad a la radiación, incluso a nivel del mar. Por ejemplo, en la figura 3.2. se muestra la sensibilidad de una CMOS SRAM de 0,18μm 6T, en una escala del 0 al 100, en relación con el voltaje [Baumann y Smith 2000]. En la figura 3.3. se muestra el efecto de partículas α y neutrones sobre dos SRAM, una de 0,25μm a 2V y otra de 0,18μm a 1,6V [Constantinescu 2002].

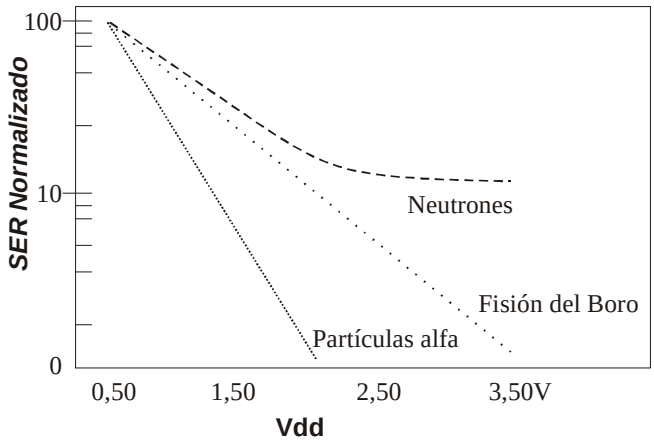


Figura 3.2. Sensibilidad ante SER *vs* Vdd

SER ("Single Error Rate") indica la frecuencia de aparición de errores reversibles sobre la memoria extrapolada entre 100 (el máximo) y 0 (el mínimo). Aunque las energías de incidencia son diferentes según el tipo de partícula, en la gráfica se aprecia el aumento general de la sensibilidad de la SRAM a medida que se baja el nivel de tensión

de alimentación.

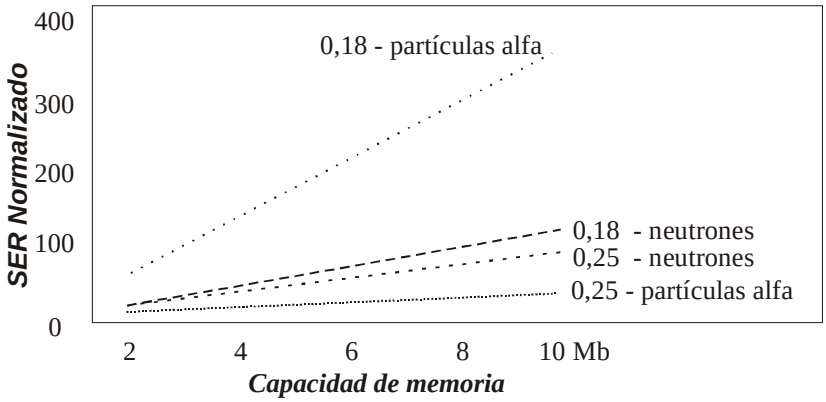


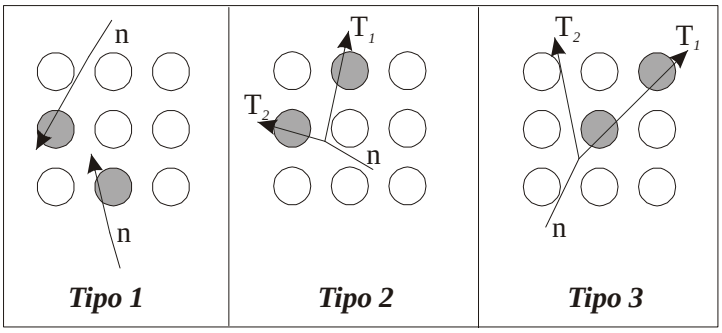
Figura 3.3. Sensibilidad ante SER *vs* capacidad de memoria

El número de transistores (y por tanto de bits) que pueden verse afectados por una incidencia simple aumenta con la densidad. Existen tres tipos de incidencias que provocan un efecto negativo en más de un transistor. A este efecto múltiple se le denomina **MBU** ("Multiple Bit Upset"):

- Que la incidencia no sea simple, sino de varias partículas al mismo tiempo (figura 3.4. – tipo 1) . Sin embargo, la probabilidad de que esto ocurra es muy pequeña. Por ejemplo, para un flujo de protones Φ (partículas/ ($cm^2 \cdot s$)), donde $\Phi = 10^5$ cm^{-2} s^{-1}, la probabilidad de una incidencia simple es de $5 \cdot 10^{-10}$, mientras que la probabilidad de dos incidencias es de $2{,}5 \cdot 10^{-19}$ [Wrobel *et al*. 2001].

- Que el ^{28}Si + n genere más de una partícula secundaria. Cada partícula traza un camino independiente (partículas secundarias T_1 y T_2 figura 3.4. – tipo 2).

- Que una misma partícula secundaria alcance más de una unión *n-p* o *p-n* pertenecientes a celdas diferentes (partícula secundaria T_1 figura 3.4. – tipo 3). Este problema se acentúa a medida que aumenta la densidad de integración de las memorias y las uniones se encuentran a menor distancia.



T*i*: Trayectoria partícula secundaria *i* generada a partir de ^{28}Si + n

Figura 3.4. Efectos múltiples originados por una incidencia simple

Por tanto, un MBU puede derivar en múltiples errores. Cada error corresponde a una **inversión**[33] del valor contenido en una celda de almacenamiento (un bit). Aunque es importante remarcar que, ya que los tipos 2 y 3 son los más probables, en caso de que se produzcan Múltiples Bits Erróneos (**MBE**) siempre serán en *celdas adyacentes*.

Entre los trabajos experimentales donde se han detectado errores múltiples encontramos [Musseau *et al*. 1996, Reed *et al*. 1997] y [Barak *et al*. 2000]. O. Musseau y R.A. Reed trabajan en el mismo equipo y ambos artículos están relacionados. Describen casos prácticos en los que se han observado MBU del tipo 2 y 3. Para que ocurra un MBU de tipo 3, el ángulo de incidencia de la partícula debe estar cerca de los 90°. Además, en sus experimentos observan un nuevo efecto, el acoplamiento entre uniones físicamente próximas. Si las uniones pertenecen a celdas adyacentes se obtiene un MBU. En cualquier caso, el número máximo de celdas afectadas son cuatro, dos en el eje-X y dos en el eje-Y.

J. Barak [Barak *et al*. 2000] describe un caso práctico sobre un satélite en la órbita terrestre. Las latitudes que presentan un mayor flujo de iones pesados son las correspondientes al Ecuador (campanas de protones de Van Allen) y la Antártida. Los paneles de control incluyen tres tipos de memorias SRAM de doble puerto. En una de ellas, la IDT7006 (a 5,0 V) se detectaron, durante un año de órbita, 5 MBE *dobles* en posiciones adyacentes.

3.1.1.3. Efectos de la radiación en la lógica combinacional

Un fallo en la lógica combinacional debido al efecto de la radiación sólo resultará en un error si llega a producir una lectura o escritura incorrecta en registros o memoria. La proporción de errores derivados de la lógica combinacional es inferior a aquellos que se generan directamente en el elemento de almacenamiento, ya que muchos son enmascarados y no llegan a producir ningún error [Baraza 2003]. Podemos diferenciar tres tipos de enmascaramiento [Shivakumar *et al*. 2002]:

Enmascaramiento lógico: Si la parte afectada por el fallo no se tiene en consideración o no tiene peso en la operación lógica correspondiente.

Enmascaramiento eléctrico: Cuando el fallo genera un pulso que es atenuado por las propias puertas lógicas hasta el punto de no tener efecto en el resultado final.

Enmascaramiento temporal: Cuando el fallo genera un pulso erróneo en la entrada de un biestable síncrono pero, debido a su carencia de sincronización con el reloj, no cumple con tiempos de $t_{set\text{-}up}$ y t_{holt}. Si el pulso erróneo comienza antes de $t_{set\text{-}up}$ y finaliza después de t_{holt} será capturado provocando un cambio de estado no deseado en el biestable. Sin embargo, en caso de no cumplir alguno de estos tiempo, el fallo puede resultar enmascarado.

El efecto que tiene la incidencia de una partícula sobre una región sensible es la de producir un pulso con un flanco de subida pronunciado y un flanco de bajada más suave. Las ecuaciones para el cálculo de dichos flancos se encuentran en [Freeman 1996, Hazucha y Svensson 2000] y dependen de la corriente generada y de la tecnología de diseño utilizada. Este pulso puede entrar en conflicto con una señal transferida, en cuyo caso se producirá una deformación de dicha señal, por ejemplo, alterando su nivel de tensión o produciendo una variación en el cambio de valor de la señal a la entrada del biestable [Shivakumar *et al*. 2002]. En cualquier caso, el efecto del pulso sobre la señal se

[33] Bit-flip

traduce en un estado metaestable transitorio seguido de un posible error [Texas Instruments 1997].

3.1.1.4. Efectos debidos al deterioro, desgaste o condiciones ambientales

Las nuevas tecnologías presentan una variación de dimensiones relacionadas con las líneas de conexión, el grosor de las capas de metalizado y dieléctrico, la distancia de separación entre líneas, etc. que agravan los problemas físicos internos que normalmente se presentan en circuitos integrados, produciendo fallos, ya sean permanentes o transitorios.

A. Fallos intermitentes y permanentes

El propio proceso de fabricación puede originar defectos como microfracturas, puentes entre líneas, defectos en la estructura cristalina o en el encapsulado, impurezas, etc. Algunos de estos defectos no son fuente de problemas hasta que el dispositivo está en funcionamiento durante algún tiempo. Otros fallos aparecen por el propio uso y desgaste del dispositivo, al margen del proceso de fabricación. Además, algunos fallos permanentes derivan de fallos intermitentes que poco a poco incrementan su frecuencia de aparición, hasta transformarse en fallos permanentes.

Algunos de los problemas relacionados con los transistores son [Pradhan 1986, Amerasekera y Najm 1997, Gil 1999, DBench 2002]:

- El **estrés eléctrico,** con causas muy diversas, produce sobretensiones o sobrecorrientes de larga duración y descargas electrostáticas de menor duración. También, una alta intensidad de campo aumenta la energía de los electrones, causando que estos ***electrones calientes*** adquieran mayor movilidad y queden atrapados en la capa *thinox* de los transistores MOS produciendo cierto deterioro que, al igual que la **contaminación iónica** de esta capa fina de óxido, dan lugar a variaciones en la tensión umbral del transistor.
- Estos y otros motivos producen el **deterioro o rotura de la capa de óxido** en tecnologías MOS que puede producir una excesiva corriente de fuga en la puerta del transistor, un incremento de la disipación de potencia estática o afectar a la velocidad de conmutación de las puertas.
- Finalmente, la movilidad de los electrones en el semiconductor se ve afectada por los cambios de **temperatura**, produciendo retardos en las señales.

Relacionados con las metalizaciones internas son [Amerasekera y Najm 1997, Gil 1999, Constantinescu 2002]:

- La reducción en los espesores de las líneas y el grosor de las capas de metalizado y dieléctrico, junto con el aumento de la densidad de corriente y de la temperatura, hacen que el fenómeno de la **electromigración** sea cada vez más frecuente. La electromigración aumenta la resistencia en las líneas y el desplazamiento de átomos de metal que acaban formando flecos[34] o rebabas.
- La ínterdifusión de átomos o **migración** en la unión aluminio-silicio es debida a la diferencia entre los coeficientes de expansión térmica de los

[34] Whiskers

elementos. La migración de átomos del silicio hacia el metal produce vacíos en el contacto, aumenta la resistencia y finalmente el contacto se fractura. La migración de átomos hacia el silicio hace que se formen puntos y hendiduras (desplazamiento del metal), aumentando las corrientes de fuga. Un efecto similar se puede producir en las vías, aunque los elementos implicados son los metales que conforman la vía, ocasionando fracturas en el contacto. Tanto la humedad como la contaminación iónica son aceleradores de la **corrosión** que aumenta la resistencia de la unión y favorece la migración de átomos del metal.

- Otro problema que puede aparecer durante el proceso de fabricación es el **estrés mecánico** que deforma las capas de metalizado del dispositivo.

B. Fallos transitorios

De los trabajos publicados por [Amerasekera y Najm 1997, Gil 1999, Constantinescu 2002] podemos extraer las siguientes apreciaciones sobre fallos transitorios:

- La reducción en los espesores de las líneas tiene como efecto negativo **el aumento de su resistencia**. Parejo a la reducción del espesor también se disminuye al máximo el espacio entre pistas con el objetivo de reducir el tamaño del circuito. El efecto negativo es el **incremento del acoplamiento capacitivo** [Chen *et al*. 1997, Metra *et al*. 1998 y Chen *et al*. 2002]. En general, aumentar resistencia y capacidad influye en los retardos de los flancos de subida de las señales. Además, el acoplamiento capacitivo tiene un especial impacto cuando dos líneas adyacentes conmutan simultáneamente a valores opuestos, produciendo un retardo en el cambio de valor, efecto conocido como **Miller**.
- El aumento de la inductancia a altas frecuencia produce un segundo efecto conocido como efecto **piel** o "**skin**": la señal se propaga por la parte más superficial de la línea de conducción. Este efecto hace que la resistencia de las interconexiones varíe en relación con la frecuencia, siendo difícil predecir los retardos de las señales y por tanto, incurriendo en violaciones de los tiempos de $t_{set\text{-}up}$ y t_{hold}.
- También las **interferencias electromagnéticas**, ya sean radiadas o conducidas, deben ser tenidas en cuenta. El encapsulado del dispositivo puede aislar, hasta cierto punto, al circuito del ruido radiado, por ejemplo de la luz. Sin embargo, otras interferencias se introducen en el circuito a través de los pines de entrada y salida, siendo su aislamiento más complejo. El ruido conducido afecta a los niveles de tensión y a alta frecuencia puede generar nuevas perturbaciones debido al acoplamiento de la señales[35].
- Otro efecto negativo deriva de la **variación en la tensión de alimentación**. Esta situación es causada por la conmutación simultánea de varias puertas, provocando variaciones de la corriente en la fuente (el consumo dinámico aumenta al aumentar la frecuencia de conmutación) y como consecuencia, provocando variaciones de tensión inducida en las líneas de alimentación.
- El **aumento de la temperatura** que se deriva de la densidad de integración, también puede provocar un aumento significativo de pares electrón-hueco que se traduce en un aumento de la corriente inversa de saturación en las uniones *p-n*, lo que a su vez puede afectar a los valores de tensión y corriente que definen los niveles lógicos.

[35] Crosstalk

3.1.1.5. Efectos debidos al deterioro de las soldaduras

Finalmente, de los trabajos publicados por [Amerasekera y Najm 1997, Patterson 2002, Balkan *et al*. 2002] podemos extraer las causas que provocan fallos directamente en los pines de entrada y salida de los circuitos integrados. La persistencia del fallo depende del nivel de desgaste o corrosión de la unión metal-substrato o metal-soldadura.

- La **fatiga de la soldadura** deriva de la diferencia entre los coeficientes de expansión térmica de los elementos metálicos que conforman la unión, produciendo un fuerte estrés y las consecuentes fracturas.
- Al igual que en las metalizaciones internas, tanto la humedad como la contaminación iónica son aceleradores de la **corrosión** que aumenta la resistencia de la unión y favorece la migración de átomos del metal. Un ejemplo del efecto de la humedad en una unión que presente cierto grado de fatiga es la delaminación. La humedad es absorbida produciendo una deformación de la soldadura que presiona el encapsulado hasta que se desprende.
- Los elementos utilizados en el material de soldadura incluyen metales como plata, oro, plomo, níquel, etc. Un **exceso del material intermetálico** de la soldadura, junto con el aumento de la temperatura, produce microfracturas que debilitan la unión.
- Operar a altas temperaturas o aumentar la densidad de corriente favorece la **electromigración** en la unión metal-substrato o metal-soldadura, causando la segregación de los elementos que la componen.
- En los encapsulados con terminales cableadas, las **tensiones mecánicas del cable** pueden ser fuente de fallos. Si la sección del cable es demasiado estrecho, una fuerte tensión lo puede fracturar. Pero si la tensión es demasiado débil, el cable se mueve y puede provocar cortocircuitos con cables adyacentes.
- Finalmente, los **defectos en el proceso de fabricación** son causa de cortocircuitos entre los terminales y uniones débiles e inconsistentes que se fracturan después de algún tiempo de operación.

3.1.2. Modelos de fallos

Los modelos de fallos deben definirse según el nivel al que se vaya a aplicar la técnica de inyección de fallos. Los niveles crecen desde eléctrico, lógico, almacenamiento, algorítmico, etc. hasta los niveles superiores de aplicación y operación (figura 3.5.) [DBench 2002]. En la inyección física de fallos los modelos se definen sobre los tres primeros niveles. A nivel RT resulta más comprensible la relación entre el fallo inyectado y cómo se modifica, a causa de ese fallo, la información transferida o almacenada.

En un estudio realizado bajo el proyecto Europeo “Dependability Benchmarking” [DBench 2002] se presentan dos tablas que relacionan las causas más frecuentes de fallos físicos con los modelos de fallos a nivel lógico y RT, una para fallos permanentes (figura 3.6.) y otra para transitorios (figura 3.7.).

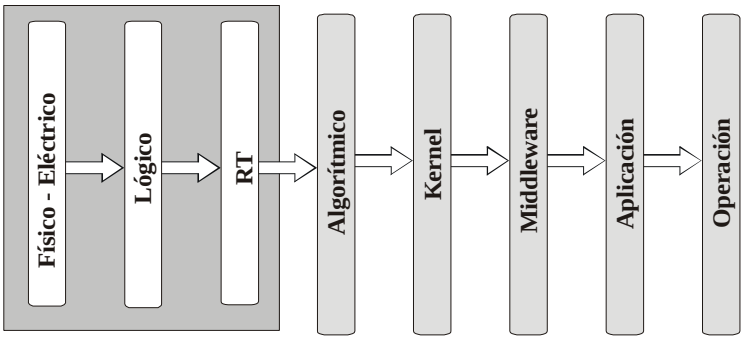


Figura 3.5. Niveles de descripción de los modelos de fallos

3.1.2.1. Fallos intermitentes y permanentes

La figura 3.6. [DBench 2002] muestra la relación entre causas de fallos físicos intermitentes y permanentes y los modelos de fallos aplicables a nivel lógico y RT.

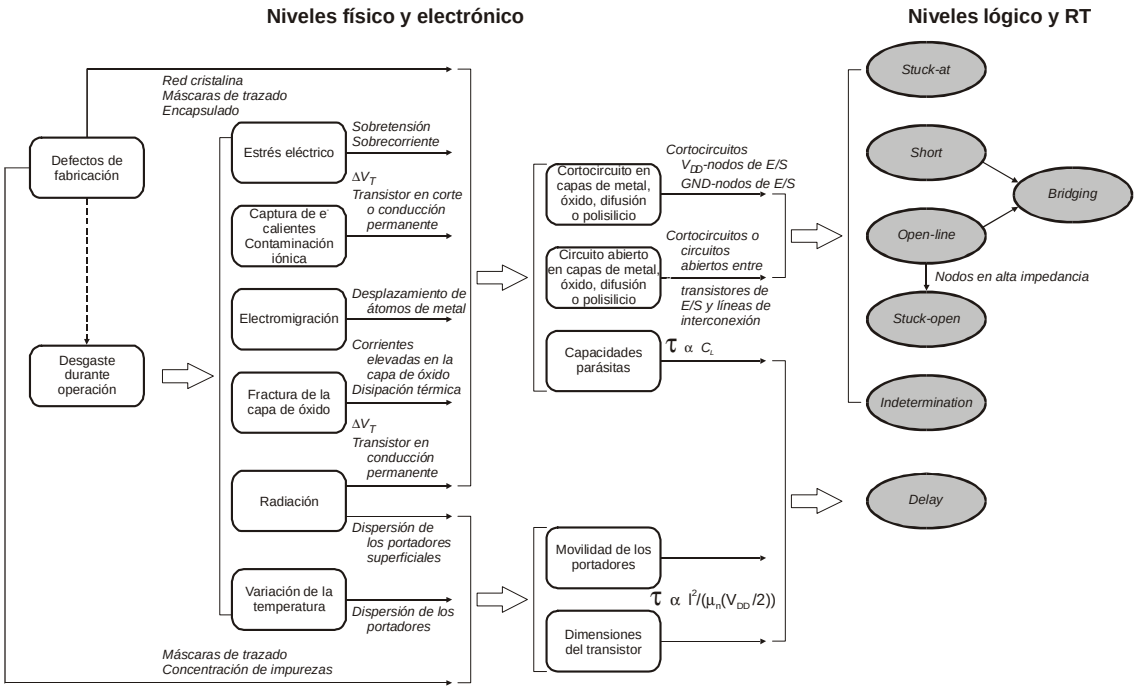


Figura 3.6. Causas de fallos físicos permanentes y modelos equivalentes

Los modelos que se contemplan son los siguientes:

- **Stuck-at** (**pegado de la línea**): En este modelo, la línea afectada por el fallo mantiene un valor constante sin posibilidad de cambio.
- **Short** (**puente entre dos líneas**): Se trata de un cortocircuito entre dos líneas supuestamente independientes.
- **Open-line** (**línea abierta**): Describe la desconexión física de la línea.
- **Bridging** (**puente**): Algunos autores consideran este modelo idéntico al "short". Otros (como en la figura 3.4.) consideran que se trata de un modelo que representa un puente entre dos líneas siempre que una se encuentre previamente abierta.

- **Stuck-open** (**pegado en nodos en alta impedancia**): Este fallo se debe a nodos flotantes en alta impedancia que mantienen el valor lógico previo del transistor durante un tiempo denominado tiempo de retención, que es el tiempo que tardan en descargarse las capacidades parásitas a causa de las corrientes de fuga.
- **Indetermination** (**indeterminación**): Indica que el valor de la línea no es seguro, pudiendo dar lecturas diferentes.
- **Delay** (**retardos**): Este modelo se debe a variaciones en la capacidad o resistencia de la línea, donde un cambio esperado se realiza con un retardo inesperado.

La deducción de los modelos de fallos de la figura 3.6. se centra principalmente en los efectos sobre los transistores que pueden favorecer la aparición de fallos físicos. Una propuesta para completar la relación entre causas de fallo y modelos sería añadir las causas de fallos en las metalizaciones y en las soldaduras, tal como muestra la figura 3.7.

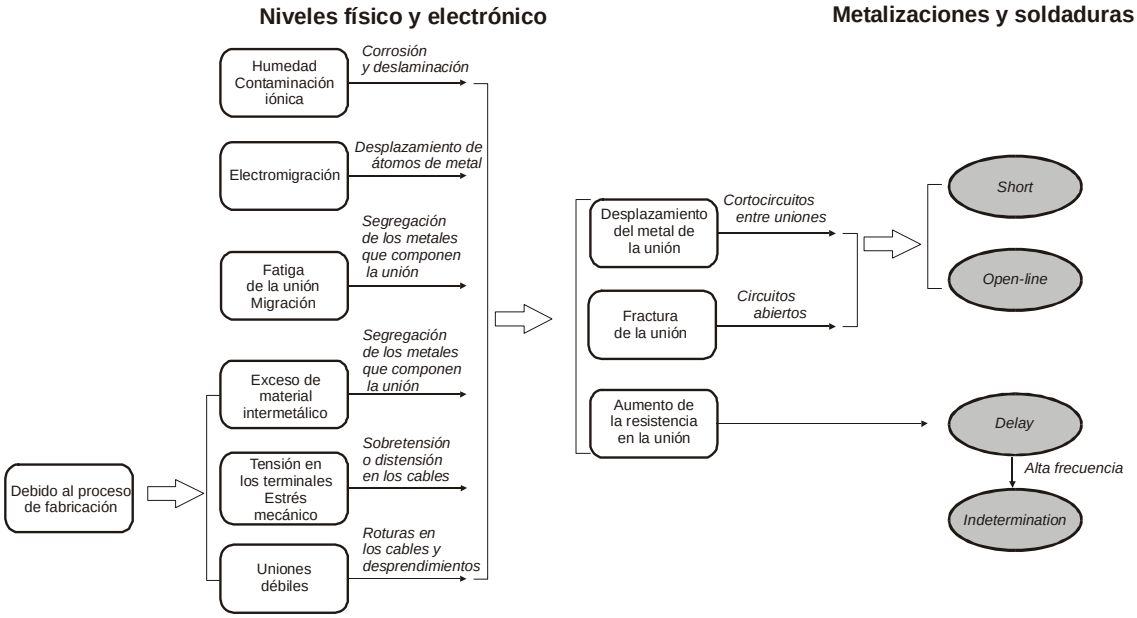


Figura 3.7. Causas de fallos físicos permanentes en metalizaciones y soldaduras

Cortocircuitos en uniones metálicas y en soldaduras: Con el objetivo de reducir el tamaño de los circuitos, se reduce la distancia entre las soldaduras incrementando el riesgo de cortocircuitos que origina el desplazamiento del metal en la unión. Las principales causas de este desplazamiento son el aumento de la densidad de corriente (electromigración) y la diferencia entre los coeficientes de expansión térmica a altas temperaturas entre aluminio-silicio (migración) o entre los elementos que componen el material de soldadura (fatiga). Tanto la migración como la fatiga son causa de microfracturas en la unión.

Circuito abierto en capas de metal o líneas abiertas en las soldaduras: Los diversos procesos a los que se someten los componentes integrados durante la fabricación pueda originar dispositivos defectuosos cuyas imperfecciones no se detectan por los propios procesos. El exceso de material intermetálico en la soldadura, las tensiones mecánicas en los cables, el estrés mecánico que sufren las capas de metal o incluso microfracturas derivadas del calentamiento de las soldaduras debilitan las uniones que pueden acabar rompiéndose.

Variación de la resistencia: Las fracturas internas que aparecen en una unión metálica, debida a la segregación de los metales que la componen, debilita la soldadura aumentando su resistencia ante la transmisión de la señal. El incremento de la resistencia en una línea produce un retardo no deseado en la señal transmitida.

Efecto de la transmisión a altas frecuencias: La debilitación de las soldaduras *aumenta la resistencia* de la componente RC en la transmisión. Por otro lado, al igual que se ha disminuido la distancia entre las pistas, la distancia entre los pines de entrada y salida es cada vez menor *incrementando el acoplamiento capacitivo* a altas frecuencias a nivel de pin. Teniendo en cuenta la corta duración de los pulsos en una transmisión a alta frecuencia, cualquier variación de la componente RC puede derivar en un pulso con un valor de tensión **indeterminado** al no alcanzar los márgenes asignados a los valores lógicos correspondientes.

3.1.2.2. Fallos transitorios

La figura 3.8. [DBench 2002] muestra la relación entre causas de fallos físicos transitorios y los modelos de fallos aplicables a nivel lógico y RT.

Los modelos que se contemplan son los siguientes:

- **Delay** (**retardos**): Es el mismo modelo que encontramos con fallos permanentes, pero ahora, de forma transitoria. Se debe a variaciones en la capacidad o resistencia de la línea, donde un cambio esperado se realiza con un retardo inesperado.
- **Bit-flip** (**inversión**): Se produce en celdas de memoria o en registros. Indica que el valor lógico almacenado en la celda se ha invertido.
- **Pulse** (**pulso**): Se trata de la observación de un pulso inesperado en la lógica combinacional.
- **Indetermination** (**indeterminación**): Debido a diferentes causas, el valor de la línea no es seguro, aunque en este caso el efecto del fallo es transitorio.

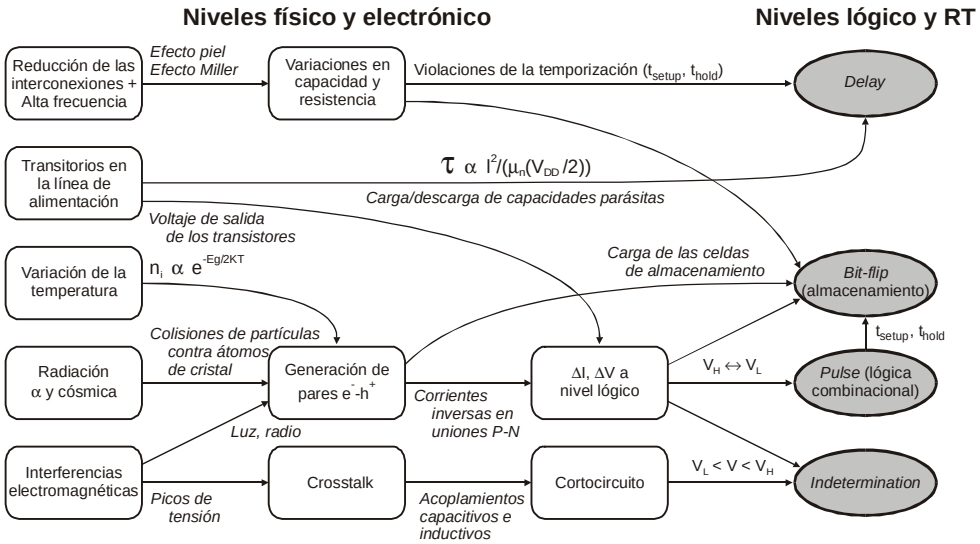


Figura 3.8. Causas de fallos físicos transitorios y modelos equivalentes

3.1.3. Fallos físicos y técnicas de inyección

Una vez se establece la traducción de los fallos físicos al contexto de la validación experimental, queda determinar cuál va a ser el proceso que permita, no sólo inyectar estos fallos de forma controlada en el sistema a validar, sino también, que resulte útil en el análisis de la confiabilidad del producto.

Así vemos que se plantean diferentes alternativas de inyección, las denominadas **técnicas de inyección de fallos**. Los recursos utilizados para el desarrollo del proceso de inyección son distintos, dando nombre a cada técnica. Una técnica, y en concreto una herramienta, será capaz de inyectar un conjunto de fallos, o al menos, de emular sus consecuencias a través de los modelos de fallos que las representan. Por tanto, la eficacia de un herramienta dependerá de la representatividad de los fallos inyectados, así como de su potencial para observar las consecuencias de estos fallos sobre un sistema determinado.

3.2. Descripción de la Inyección de Fallos

La inyección de un fallo en un sistema físico es una acción compleja, ya que siendo una experiencia controlada, requiere de una descripción formal del proceso. La inyección de fallos contempla dos dominios: el **dominio de entrada** y el **dominio de salida**. El *dominio de entrada* incluye aquellas acciones aplicadas sobre el sistema cuando se encuentra realizando un servicio, mientras que el *dominio de salida* hace referencia a la evolución del servicio tras la inyección del fallo y al estudio y evaluación del comportamiento observado.

Para precisar con más detalle las variables implicadas en ambos dominios es posible utilizar los denominados conjuntos **FARM,** propuestos inicialmente por J. Arlat [Arlat *et al*. 1993] y revisados en este capítulo, que corresponden a las siglas de **Fallos** y **Activación**, conjuntos pertenecientes al dominio de entrada, y **Resultados** y **Medidas**, pertenecientes al dominio de salida. Los parámetros que describen el fallo pertenecen al conjunto de *Fallos* del dominio de entrada, mientras que el conjunto de *Activación* define la selección del instante del servicio en el que se inyecta el fallo.

Aunque la precisión del dominio de entrada sea muy alta, si no existe la posibilidad de obtener variables representativas que muestren cuál ha sido la evolución del servicio tras la inyección, la experiencia estará incompleta. Estas variables pertenecen al conjunto de *Resultados*, mientras que el análisis realizado sobre ellas pertenece al conjunto de *Medidas*.

3.2.1. Fallos

Los fallos inyectados son representaciones de eventos reales que perturban o afectan al sistema. Si diferenciamos entre fallos de diseño e implementación y fallos físicos, ambos tienen parámetros descriptivos diferentes.

Los fallos de diseño e implementación se pueden dividir entre **fallos estructurales** y **fallos en el software**. Los *fallos estructurales* se deben a carencias o ambigüedades en las especificaciones, carencias o errores en la estructura del diseño o a una implementación incorrecta de las especificaciones. Los fallos estructurales no se inyectan sobre el sistema, sino que deben descubrirse como parte de la verificación, utilizando cargas de estrés como los tests de robustez [Koopman y DeVale 2000] o la propia inyección de fallos. En

[Tsai *et al*. 1996] se introduce el concepto de **inyección basada en el estrés**[36] que consiste en la ejecución de cargas sintéticas que acentúan la actividad en una parte determinada del sistema donde se concentran las inyecciones de fallos físicos. Por otra parte, los *fallos en el software* son fallos de programación, ya sea por carencias o errores en el código. Los fallos en el software se pueden emular mediante **técnicas de simulación**, o para sistemas más complejos, mediante técnicas de Inyección de fallos implementadas por software [Madeira *et al*. 2000]. La ODC ("Orthogonal Defect Classification" [Chillarege *et al*. 1992]) es el trabajo más preciso que se ha llevado a cabo para la clasificación de los posibles fallos que suelen aparecer en el software. Además de simulación y SWIFI, existe otra técnica para generar deliberadamente fallos en el software durante la compilación y ensamblado del código: **la mutación**, consistente en crear varias versiones de un mismo programa con diferencias sintácticas [DeMillo 1988, Daran y Thévenod-Fosse 1996].

Los parámetros utilizados para describir un fallo físico son la **localización del fallo**, el **modelo del fallo** y su **extensión temporal**.

Localización del fallo: Es el lugar donde se inyecta el fallo. Cada vez con más frecuencia, un fallo físico simple afecta a más de un componente básico. La frecuencia de operación de los sistemas, la densidad de integración y en general, la complejidad de los circuitos integrados hacen que ya no sea justificable el pensar que un fallo simple sólo genera un error simple. Los errores múltiples debidos a fallos físicos simples son, a partir de ahora, una realidad básica en la validación experimental.

Modelo del fallo: Descritos en el apartado 3.1.2. determinan en el entorno experimental el efecto de un fallo físico observado sobre un circuito integrado.

Extensión temporal del fallo: Se refiere a la duración o persistencia del fallo. Según su persistencia los fallos pueden clasificarse en permanentes, como en los fallos debidos a la degradación de los componentes; intermitentes, debido a componentes dañados; y transitorios, normalmente debido a causas externas con duración limitada.

3.2.2. Activación

El conjunto de activación incluye las entradas del sistema y el estado en el que se encuentra. Dentro de este conjunto se puede distinguir entre el **agente de activación** y el **agente de inyección**. El *agente de activación* es el encargado de generar el disparo que activa al *agente de inyección* encargado, a su vez, de inyectar los fallos en el sistema. Generar el disparo de inyección significa, por tanto, determinar el momento de la inyección, ya sea en función del tiempo transcurrido desde el inicio del experimento o dependiendo de la ocurrencia de alguna condición. En general, podemos diferenciar tres tipos de disparo de la inyección, el **disparo por tiempo**, el **disparo por condición** o un **disparo mixto** entre ambos.

En el *disparo por tiempo*, la inyección se activa dependiendo de parámetros temporales relativos a la ejecución de la carga en el sistema. Por ejemplo, que expire un tiempo aleatorio desde el inicio de una tarea. En el *disparo por condición*, la inyección se activa cuando la carga alcanza un estado o condición. Podemos diferenciar entre condiciones cuantitativas, donde se define el número de veces que debe ocurrir la

[36] Stress based injection

condición antes de activar al agente de inyección; o cualitativas, donde se define como activador una transición determinada de la máquina de estados asociada a la aplicación. Además, es posible combinar ambos disparos en un *disparo mixto*. Por ejemplo, esperar un tiempo aleatorio tras el cual se activará el agente de inyección dependiendo del cumplimiento de alguna condición.

3.2.3. Resultados

El conjunto de los resultados incluye todas las lecturas extraídas de los experimentos, ya sean lecturas referidas a la efectividad de la inyección o lecturas referidas al comportamiento del sistema tras la inyección de los fallos.

Las lecturas referidas a la efectividad de la inyección son aquellas que indican si el fallo ha sido inyectado correctamente provocando la alteración deseada. Por ejemplo, si un pegado a 0 sobre una línea de un bus ha modificado el valor de la línea, o si por el contrario, la línea ya estaba a 0. Para comprobar esta efectividad, algunas herramientas incorporan mecanismos adicionales que detectan la efectividad del fallo. En otras ocasiones se recurre a la comparación de la traza resultante tras la inyección con una libre de fallo, también llamada "**golden run**".

Las lecturas referidas al comportamiento del sistema incluyen, por un lado, la activación de los mecanismos deseados: detección del error, su aislamiento y tratamiento, así como la recuperación, en caso necesario, de la parte afectada. Por otra parte podemos obtener medidas relacionadas con los tiempos de latencia derivados de dichas acciones.

3.2.4. Medidas

Finalmente, el conjunto de medidas se obtienen a partir de las lecturas extraídas durante los experimentos. Estas medidas dependen del objetivo de la validación. En el caso de que el objetivo sea la ***eliminación de fallos***, se trata de identificar debilidades del sistema y sintonizar los mecanismos de tolerancia a fallos. Existen medidas tales como, por ejemplo, la **sensibilidad** de los mecanismos de detección [Steiniger y Scherrer 1997] que mide el porcentaje de utilización de cada mecanismo de tolerancia a fallos implementado para la detección de un tipo de error determinado. La finalidad de esta medida es cuantificar el beneficio obtenido de la implementación de cada mecanismo de tolerancia a fallos. En caso de que el objetivo sea la ***predicción de fallos***, existen otras medidas como, por ejemplo, el factor de cobertura del sistema, que indica la probabilidad de que el sistema actúe correctamente ante un error, o el factor de cobertura de un mecanismo de tolerancia a fallos implementado, para obtener la probabilidad de que este mecanismo detecte un tipo de error determinado con un tiempo de latencia inferior al máximo especificado.

3.3. Técnicas de Inyección de Fallos

Existen diferentes técnicas que permiten generar un dominio de entrada y salida conveniente para llevar a cabo la validación de un sistema. La forma de acceder al sistema permite diferenciar entre tres técnicas generales, la **Inyección de Fallos Física o implementada por Hardware HWIFI**, la **Inyección de Fallos implementada por Software SWIFI** y la **Inyección de Fallos basada en técnicas de Simulación de modelos**. Tanto la *Inyección de fallos física o implementada por hardware*, como la *Inyección de fallos implementada por software*, son de aplicación sobre un prototipo del sistema real, mientras que las técnicas de *Inyección de fallos basadas en simulación de modelos* no necesitan la construcción del sistema, y por tanto, son muy prácticas durante

las primeras etapas de la fase de diseño. Existe una diferencia básica entre las técnicas *HWIFI* y las *SWIFI*. En HWIFI los fallos se inyectan sobre los componentes físicos del sistema, utilizando algún tipo de instrumentación. Por el contrario, en SWIFI se lleva a cabo una emulación de los fallos físicos mediante la generación directa de los errores, consecuencia de dichos fallos, a nivel RTL.

Cada una de las técnicas generales de inyección de fallos: *Inyección de fallos basada en simulación de modelos, Inyección de fallos implementada por hardware* e *Inyección de fallos implementada por software*, da título a otras técnicas, en un segundo nivel, que precisan con mayor detalle el modo de llevar a cabo la inyección del fallo (figura 3.9.).

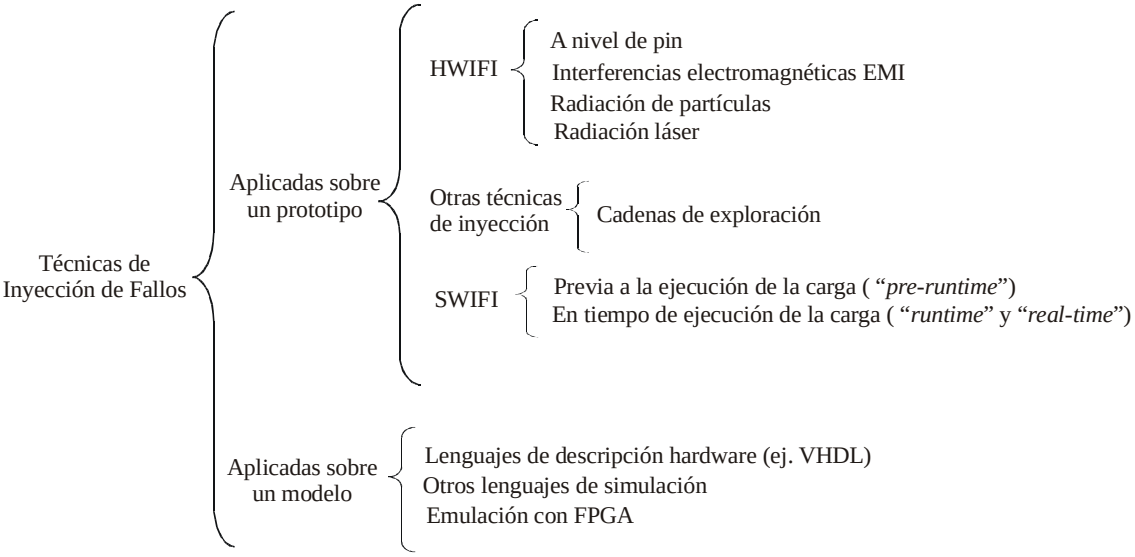


Figura 3.9. Desglose de las técnicas de inyección de fallos

La forma de acceder al sistema para inyectar los fallos hace que cada técnica en particular tenga una serie de propiedades que las diferencian entre sí. Estas propiedades valoran el entorno de inyección y pueden ser orientativas para decidir cuál sería la técnica o técnicas más apropiadas para llevar a cabo unos experimentos determinados. Dos técnicas del segundo nivel, aún agrupadas bajo el mismo epígrafe, no son necesariamente iguales ya que cada una puede tener una medida distinta de sus propiedades.

3.3.1. Propiedades de las técnicas de inyección de fallos

Cada propiedad presentada describe un aspecto concreto de las técnicas de inyección. No se trata de conceptos totalmente independientes, sino que existe relación entre ellas. Las propiedades son:

- **Nivel de abstracción**: La inyección de fallos se puede aplicar tras la especificación de la función del sistema, desde las primeras etapas de desarrollo y a lo largo de todas las fases de su ciclo de vida. Por tanto, el nivel de abstracción implica el momento del desarrollo en el que se va a aplicar la técnica de validación, ya sea sobre la descripción estructural o comportamental de un modelo del sistema, sobre un prototipo o sobre el sistema final.

- **Naturaleza de los fallos**: Indica si los fallos físicos se simulan en un

modelo, se inyectan a través de algún tipo de instrumentación o se emulan. El primer caso hace referencia a la inyección de fallos basada en técnicas de simulación de modelos, el segundo caso hace referencia a las técnicas implementadas por hardware y por último, la emulación de fallos físicos es característica de las técnicas implementadas por software.

- **Representatividad de los fallos**: Es una de las propiedades más interesantes y a la vez, más difíciles de determinar. Se refiere a la habilidad de inyectar fallos físicos sobre el sistema que sean representativos de los fallos reales (o de emular correctamente sus consecuencias), siendo fallos reales aquellos que pueden ser observables durante la fase operacional del sistema. En esta propiedad también se incluye la emulación de *fallos en el software*, dentro de los fallos de diseño antes descritos y los fallos operacionales de origen humano.

- **Accesibilidad**: Indica los componentes o las unidades funcionales donde una técnica es capaz de acceder para inyectar el fallo. La localización es el primer parámetro utilizado para describir un fallo físico a inyectar y de su precisión dependen otras propiedades como, por ejemplo, la reproducibilidad de los experimentos.

- **Control espacial**: Esta propiedad está relacionada con la accesibilidad, evaluando la capacidad de precisar la localización de los fallos, así como los parámetros relativos al modelo y persistencia del fallo. Cuanto más cercano sea el nivel de descripción al físico-eléctrico (figura 3.5.) mejor *control espacial* se considera. Esta propiedad caracteriza, por tanto, al *agente de inyección*.

- **Control temporal**: Es la capacidad de identificar el estado del servicio en el momento de la inyección. Esta propiedad caracteriza al *agente de activación*. Algunas técnicas mejoran su *control temporal* al ser capaces de seleccionar en qué estado del servicio se desea forzar la inyección del fallo.

- **Multiplicidad**: Un fallo no siempre es simple, podemos encontrar fallos múltiples conexos con la misma persistencia pero en localizaciones diferentes, aunque con algún tipo de relación. Por tanto, definimos la *multiplicidad* como la capacidad de generar tanto fallos simples como fallos múltiples.

- **Reproducibilidad de los experimentos**: Se define como la capacidad de reproducir un determinado experimento de inyección. La reproducción puede ser **determinista** o **probabilística**. Un experimento se considera *determinista* cuando es posible reproducir el mismo dominio de entrada *n* veces obteniendo siempre la misma respuesta del sistema y por tanto, un único dominio de salida. Cuando el dominio de entrada sólo se pueda reproducir de forma aproximada, existiendo ciertas diferencias entre dos experimentos cualesquiera, la respuesta del sistema varía de un experimento a otro. En este caso es conveniente trabajar con una muestra de experimentos con la cual valorar la respuesta del sistema en función de un conjunto de medidas. La reproducción *probabilística* consiste en generar varias muestras, cada una con el suficiente número de experimentos como para reproducir, con un alto grado de confianza, los valores de las medidas analizadas.

- **Duración de los experimentos**: Esta propiedad adquiere mayor importancia a medida que aumenta el número de experimentos necesarios para llevar a cabo la validación del sistema. La complejidad de los sistemas actuales incide negativamente sobre algunas técnicas de inyección, ya que el tiempo necesario para realizar un conjunto de muestras, sobre cuyos resultados se pueda depositar cierto grado de confianza, no es razonable. Esto puede conducir al empleo de una infraestructura más costosa con el fin de reducir la duración de los experimentos.

- **Observabilidad**: Se define como la capacidad de obtener lecturas válidas sobre el comportamiento del sistema en presencia de fallos. También puede definirse como la capacidad de observar la evolución de un servicio tras la inyección del fallo. Por tanto, el conjunto de *Resultados* depende de esta propiedad.

- **Intrusión**: Evalúa el grado de intrusión no deseada que una instrumentación ajena al sistema pueda ocasionar sobre éste durante el proceso de validación.

- **Sobrecarga**: Cuantifica la sobrecarga en tiempo de ejecución y requisitos de memoria del sistema que la técnica pueda ocasionar.

- **Medida de tiempos**: Se refiere a la capacidad de obtener los tiempos de latencia de los diferentes mecanismos tolerantes a fallos implementados para detectar errores. Con frecuencia, las *medidas de tiempos* en sistemas distribuidos observan si la reacción de dichos mecanismos es suficientemente rápida como para evitar la propagación del error.

- **Riesgo**: Evalúa si la técnica supone algún peligro para las personas o si puede provocar daños irreparables en el sistema validado.

- **Reutilización**: Es una propiedad específica de la herramienta de inyección que describe si la herramienta es genérica, por tanto se puede utilizar la misma herramienta sobre diferentes sistemas, o es específica, siendo útil sólo para un sistema concreto. Esta propiedad también cuantifica las dificultades de adaptación de una herramienta genérica sobre un nuevo sistema a validar.

- **Potencial de automatización**: Es una propiedad específica de cada herramienta de inyección. Se define como la capacidad de una herramienta de inyección a realizar un conjunto de experiencias de forma autónoma, o si por el contrario, necesita de la intervención constante de un supervisor humano.

- **Coste de desarrollo**: Evalúa el coste económico, así como las capacidades humanas y técnicas necesarias para el desarrollo de una herramienta de inyección basada en una técnica determinada.

- **Coste de la infraestructura**: Evalúa el coste de la infraestructura necesaria para aplicar una herramienta de inyección sobre un sistema.

3.3.2. Inyección física de fallos o implementada por hardware HWIFI

La inyección de fallos físicos puede ser interna o externa al componente. Se consideran como técnicas de inyección externa: la **inyección de fallos a nivel de pin** y las **interferencias electromagnéticas**. Son técnicas de inyección interna: la **radiación de partículas** y la **radiación mediante láser**, para las que es necesario rebanar el encapsulado del componente.

3.3.2.1. Inyección física de fallos a nivel de pin

En la inyección física de fallos a nivel de pin, el fallo se localiza en los terminales o patillas de los circuitos integrados, con el objetivo de alterar los valores lógicos de las señales de entrada y/o salida del componente [Arlat *et al*. 1990, Gil 1992, Madeira *et al*. 1994, Gil *et al*. 1997, Martínez *et al*. 1999].

Existen dos posibles implementaciones de la técnica, **inyección por forzado** e **inyección por inserción** (figura 3.10). En la *inyección por forzado* el fallo se inyecta directamente sobre el terminal o patilla del componente sin ningún tipo de desconexión. La sonda de inyección fuerza el terminal a un valor lógico determinado, independientemente de cual sea su valor real. En la *inyección por inserción* se requiere de un dispositivo adicional que reemplaza uno o más componentes. Este dispositivo se inserta en el lugar del componente original, con dos modos de funcionamiento. Primero, el comportamiento del dispositivo es idéntico al componente original (funcionamiento libre de fallos). Segundo, se introduce un valor lógico determinado en la salida del componente reemplazado, independientemente de cual fuera el valor esperado (inyección del fallo). Esta técnica asegura la no destrucción o daño del componente. Sin embargo, es de difícil implementación en circuitos integrados con soldaduras de montaje superficial.

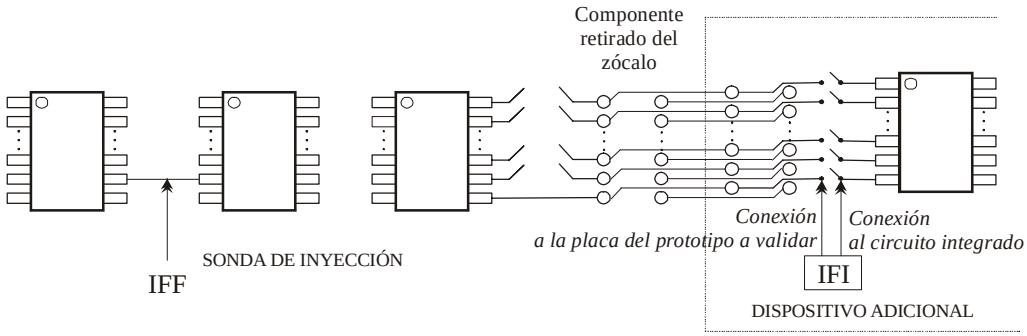


Figura 3.10. Puntas de inyección por forzado (IFF) e inserción (IFI)

Entre las ***ventajas*** que aporta la *inyección de fallos físicos a nivel de pin* podemos considerar que la utilización de prototipos físicos, muy próximos al sistema final, genera unos resultados más representativos que con otras técnicas de simulación o emulación de fallos. Esta ventaja la encontraremos en muchas de las técnicas HWIFI, al igual que una sobrecarga nula inducida sobre el sistema, ya que son herramientas que utilizan una instrumentación externa que no necesita alterar el código de ejecución. Dicha instrumentación permite generar disparos por tiempo y por condición, llevando a cabo la inyección sin provocar retardos o paradas temporales de la ejecución, requisito imprescindible para la validación de sistemas distribuidos y sistemas de tiempo real. Además, es posible mantener un control de la efectividad del fallo que determine si este ha producido la alteración deseada, evitando la necesidad de comparar la traza del servicio en presencia de fallos con otra traza libre de fallos (una "*golden run*").

También se puede destacar la baja duración de los experimentos, su reproducibilidad y el potencial de automatización. El control espacial es alto, sin embargo, la accesibilidad se restringe a los pines de los componentes. Permite multiplicidad de fallos y es destacable la fácil obtención de tiempos de latencia en la detección de los errores.

Entre los ***inconvenientes*** cabe destacar la restricción de su accesibilidad a las líneas de entrada, salida y alimentación del componente. El problema aumenta con el grado de integración de los componentes de nueva generación. Este mismo inconveniente es aplicable a la observabilidad del servicio del sistema. También disminuye la distancia entre las patillas de los integrados aumentando las dificultades físicas en la conexión de las herramientas al componente, especialmente sobre los de montaje superficial en los que no se puede aplicar la inyección por inserción. Existe cierta probabilidad de daño sobre los componentes al utilizar la técnica de forzado. Aunque esta probabilidad es muy baja, ha de tenerse en cuenta.

El coste y esfuerzo de desarrollo de este tipo de herramientas no es despreciable. La inversión económica para construir una herramienta con suficientes capacidades puede ser alta, por lo que la tendencia es construir herramientas genéricas que puedan ser reutilizables en la validación de diferentes sistemas físicos.

Entre las herramientas más importantes implementadas bajo esta técnica cabe destacar **AFIT** ("Advanced Fault Injection Tool"), de la Universidad Politécnica de Valencia [Gil 1992, Gil *et al*. 1997, Martínez *et al*. 1999, Blanc *et al*. 2001, Blanc *et al*. 2002 (a), Blanc *et al*. 2002 (b), Blanc y Gil 2003]. Es una herramienta genérica que implementa la inyección por forzado. Es capaz de inyectar fallos tanto permanentes como intermitentes y transitorios a una frecuencia de inyección de 40MHz. La herramienta sigue estando activa en la actualidad.

Otras herramientas son:

- **FTMP**, presentada por J. Lala en [Lala 1983], fue la primera herramienta publicada que utilizaba esta técnica implementada mediante inyección por inserción.

- **MESSALINE**, del Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS) [Arlat *et al*. 1990], es una herramienta genérica que implementa tanto el forzado como la inserción. Sobre la persistencia del fallo, es capaz de generar fallos transitorios a una frecuencia máxima de inyección de 10MHz.

- **RIFLE**, de la Universidad de Coimbra (Portugal) [Madeira *et al*. 1994], es una herramienta genérica que implementa la técnica de inserción. Su frecuencia máxima de inyección es de 4MHz.

- A. Steiniger y C. Scherrer presentan en [Steiniger y Scherrer 1997] una herramienta, sin bautizar, que implementa tanto la inyección por forzado como por inserción. Se probó en la validación del sistema SCRIBO, basado en un microcontrolador de Motorola 88000 con caché externa MMU para datos e instrucciones.

- **VIRUS** ("Versatile Fault Injector for Reproducible Single Injections"), de la Universidad de Viena [Lettner *et al*. 1998], se implementó para validar, mediante inyección por forzado, uno de los primeros prototipos

del sistema TTP^{TM}/C, único sistema con el que se ha probado.

- C. Constantinescu presenta en [Constantinescu 1998] una herramienta para validar el supercomputador Teraflop de gran potencia, con 9.260 procesadores Intel Pentium Pro a 200MHz y 573 terabytes de memoria. No consta que la herramienta fuera genérica y que se haya vuelto a utilizar.

- También existe una herramienta comercial, **DVT-100** [Proteus 1996, Stewart 1997], comercializada por *Proteus Corporation*. Se trata de una herramienta automatizada, dotada de sondas de inyección y monitorización robotizadas.

- De la Universidad de Chalmers, Suecia, existen varios trabajos publicados [Damm 1986, Gunneflo 1990, Miremadi *et al*. 1992] sobre un tipo de herramienta de inyección específica para la perturbación de las líneas de alimentación. Esta técnica consiste en provocar fluctuaciones de tensión en la entrada de alimentación.

3.3.2.2. Interferencias electromagnéticas EMI

El ejemplo más destacable de esta técnica es el trabajo de la Universidad de Viena (Austria) divulgado en [Leber 1993 y Karlsson *et al*. 1995]. Es este trabajo utilizan un generador comercial de ráfagas de alta tensión según el estándar EN 61000-4-4, con las siguientes características: duración de 15ms, periodo de 300ms, frecuencias entre 1,25 KHz, 2,5 KHz, 5 KHz o 10KHz y voltajes entre 225V hasta 4400V. Es necesario quitar la caja o protector que típicamente se utiliza para aislar al circuito sobre el que se va a inyectar del efecto de las interferencias electromagnéticas. El ruido se induce en los componentes a través de las soldaduras. Desarrollaron dos tipos de sonda, una para generar interferencias en cualquier componente del prototipo (izquierda de la figura 3.11.) y otra para concentrar las interferencias en un solo componente (derecha figura 3.11.).

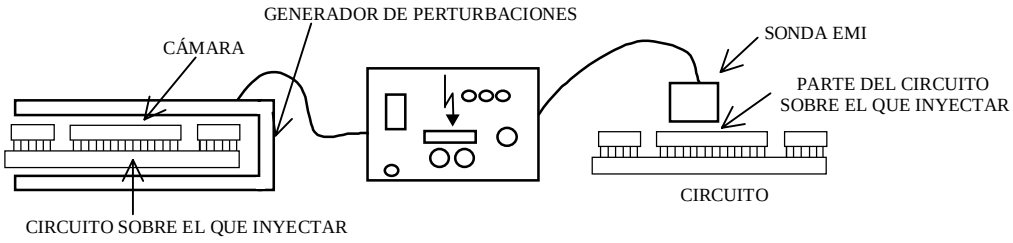


Figura 3.11. Experimento de inyección mediante interferencias electromagnéticas

La principal ***ventaja***, con respecto a la representatividad de los fallos, es la aplicación de la técnica sobre un prototipo físico próximo al sistema final. La sobrecarga inducida es nula y la duración de los experimentos no es muy elevada. Sin embargo, no existe control de la efectividad del fallo, siendo necesario comparar la traza del servicio obtenida con otra libre de fallos. Tiene un alto potencial de automatización y la dificultad de implantación es mínima ya que no existe conexión ninguna con el prototipo.

Entre los ***inconvenientes*** destacamos el bajo control espacial y temporal que presenta este tipo de inyección. Por tanto, los experimentos son difíciles de reproducir, tan solo cabe la posibilidad de una reproducción probabilistica. La observabilidad tampoco es muy alta, depende del grado de integración de los componentes y de la

orientación del diseño[37]. Las interferencias pueden causar fallos múltiples difíciles de detectar, por ejemplo, a través de las líneas de alimentación. También existe un riesgo, aunque bajo, de dañar los componentes. Finalmente, el coste de infraestructura es alto, ya que incluye el generador comercial y la fabricación de las sondas.

3.3.2.3. Inyección mediante radiación de partículas

Los efectos de la radiación pueden causar alteraciones no deseadas en las propiedades eléctricas de los semiconductores y por tanto, se producen operaciones incorrectas en estos sistemas [LaBel 2000].

Las herramientas de inyección de fallos por radiación tratan de generar efectos del tipo SEE, especialmente los del tipo SEU[38], ya que son los más frecuentes, mediante la utilización de **fuentes radiactivas** y **aceleradores de partículas**.

Fuentes radiactivas: Algunos ejemplos de isótopos emisores de partículas alfa utilizados para observar el efecto de la radiación sobre los semiconductores son el Americio, el Torio y el Californio. El ^{241}Am tiene un efecto muy parecido al ^{210}Po, isótopo que se encuentra en el material de soldadura. La ventaja del Americio es su legalidad y que no emite gases perjudiciales para la salud. El ^{228}Th también se ha utilizado en algunos experimentos, sin embargo es el ^{252}Cf el isótopo radioactivo que proporciona niveles energéticos más parecidos a los derivados por los isótopos naturales que forman parte de las impurezas de los semiconductores. Las fuentes de ^{252}Cf aplicadas sobre circuitos integrados, a los que previamente se les ha desprotegido del encapsulado, se han utilizado en diferentes ocasiones para provocar SEU en los semiconductores.

Aceleradores de partículas: Las herramientas de inyección con fuentes radiactivas tienen una precisión muy baja sobre el control de la inyección. Por ejemplo, aunque la intención de aplicar esta técnica sea la de generar fallos transitorios, la probabilidad de generar fallos permanentes o daños irreversibles en el semiconductor no es despreciable. Además, la radiación es continua, sin poder determinar el momento de inyección, la localización de la zona afectada, ni la cantidad de partículas emitidas. Esta falta de control se resuelve con los aceleradores de partículas de tipo ciclotrón. Actualmente, este tipo de aceleradores ofrecen suficientes facilidades para realizar inyecciones con un alto nivel de precisión y un buen diagnóstico (observación) como, por ejemplo, el que se presenta en [Virtanen 2002].

Las ***ventajas*** e ***inconvenientes*** de la inyección de fallos mediante radiación de partículas son similares a las descritas en interferencias electromagnéticas, aunque se mejora la accesibilidad. Sin embargo, son de difícil aplicación y existe un alto riesgo de generar fallos permanentes en el componente.

Los trabajos presentados en [Gunneflo *et al*. 1989] y [Karlsson *et al*. 1994] son ejemplos de aplicación de fuentes radiactivas. En ambos casos utilizan una herramienta llamada **FIST** ("Fault Injection System for Study of Transient Fault Effects"), capaz de generar fallos transitorios mediante una fuente de ^{252}Cf bajo una cámara de aislamiento, ya que el aire contiene partículas que pueden frenar las partículas cargadas emitidas por la fuente.

[37] La orientación del diseño hace referencia a las facilidades incluidas en el diseño del circuito para posibilitar la observación del estado interno de un componente con una herramienta externa.
[38] Apartado 3.1.1.1.

Otro ejemplo realizado con fuentes radiactivas es el trabajo presentado en [Miremadi *et al*. 1992], donde se utiliza una fuente radiactiva para evaluar y comparar la efectividad de dos mecanismos de detección de errores en el flujo de ejecución (llamados "Block Signature Selt Checking" y "Error Capturing Instruction").

El trabajo presentado en [Karlsson *et al*. 1995] es una comparación de los resultados obtenidos en la validación del sistema MARS (antecedente de TTP^{TM}/C) mediante radiación con una fuente de ^{252}Cf, la inyección de fallos a nivel de pin, las interferencias electromagnéticas EMI y también con una herramienta específica SWIFI. El trabajo concluye con la compatibilidad de los resultados, ya que debido a las diferencias en las propiedades de cada técnica, especialmente en lo referente a accesibilidad, los resultados no se solapan sino que se complementan.

También destacamos la utilización de fuentes radiactivas durante la validación de los prototipos TTP/C-C1 y C2 de *Austrian Micro Systemes* [Sivencrona *et al*. 2003 (a)].

En [Koga *et al*. 1990] se utiliza un ciclotrón de protones para determinar la vulnerabilidad de los circuitos lógicos ante SEU. Utilizaron un rango bastante amplio desde circuitos muy simples hasta microprocesadores. Se observaron SEL ("latch-ups") que pueden dañar permanentemente el circuito debido al exceso de disipación de calor.

En [Gaisler 1997] se muestran los resultados de validar el procesador de 32 bits ERC32 SPARC utilizando un acelerador de partículas. En este estudio se muestra que la unidad de coma flotante es la más sensible ante SEU ya que es en la que más errores sin detectar se observan. Los resultados también mostraron que un código de paridad para las instrucciones decodificadas en la CPU es suficiente para detectar errores generados por SEU. El mismo autor valida en [Gaisler 2002] una nueva versión, el LEON-FT.

3.3.2.4. Inyección mediante radiación láser

Las primeras aproximaciones se presentan en [Velazco *et al*. 1990 y Martinet 1992]. El láser se utiliza para cortar pistas internas y conexiones metalizadas de los circuitos integrados, generando fallos permanentes.

Actualmente, está técnica permite la inyección de fallos, con un control espacial muy alto, dentro de los circuitos integrados. Se realiza mediante un puntero láser, basándose en el principio de que la energía de un láser genera portadores (pares e^--h^+) en el substrato de silicio. Se trata, por tanto, de un efecto no destructivo muy similar al SEU.

El ejemplo más destacable es el de [Sampson *et al*. 1998 y Moreno *et al*. 1999] donde se presentan los experimentos realizados con una herramienta semiautomática de radiación láser (figura 3.12.) con la que se pueden provocar errores reversibles[39] con una precisión muy alta tanto espacial como temporal. Esta última se consigue sincronizando la herramienta con la ejecución de la aplicación. La localización del fallo se determina con una herramienta CAD donde se visualiza todo el circuito y con la que se puede seleccionar el punto de inyección. Dicho punto, se traslada a coordenadas X-Y de la tabla del láser. Por último, se dispara el láser, produciendo un pulso corto de suficiente potencia para inducir un fallo transitorio.

[39] Soft errors

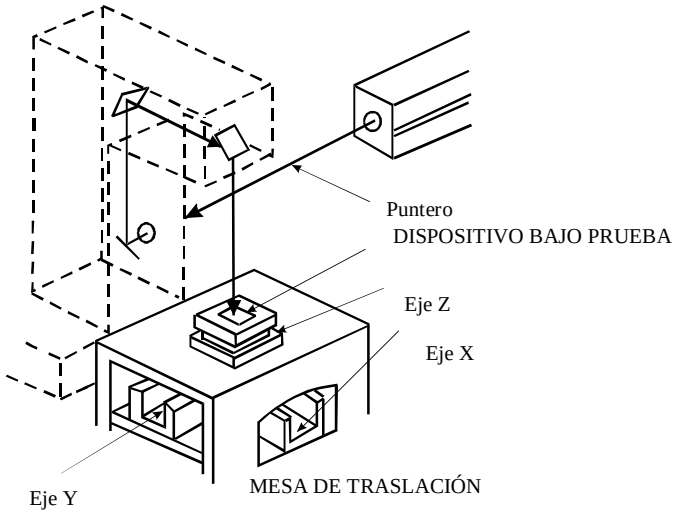


Figura 3.12. Experimento de inyección mediante radiación láser.

En [Darracq *et al*. 2002] se utiliza esta técnica para comprobar el efecto de los SEE sobre las celdas de memoria SRAM. Utilizan una herramienta presentada en [Pouget *et al*. 1998], capaz de generar pulsos de picosegundos y con margen de error sobre el objetivo de 0,1μm. Con esta herramienta se pueden generar pulsos simples, o repetirlos con una frecuencia de hasta 80MHz.

Al igual que las técnicas HWIFI anteriores, la inyección mediante láser trabaja con el prototipo físico próximo al sistema final. Entre sus ***ventajas*** destacan la sobrecarga nula inducida sobre el sistema y la baja duración de los experimentos. Tiene gran potencial de automatización, la accesibilidad es alta, así como el control espacial y temporal.

Como ***inconvenientes***, mantiene una observabilidad no muy alta, dependiendo del grado de integración de los componentes y de la orientación del diseño[7]. El coste de infraestructura y de construcción del entorno de desarrollo es muy alto.

3.3.3. Otras técnicas de inyección

3.3.3.1. Inyección mediante cadenas de exploración

Se puede considerar una técnica de inyección física, ya que emplea mecanismos hardware disponibles dentro del propio integrado. Sin embargo, desde el punto de vista del tipo de fallos inyectados, se trata de una emulación de los fallos físicos idéntica a las técnicas SWIFI.

Precisa de circuitería adicional en los circuitos VLSI ("Very Large Scale Integration"). Hasta ahora, la técnica más interesante es la llevada a cabo en la Universidad de Chalmers (Suecia) [Folkesson *et al*. 1998]. La herramienta FIMBUL ("Fault Injection and Monitoring using BUilt in Logic") está basada en la utilización del Boundary Scan (BSCAN) adoptado por IEEE a principios de los años 90 para la realización de tests funcionales sobre circuitos VLSI. A través del TAP ("Test Access Port"), como el JTAG del estándar Boundary Scan, es capaz de modificar el valor de los registros internos del procesador para emular distintos tipos de fallos.

GOOFI ("Generic Object-Oriented Fault Injector"), presentada por primera vez en

[Aidemark *et al*. 2001], es otra herramienta de este tipo adaptable a varios sistemas. Se trata de un software para la definición de cadenas de inyección implementado en Java y SQL.

Como ***ventajas*** destacan su buena accesibilidad, que permite inyectar fallos en profundidad, aunque depende del número de registros conectados a través del puerto. Es posible controlar la efectividad de la inyección. Por tanto, no necesita comparar la traza del servicio con otra libre de fallos. El control espacial es bueno y no existe posibilidad de dañar el prototipo.

El mayor ***inconveniente*** es que no se puede inyectar el fallo sin detener la aplicación. Es una gran desventaja en la validación de sistemas distribuidos de tiempo real. Además, generalmente los experimentos requieren demasiado tiempo.

3.3.4. Inyección de fallos implementada por software SWIFI

El objetivo de la Inyección de fallos implementada por software es el de reproducir, a nivel lógico, errores que pueden producirse tras la ocurrencia de un fallo físico. Existe un elevado número de localizaciones accesibles mediante software. Sin embargo, no todas lo son, como por ejemplo, registros internos ocultos, códigos de paridad en memoria, buses, líneas de control, etc. Generalmente, una herramienta de inyección de fallos implementada por software se integra como parte de la carga del sistema. Por tanto, su mayor crítica es la sobrecarga que producen. Los modelos de fallo utilizados son la inversión y el pegado a un valor lógico determinado. Debido a su bajo coste económico, existe un gran número de herramientas desarrolladas bajo esta técnica. Pero hay que destacar que no todas las herramientas son genéricas, sino desarrolladas específicamente para un sistema concreto.

Las herramientas SWIFI se implementan sobre el prototipo físico del sistema a validar, pero a diferencia con las técnicas HWIFI, los fallos físicos son emulados a través de errores consecuencia de dichos fallos. Entre las ***ventajas*** de la inyección SWIFI destacamos la alta accesibilidad de las herramientas, aunque restringida a bloques funcionales alcanzables por software. Mejoran la observabilidad en comparación con las técnicas HWIFI. En general, presentan un control espacial y temporal alto. Por tanto, la reproducción de los experimentos es muy buena, así como la observación de los tiempos de latencia.

Pero la ventaja que ha prevalecido sobre las anteriores, cuya consecuencia es el alto número de herramientas de este tipo desarrolladas hasta el momento, es su bajo coste económico y de infraestructura que compensa la dificultad de desarrollar herramientas genéricas.

Entre los ***inconvenientes***, citar que los errores se generan entre los niveles RTL (sólo registros accesibles por software) y de aplicación, no pudiéndose alcanzar niveles más bajos como el nivel lógico o de transistor.

3.3.4.1. División práctica de las herramientas SWIFI

Las técnicas de inyección de fallos implementadas por software se pueden dividir a priori en dos grupos, técnicas que inyectan los fallos previamente a la ejecución de la carga (“**pre-runtime**”) o en tiempo de ejecución, diferenciando entre “**runtime**” y “**real-time**”.

SWIFI previa a la ejecución de la carga “pre-runtime”: Son técnicas que se basan

en la modificación (introducción de errores) de la carga del sistema antes de que comenzar el servicio. Normalmente se trata de errores en memoria de código. No necesita implementar carga adicional, como pudieran ser agentes de activación e inyección, siendo la sobrecarga inducida nula. Es bastante efectiva para emular fallos físicos permanentes o fallos en el software. Sin embargo, con esta técnica no se pueden emular los efectos de los fallos físicos transitorios.

SWIFI en tiempo de ejecución "runtime": Con esta técnica, la carga comienza a ejecutarse libre de errores. Durante la ejecución, el agente de activación dispara la inyección según alguna condición preestablecida. La inyección se lleva a cabo por el agente de inyección. Ambos agentes se implementan junto con la carga normal del sistema, por lo que sí pueden suponer una sobrecarga adicional. Sin embargo, depende de la habilidad y saber hacer del programador, así como de su conocimiento de la arquitectura del sistema, conseguir que esta sobrecarga sea mínima.

Tras el estudio de las herramientas SWIFI "runtime" publicadas hasta el momento, se plantea su división en tres grupos atendiendo al diseño del agente de activación: mediante **temporizadores**, **interrupciones condicionales** e **inserción de nuevo código**.

Temporizadores: Se encuentran en herramientas orientadas a la validación de la función de los controladores o procesadores pertenecientes al sistema. Aprovechan la utilidad de los periféricos internos del controlador, como es el caso de los temporizadores, para la programación de tiempos aleatorios que determinen el instante de inyección. La programación típica de estas herramientas utiliza la rutina de tratamiento de la interrupción asignada al temporizador como agente de inyección.

Algunos ejemplos de herramientas diseñadas con un agente de activación del tipo *temporizadores* son FERRARI [Kanawati *et al*. 1992, Kanawati *et al*. 1995], DEFINE [Kao e Iyer 1994], SFI [Rosemberg y Shin 1993], DOCTOR [Han *et al*. 1995], Xception [Carreira *et al*. 1998], SOFI [Campelo 1999] y FIMD-MPI [Blough y Liu 2000].

Interrupciones condicionales: Se genera una interrupción ante la observación de una condición o evento en la decodificación del código. Esta condición puede ser desde la decodificación de una instrucción concreta, la decodificación de dicha instrucción un número delimitado de veces, una decodificación enmascarada, o la decodificación de una marca previamente insertada.

Algunos ejemplos de herramientas diseñadas con un agente de activación del tipo *interrupciones condicionales* son FERRARI [Kanawati *et al*. 1992, Kanawati *et al*. 1995], DEFINE [Kao e Iyer 1994], Xception [Carreira *et al*. 1998] y [Maia *et al*. 2002], EXFI [Benso *et al*. 1998], NFTAPE [Stott *et al*. 2000] y SFIDA [Lee *et al*. 2000].

Inserción de código: El agente de activación está formado por un conjunto de instrucciones cuya ejecución dispara la inyección. Estas instrucciones se insertan como parte del código de una aplicación, y están orientadas a provocar un fallo antes de que el sistema ejecute algún tipo de acción determinada. Por ejemplo, en un sistema distribuido, se puede alterar el valor de un mensaje antes de su transmisión.

Algunos ejemplos de herramientas diseñadas con un agente de activación del tipo *inserción de código* son FIAT [Segall *et al*. 1988], FERRARI [Kanawati *et al*. 1992, Kanawati *et al*. 1995], ORCHESTRA [Dawson *et al*. 1996], NFTAPE [Stott *et al*. 2000], MAFALDA [Rodríguez *et al*. 1999] y FIMD-MPI [Blough y Liu 2000].

SWIFI en tiempo de ejecución "real-time": Es una técnica poco conocida hasta el momento. Podemos destacar la implementación basada en el estándar NEXUS de IEEE, como la de **INERTE** [Yuste *et al*. 2003 (a) y (b)]. Utiliza los recursos de depuración de NEXUS [NEXUS] para la inyección y observación del sistema. La ventaja es que permite el acceso a nivel RTL de cualquier elemento integrado durante la ejecución de la carga, sin necesidad de parar o alterar la ejecución. Además, no necesita añadir rutinas o elementos software como agentes de activación o inyección, por lo que la sobrecarga inducida sobre el sistema es nula. Está específicamente orientada a SoC[40] y requiere que el sistema contemple el estándar de NEXUS en su arquitectura.

3.3.4.2. Algunos ejemplos de herramientas SWIFI

La mayoría de las herramientas SWIFI implementadas hasta el momento son herramientas "runtime". Provocan inversiones a nivel RTL durante la ejecución de la carga, permitiendo la inyección de fallos transitorios. Al final de la sección se presenta la única herramienta "real-time" desarrollada hasta el momento.

FIAT ("Fault Injection-based Automated Testing Environment") [Segall *et al*. 1988, Barton *et al*. 1990] es una de la primeras herramientas automáticas de inyección implementas por software. Consiste en varios nodos (IBM RT PC) conectados mediante una red *token-ring*, donde uno de ellos actúa como agente de activación (FIM) y el resto como candidatos o receptores (FIRE). El agente de inyección implementado en cada nodo FIRE actúan bajo el control de FIM. Esta herramienta, desarrollada por la Universidad de Carnegie Mellon, permite la inyección de fallos en los segmentos de memoria de datos y código, estableciendo a priori el tipo y localización del fallo que se va a inyectar. Puede realizar hasta ocho pegados en la misma palabra, o hasta dos inversiones. La observación del sistema tras la inyección se realiza utilizando la redundancia de un nodo replicado, donde una réplica es la inyectada y la otra funciona libre de fallos. Las salidas de ambas son comparadas por el nodo FIM a fin de determinar si el error es detectado correctamente antes de propagarse fuera del nodo.

EFA, de la Universidad de Dortmund [Echtle y Leu 1992], es un inyector orientado a generar averías bizantinas en un sistema distribuido. Sin embargo, no describe ningún tipo de especificación tolerante a fallos para el protocolo de comunicaciones. **ProFI** ("Processor Fault Injector") [Lovric y Echtle 1993] fue desarrollado en el mismo centro con el objetivo de detectar los fallos de diseño mediante la inyección de fallos permanentes en los registros del procesador.

FERRARI fue desarrollada en la Universidad de Texas [Kanawati *et al*. 1992, Kanawati *et al*. 1995], sobre una estación SUN SPARC. Utiliza tanto interrupciones condicionales (con *traps* software) como temporizadores e inserción de código. También permite inyección "pre-runtime". Permite emular fallos físicos, con los modelos de pegado e inversión de un bit o in byte en memoria de datos y código, así como errores de control de flujo en estaciones SUN SPARC.

[40] Systems-on-Chip

FINE ("Fault Injection and moNitoring Environment"), de la Universidad de Illinois [Kao *et al*. 1993], es una herramienta de inyección que también actúa como monitor del sistema. FINE puede emular fallos en la CPU, en memoria y en las líneas de bus, con el objetivo de estudiar la propagación de fallos en el núcleo del sistema operativo UNIX. **DEFINE** [Kao e Iyer 1994] es una evolución de FINE hacia sistemas distribuidos. Utiliza dos tipos de agentes de activación: los temporizadores y las interrupciones condicionales (*traps* software). DEFINE se probó con dos protocolos UDP y TCP, omitiendo y corrompiendo los mensajes transmitidos por la red.

SFI ("Software Fault Injector"), de la Universidad de Michigan [Rosemberg y Shin 1993], puede inyectar fallos de tres formas diferentes. En memoria, con inyección "runtime" utilizando como agente de activación un temporizador interno; mediante la generación de fallos "pre-runtime" en la secuencia de ejecución; y como tercera opción, generando errores de omisión y retardos en la comunicación. La implementación de esta tercera opción se realiza compilando varias versiones con errores. Inicialmente se ejecuta una versión libre de errores y cuando se activa la inyección se sustituye aleatoriamente por otra versión errónea. **DOCTOR** ("integrateD sOftware implemented fault injeCTiOn enviRonment") [Han *et al*. 1995], se desarrollo a partir de SFI, con el objetivo de aumentar su portabilidad. **CSFI** ("Communication Software Fault Injector") es una herramienta desarrollada durante el proyecto europeo FTMPS nº 6731 con el fin de inyectar fallos de comunicaciones en computadores paralelos (averías bizantinas).

ORCHESTRA, desarrollada en la Universidad de Michigan [Dawson *et al*. 1996], está orientada a sistemas distribuidos. El agente de activación es una nueva capa (PFI – "Protocol Fault Injection-layer") insertada entre el nivel de aplicación y el protocolo de comunicaciones. Los mensajes son filtrados y manipulados en esta capa de inyección antes de ser transmitidos. También es capaz de generar mensajes espurios, retrasarlos, duplicarlos y en general, provocar averías bizantinas. Inicialmente desarrollada para "Real-Time Mach", la herramienta se amplió para plataformas Solaris y SunOS.

SOFI ("Software Fault Injector"), de la Universidad Politécnica de Valencia [Campelo 1999], Grupo de Sistemas Tolerantes a Fallos, está orientada a sistemas distribuidos. Utiliza periféricos internos al procesador del nodo como agentes de activación. Los modelos son el pegado y la inversión tanto en memoria de código como en memoria de datos. En [Blanc *et al*. 2001] se presenta una comparación entre la inyección SWIFI y la inyección física a nivel de pin utilizando las herramientas SOFI y AFIT. Se muestran las diferencias en el tipo de inyección y como se reflejan en los resultados obtenidos. Además se propone un método para realizar, en el caso de utilizar diferentes técnicas para la validación de un mismo sistema, una evaluación global mediante **Estratificación** [ver también: STSARCES 1999].

Xception, de la Universidad de Coimbra [Carreira *et al*. 1998], es una herramienta que aprovecha los mecanismos internos de depuración de algunos procesadores (inicialmente PowerPC) como agentes de activación e inyección. Implementa los modelos de pegado e inversión en memoria de código y datos, disparados ante la decodificación de una instrucción predeterminada, o simplemente ante la expiración de un tiempo aleatorio (temporización). **Xception**™ [Maia *et al*. 2002] es una nueva versión para Intel Pentium y SPARC y plataformas LynxOS, SMX, Windows NT/2000 y Linux, que incorpora una versión que permite mutaciones en el software y es compatible con otras herramientas de inyección, como por ejemplo inyección física a nivel de pin.

M. Rebaudengo y M. Souza Reorda, del Politécnico de Torino [Rebaudengo *et al*. 1998], utilizan el "**Trace Exception Mode**", disponible en varios microprocesadores,

para activar la inyección del fallo. Es otro ejemplo de utilización de mecanismos de depuración como activadores de la inyección.

L. Impagliazzo publica la "FAULT List Collapsing" [Impagliazzo *et al*. 1998] como un conjunto de reglas que permiten reducir el número de experimentos SWIFI a realizar sobre un sistema. El objetivo es generar una lista de fallos que sean representativos y que mejoren el análisis de los resultados. **EXFI** ("EXception-based Fault Injector"), de los mismos autores [Benso *et al*. 1998], es una herramienta que utiliza las excepciones del procesador para llevar a cabo la inyección de fallos seleccionados según su trabajo anterior. Utiliza una placa comercial M68KDIP, con un procesador M68040. Aunque es una técnica de bajo coste, el gran inconveniente que se deduce es el alto grado de intrusión y un factor de retardo relativamente alto. **FlexFi** [Benso *et al*. 1999] se desarrollo a partir de EXFI, y además del disparo de la inyección mediante excepciones también incorpora otro híbrido, donde el agente de activación es una interrupción externa provocada por un circuito adicional. También introduce otra técnica, utilizando el modo de diagnóstico incorporado en algunos procesadores y controladores, el "**Background Diagnostic Mode**" (BDM).

FIMD-MPI [Blough y Liu 2000] es una herramienta SWIFI que provoca averías bizantinas en sistema distribuidos. Utiliza como agentes de activación tanto temporizadores como inserción de código. La herramienta se ha utilizado para validar algoritmos de sincronización de la base de tiempos de terminales Solaris o Linux conectadas a la misma red.

SFIDA [Lee *et al*. 2000] es una herramienta SWIFI orientada a sistemas distribuidos para plataformas Linux. Es capaz de inyectar fallos sobre la aplicación usuario, pero no sobre el sistema operativo. También utiliza mecanismos de depuración como agentes de activación e inyección.

MAFALDA del Laboratoire d'Architecture et d'Analyse des Systemes (LAAS) [Rodríguez *et al*. 1999] genera parámetros inválidos en la invocación de las primitivas del *microkernel*, utilizando inserción de código. También inyecta fallos en los segmentos de datos y código asignado al *microkernel*. **MAFALDA-RT** [Arlat *et al*. 2002] es una versión para sistemas de tiempo real no distribuidos, donde se trata de mejorar el diseño de la herramienta reduciendo la intrusión temporal que la inserción del código supone.

THESIC ("Testbed for Harsh Environment Studies on Integrated Circuits") [Velazco y Rezgui 2000, Velazco *et al*. 2000] es una herramienta para sistemas empotrados. A pesar de ser una herramienta SWIFI, utiliza una placa complementaria específica para cada sistema a validar. Sin embargo, esta herramienta puede utilizarse a modo de observador del sistema y la técnica es compatible con otras técnicas de inyección como la radiación de partículas.

FIRE [Martins y Rosa 2000] y **Jaca** [Martins *et al*. 2002] son herramientas SWIFI para aplicaciones orientadas a objetos, basadas en la **programación reflexiva**. La *reflexión* permite al sistema aprender de sus propias acciones. Divide la estructura de la aplicación en dos niveles, el nivel base y el *meta-nivel*. El nivel base es el nivel de definición. El meta-nivel permite la observación y manipulación de las estructuras de datos, así como de las acciones realizadas en el nivel base. Es a este nivel donde se implementan las tareas de inyección y observación.

Para la validación de los distintos prototipos del controlador **TTP^{TM}/C**, desarrollado por la Universidad de Viena y basado en la arquitectura TTA ("Time-Triggered Architecture"), cabe destacar tres trabajos. El primero, el de **E. Fuchs** [Fuchs 1996], que

llevó a cabo una serie de experimentos sobre el primer modelo del prototipo TTP™/C, llamado MARS. Es un sistema distribuido de tiempo real especialmente orientado a aplicaciones empotradas que se consideran críticas. TTP/C asegura que los nodos del sistema son tolerantes a fallos. Los nodos se llaman Unidades Tolerantes a Fallos o FTU y ante un error, la unidad deber ser capaz de tratarlo o de permanecer en silencio (no envía ningún mensaje por el canal de comunicaciones). Pero en cualquier caso, el error debe contenerse dentro de la FTU y no propagarse a otras unidades del sistema. La inyección sobre los segmentos de datos y código de la aplicación se realiza "pre-runtime" (fallos permanentes). También se corrompe el contenido de los mensajes a enviar, mediante inyección "runtime".

El segundo trabajo, de **R. Hexel** [Hexel 1999] utiliza un nuevo inyector, similar al anterior, para la validación de la segunda versión del prototipo TTP/C. Los resultados se comparan con los obtenidos con interferencias electromagnéticas. La conclusión de este trabajo es la necesidad de utilizar un módulo adicional en la arquitectura del sistema: el guardián del bus. Su función es la de evitar que una FTU errónea genere averías bizantinas sobre el canal de comunicaciones.

Finalmente **A. Ademaj** utiliza inyección SWIFI para validar los dos últimos prototipos comercializados del TTP™/C, el TTP/C-C1 y TTP/C-C2. Estos experimentos, realizados bajo el proyecto FIT ("Fault Injection for TTA") subvencionado con fondos europeos IST-2000-25425, tenían dos finalidades. La primera, validar los mecanismos tolerantes a fallos implementados en el protocolo de comunicaciones [Ademaj 2002 (a) y (b)] y la segunda, obtener la cobertura del sistema ante fallos provocados en el segmento de memoria de datos del microcontrolador que alberga la aplicación y en el segmento de memoria del microcontrolador de comunicaciones destinado al almacenamiento y tratamiento de los mensajes [Ademaj 2003]. En [Blanc *et al*. 2002 (a)] se presenta un esquema de la localización de los fallos en la arquitectura del sistema y de cómo se puede utilizar esta técnica como complemento de la inyección física, en concreto, de la radiación de partículas α y de la inyección a nivel de pin.

Finalmente, la única herramienta desarrollada hasta el momento con el modelo "real-time" es **INERTE** ("Integrated NEXUS-based Real-Time Fault Injection Tool for Embedded Systems") desarrollada en la Universidad Politécnica de Valencia por el grupo de Sistemas Tolerantes a Fallos [Yuste *et al*. 2003 (a) y (b)]. Utiliza el estándar de NEXUS tanto para la inyección como para la observación de sistemas empotrados. La sobrecarga que genera es nula y el acceso es a nivel RTL. El modelo de fallos utilizado es la inversión de uno o varios bits en memoria de datos, de código y en registros, incluyendo los registros de los periféricos. El fallo se activa según una condición observada en la ejecución de la carga o la expiración de un tiempo preestablecido. El fallo permanece activo hasta el final del experimento.

3.3.4.3. Otras herramientas compatibles con SWIFI

FTAPE ("Fault Tolerance And Performance Evaluator"), de la Universidad de Illinois [Tsai e Iyer 1995], es una herramienta orientada a comparar diferentes sistemas tolerantes a fallos. Consiste en un generador de cargas de estrés que incrementan la actividad de alguna parte concreta del sistema [Tsai *et al*. 1999], sobre la que además, se inyectan fallos. **NFTAPE** [Stott *et al*. 2000] es una versión actualizada para sistemas distribuidos. Incorpora varios agentes de activación basados en la inserción de código o la utilización de mecanismos de depuración. Es compatible con otras técnicas de inyección como inyección física a nivel de pin.

Loki, de la Universidad de Illinois [Chandra *et al*. 2000], es una herramienta abierta

a la que se le pueden incorporar diferentes agentes de inyección. Basada en máquinas de estados, la inyección se ha de realizar en un nodo, en base a una visión parcial que Loki proporciona del estado de los otros nodos o componentes del sistema.

Ballista ("Automated Robutness Testing Apprach") [Koopman 1999] se utiliza para verificar el tratamiento de excepciones del sistema operativo. Es la primera herramienta software que se plantea bajo el concepto de **caja negra**, para el cual no es necesario tener un conocimiento exhaustivo del sistema, sino sólo de la unidad de entrada-salida. Está orientada a la verificación de COTS[41]. Se basa en la generación de cargas excepcionales, como por ejemplo, parámetros inválidos en las llamadas al sistema operativo, que ponen a prueba el sistema bajo test. Son los llamados test de robustez y se ha probado con la API de Microsoft Win32. Windows™ 95, Windows™ 98, Windows™ 98 SE y Windows™ CE probaron ser vulnerables (no robustos), mientras que Windows™ NT, Windows™ 2000 y Linux pasaron satisfactoriamente los tests de robustez.

3.3.5. Inyección de fallos basada en técnicas de simulación de modelos

Esta técnica no requiere de un prototipo físico y se puede aplicar desde las primeras etapas del diseño. Una de sus ventajas es su aplicación para la detección temprana de fallos de diseño, evitando que éstos se propaguen a otra etapa del proceso de desarrollo y resulte más costosa su reparación.

La complejidad del modelo simulado abarca desde el circuito integrado más sencillo hasta un sistema distribuido completo. La accesibilidad de la herramienta y el control de la inyección dependen del nivel de detalle descriptivo del modelo. Entre sus ***ventajas*** destacamos que los experimentos son totalmente reproducibles, obteniendo medidas de tiempo sobre acciones determinadas observadas sobre el modelo del sistema.

Sin embargo, como ***inconvenientes*** destaca el elevado tiempo de simulación, así como el esfuerzo necesario para desarrollar una nueva herramienta completa, especialmente si se quiere conseguir procesos de inyección automatizados.

Las herramientas basadas en técnicas de simulación se pueden dividir en función del nivel de abstracción empleado. Diferenciamos entre el nivel eléctrico, el nivel de puerta o lógico, el nivel de RTL y el nivel de sistema. En este último nivel se hace especial hincapié en las herramientas de simulación de modelos VHDL y en la emulación de fallos con FPGA.

3.3.5.1. Nivel eléctrico

La simulación de fallos a nivel eléctrico implica efectuar variaciones tanto en la intensidad de corriente como en los niveles de voltaje del modelo, proporcionando información detallada sobre el impacto de una variación no esperada en partes vitales de un circuito. Este tipo de simulación está orientada a circuitos con componentes digitales y analógicos no demasiado complejos que no pueden ser completamente caracterizados a niveles superiores.

Un ejemplo es la herramienta **FOCUS** [Choi e Iyer 1992] en la que se simula el efecto de la radiación. Se utilizó para analizar un microcontrolador específico para el Boeing 747 y 757. Aunque normalmente el tiempo de simulación de cualquier modelo es

[41] Commercial Off-The-Self

bastante elevado, FOCUS alivia este tiempo dividiendo la simulación en dos simuladores, uno digital (más rápido) y otro analógico (más lento). La simulación analógica se centra en el efecto del fallo en el transistor, mientras que la propagación del posible error ocasionado a otras partes funcionales del controlador se simula a nivel digital.

Otro trabajo centrado a nivel de transistor es el descrito en [Linden y Dahlgren 1995 (a) y (b)] sobre modelado de pegados a nivel de transistor en la tecnología CMOS.

3.3.5.2. Nivel lógico

La simulación a nivel lógico observa el comportamiento de un dispositivo a través sus operaciones binarias, obteniendo salidas discretas e información temporal aproximada. Se utilizan modelos de fallos, sobre las entradas de las puertas lógicas, tales como pulsos o pegados transitorios y permanentes.

Ejemplos de simulación a nivel lógico son los presentados en [Lomelino e Iyer 1986] y [Czeck y Siewiorek 1990]. Ambos utilizan el pegado como modelo de fallo, el primer trabajo, sobre una descripción esquemática del procesador AMD 2901 y el segundo, sobre un modelo del procesador IBM RT PC en Verilog, lenguaje de descripción HDL. En este estudio se destaca la dependencia que existe entre las medidas obtenidas y la carga ejecutada por el procesador.

3.3.5.3. Nivel RTL

La simulación de fallos a nivel RTL implica un análisis más complejo del comportamiento del dispositivo durante las operaciones de lectura y escritura tanto en registros como en memoria. Es la técnica más parecida a la emulación de fallos que se realiza con SWIFI. En caso de que sea posible ejecutar una carga en el modelo, con idénticas características a la que se ejecuta sobre el prototipo, podemos considerar el análisis de fallos en el software como utilidad de la simulación a nivel RTL.

En los trabajos presentados en [Ohlsson *et al*. 1992] y [Rimén y Ohlsson 1993] se utiliza el modelo de un procesador de 32 bits, con *watchdog* interno, descrito en lenguaje VHDL. El modelo de fallo utilizado es la inversión del valor lógico de un bit almacenado.

J. Karlsson compara en [Karlsson 1990] los resultados obtenidos simulando un modelo del microprocesador de 8-bits MC6809E con los obtenidos en un prototipo real sometido a radiación.

Un ejemplo que combina la simulación a nivel RTL con la técnica SWIFI es ASPHALT [Yount y Siewiorek 1996]. Los experimentos se realizan sobre un procesador RISC de IBM: ROMP. Los resultados demuestran la gran similitud entre ambas técnicas. La diferencia estriba en el tiempo de simulación. Mientras que los experimentos en el prototipo son muy rápidos, las simulaciones consumen un tiempo relativamente alto.

3.3.5.4. Nivel de sistema

A este nivel se modela el sistema completo, tanto sus componentes hardware como software. Los resultados de la simulación se utilizan para identificar los mecanismos cuyo funcionamiento no es el esperado o deseado, así como las carencias que pueda tener el modelo en la detección de errores causados por fallos físicos o fallos en el software.

Las herramientas que se presentan a nivel de sistema son herramientas de desarrollo.

Proporcionan las condiciones para el modelado, con la posibilidad de añadir al modelo elementos propios de la inyección y mecanismos de detección de errores como pueden ser los votadores.

NEST ("NEtwork Simulation Testbed") [Dupuy *et al*. 1990] es un ejemplo típico. Se trata de un entorno gráfico de simulación bajo UNIX para el análisis de sistemas y algoritmos distribuidos. Utilizando un conjunto de herramientas gráficas, el usuario puede desarrollar modelos de procesadores y redes de comunicación.

Otro estudio orientado a sistemas distribuidos es el presentado en [Echtle y Chen 1991]. Se inyectan fallos en los mensajes transmitidos a través del canal de comunicaciones para validar el protocolo de comunicaciones.

DEPEND [Goswami e Iyer 1991, Goswami *et al*. 1997] propone una biblioteca de clases de objetos C++, tanto elementales como complejos, a partir de los cuales el usuario construye su modelo por instanciación y composición. Se utilizó para estudiar el efecto de errores correlativos y errores simples, con diferentes latencias de activación, en un sistema con triple redundancia TMR[42].

REACT [Pradhan y Clark 1993] es una herramienta automatizada empleada en la validación de diferentes arquitecturas de multiprocesadores. Es un entorno que permite el desarrollo de sistemas monoprocesador o sistemas distribuidos. El comportamiento de un procesador en presencia de fallos se modela con una tasa de ocurrencia de errores sobre la unidad de entrada y salida o el canal de comunicaciones. También se pueden tener presentes los fallos en el software.

En [Maxion y Olszewski 1993] se describen una serie de experimentos para diagnosticar la capacidad del sistema para tratar fallos en redes de área local. [Jogannath y Rai 1995] estudia el impacto de esos fallos en el nivel de enlace de datos.

En [Avresky *et al*. 1992] se inyectan fallos para mejorar los mecanismos de tolerancia a fallos en la detección de errores. En [Goswami e Iyer 1993] los fallos se inyectan en el espacio de memoria mientras se simula la ejecución de la carga.

Bones (BONeS™ Designer de Cadence Design Systems) es una herramienta comercial orientada a la evaluación de la confiabilidad de sistemas tolerantes a fallos. La herramienta es en realidad un entorno que permite implementar protocolos de comunicaciones mediante librerías y diagramas de bloques jerárquicos, e incluso funciones en C/C++. Este entorno de simulación se ha utilizado con versiones del protocolo de comunicaciones TTP. Aparte de la estructura hardware del sistema, también se incluyó en el modelo el protocolo de comunicaciones y una aplicación [Pallierer 2000]. Por ejemplo, una de las aplicaciones probadas es el control de movimiento de los *flaps* de un avión [Pallierer y Smith 1998]. También se utilizó para validar el algoritmo de *startup* del propio protocolo de comunicaciones [Steiner 2001].

C-Sim [CSim 2002] es otro entorno de programación en C para el diseño de modelos discretos. Una vez el modelo está diseñado, se lleva a cabo un procesamiento pseudo-paralelo del modelo en un PC compatible bajo Windows™. Es una herramienta portable y con cierto nivel de independencia con respecto a la plataforma sobre la que se ejecuta. Un ejemplo de la utilización de esta herramienta es la que se llevó a cabo durante el proyecto Europeo FIT. Se construyó un modelo del controlador TTP™/C en base a sus especificaciones de diseño. En [Grillinger *et al*. 2002] se muestra como el modelo tiene

[42] Tripple Modular Redundant

un comportamiento equivalente al prototipo en presencia de fallos. Sin embargo, la gran ventaja del entorno C-Sim es que permite determinar ambigüedades o carencias en las especificaciones del sistema.

3.3.5.5. Inyección de fallos basada en VHDL

Actualmente, una gran parte de los modelos se desarrollan en VHDL ("Very High Speed Integrated Circuit Hardware Description Language"), un lenguaje de descripción hardware. La simulación se realiza normalmente con herramientas EDA ("Electronic Design Automation") y la inyección de fallos con herramientas especialmente desarrolladas para tal fin. Los modelos de fallos que se pueden utilizar durante la simulación son muy variados. Aparte de los típicos pegados e inversiones, también se pueden modelar pulsos a nivel de puertas lógicas, indeterminaciones y retardos.

En VHDL, la descripción de un componente individual se expresa mediante la declaración de una entidad y su arquitectura. La declaración de la entidad incluye el conjunto de señales de entrada y salida del componente, mientras que la arquitectura describe su organización interna. La descripción del componente puede ser **estructural**, **comportamental** o una mezcla de ambas. La *estructural* se basa en la composición jerárquica de los componentes declarados y de la distribución de las señales entre ellos. La *comportamental* se basa en la descripción algorítmica del modelo y de la función que realiza cada componente, expresada mediante procesos. Los procesos se comunican a través de las señales declaradas.

Existen dos métodos de inyección de fallos en modelos VHDL [Arlat *et al*. 1993], la inyección sin modificar el código en VHDL y la inyección basada en su modificación. Sin modificar el código en VHDL, la inyección se realiza utilizando un simulador de comandos localizado en el compilador de VHDL. Modificando el código en VHDL, existen dos posibles formas de inyectar fallos. La primera se basa en añadir componentes específicos, los llamados **saboteadores**. La segunda posibilidad se basa en la mutación de componentes ya existentes en el modelo, generando componentes específicos llamados **mutantes**.

En [Gracia *et al*. 2001] se presenta una comparación de ambos métodos de inyección. La utilización de un simulador de comandos tiene un menor coste de implementación (en relación al esfuerzo). Saboteadores y mutantes tienen un coste de implantación mayores, debido a los altos tiempos de recompilación y simulación que ambos métodos suponen. Con saboteadores, el tamaño de los ficheros de traza se incrementa, ya que son necesarias más señales de control. Los mutantes incrementan el tiempo de simulación debido a la constante necesidad de salvar y restaurar el estado del sistema.

Un ejemplo destacable de estas herramientas es **VFIT** [Baraza *et al*. 2002] desarrollada por el Grupo de Sistemas Tolerantes a Fallos (GSTF) de la Universidad Politécnica de Valencia. Es una herramienta que trabaja con un simulador comercial de VHDL (Modelsim de Model Technology). Presenta una interfaz amigable para el usuario en entorno Windows™. Es una herramienta genérica que puede soportar modelos de baja y media complejidad. Utiliza un simulador de comandos para inyectar fallos cuyos modelos abarcan el pegado, inversión, indeterminación, pulso y retardo. Existe una versión en desarrollo que implementa los métodos de saboteadores y mutantes. De esta herramienta cabe destacar el trabajo realizado durante el proyecto Europeo FIT [Gracia *et al*. 2002 (a) y (b), Gracia *et al*. 2003] para la validación del modelo en VHDL del controlador TTP™/C.

MEFISTO [Jenn *et al*. 1994] es otra herramienta de inyección de fallos en VHDL. Fue desarrollada durante el proyecto PDCS2 [PDCS2] y es capaz de inyectar fallos en el valor de las señales, en variables, mutación en los componentes y también soporta saboteadores. Existen dos versiones, **MEFISTO-L** [Boue *et al*. 1998] del LAAS (Toulouse) y **MEFISTO-C** [Folkesson *et al*. 1998] de la Universidad Técnica de Chalmers (Goteborg).

Otra técnica es la propuesta con **VERIFY** ("VHDL-based Evaluation of Reliability by Injecting Faults") [Sieh *et al*. 1997]. Presentan una herramienta software que introduce una nueva forma de describir el comportamiento de los componentes físicos del sistema en presencia de fallos, incluyendo la frecuencia de ocurrencia de los fallos, mediante una extensión del lenguaje en VHDL.

3.3.5.2. Emulación de fallos con FPGA

Se basa en el uso de FPGA ("Field Programmable Gate Arrays") para construir prototipos físicos preliminares a la versión definitiva y orientados a la detección temprana de fallos de diseño. El modelo del sistema se obtiene a partir de las especificaciones. El objetivo ideal de la emulación de fallos con FPGA sería complementario a la simulación. Después de posibles mejoras y evaluaciones del modelo, éste se implementa sobre una FPGA a fin de poder inyectar fallos físicos y analizar el comportamiento de los mecanismos de tolerancia a fallos. Sin embargo, debido a los altos tiempos de simulación que conllevan muchos (la mayoría) de los modelos, la emulación con FPGA se ve como una alternativa, no como un complemento. El modelo se implementa directamente en la FPGA. Los fallos que se introducen en el dispositivo emulan el efecto de los fallos físicos, tratando de realizar campañas similares a las que se hacen en simulación, con la ventaja de reducir los tiempos de los experimentos.

Las primeras aproximaciones de esta técnica se encuentran en [Li y Wu 1995] y [Burgun *et al*. 1996] que contemplan sólo la inyección de fallos estática, es decir, deteniendo el funcionamiento del dispositivo para reconfigurar el modelo incluyendo un error.

[Rebaudengo *et al*. 2002] es un ejemplo de inyección dinámica, donde el modelo implementado en la FPGA es "inyectable". No es necesario detener la ejecución, sino que el modelo dispone de mecanismos que permiten provocar alteraciones emulando los efectos de los fallos físicos.

Otros trabajos de inyección sobre FPGA recientes son los presentados en [Huang *et al*. 1999] y [Wu *et al*. 1998] donde se utiliza sólo el pegado permanente como modelo de fallo. **FIDYCO** [Rahbaran *et al*. 2002] y **FIFA** [Civera *et al*. 2001] son dos herramientas para el desarrollo e implementación de modelos en FPGA.

3.4. Comparación de las Técnicas de Inyección de Fallos

La información publicada en trabajos de investigación es muy amplia. La aportación de unas propiedades aplicables a cualquier técnica de inyección, que describen de manera esquemática sus características más relevantes, puede resultar de gran utilidad a la hora de comparar dos técnicas entre sí.

La tabla 3.1. muestra una relación sobre el grado de adaptación de las diversas técnicas de inyección de fallos a los requisitos deseables durante la validación experimental. En el proyecto FIT [FIT 2002] fueron evaluadas y discutidas las

propiedades de cada una de las técnicas de inyección implicadas en el proyecto. Sin embargo, es necesario tener en cuenta las siguientes consideraciones:

Sobre la *reproducción de los experimentos*: Hace referencia a una reproducción determinista del experimento. Quiere decir, repetir un experimento simple con el mismo dominio de entrada (fallo y activación). La propiedad está ligada al control espacial y temporal. Cuando el control es alto, como en el caso de la simulación, cadenas de exploración o SWIFI, la reproducción de los experimentos también será alta. En inyección física a nivel de pin o radiación láser, el control espacial es muy alto, pero con respecto al control temporal, aquél que indica el nivel de sincronización entre la herramienta y el servicio, siendo herramientas externas, el control es relativo, de ahí que se defina como "medio". En el caso de EMI o fuentes de radiación de partículas, no existe un buen control ni espacial ni temporal, lo que impide una reproducción determinista de los experimentos permitiendo sólo apreciaciones estadísticas.

Sobre la *duración de los experimentos*: Se califica como "media" la duración de los experimentos con fuentes de radiación, radiación láser y EMI porque no implementan mecanismos que detecten la efectividad del fallo en el mismo instante de la inyección, sino que se necesita comparar la traza del servicio con otra libre de fallos. Además, la delimitación temporal del experimento se prolonga por encima de lo estrictamente necesario, ya que no pudiéndose predecir de antemano el momento en el que se activa el posible error, hay que esperar un margen de tiempo prudencial.

Sobre la *observabilidad*: Es fácil dividir las herramientas de inyección según la observabilidad. Aquellas que se desarrollan internas al sistema (SWIFI y cadenas de exploración) o aquellas que integran el sistema como parte de la herramienta (simulación) tendrán un nivel de observación muy alto. Para las herramientas externas, como es el caso de todas las de inyección física, la observabilidad del sistema se define como la observación externa del servicio en presencia de fallos por parte del usuario.

Sobre la *intrusión*: Las que mayor intrusión tienen son aquellas que afectan directamente a la unidad de entrada y salida del prototipo físico: EMI e inyección a nivel de pin. Con AFIT está intrusión se ha reducido considerablemente añadiendo terminadores en las puntas de inyección especialmente diseñados para tal efecto.

Sobre la *sobrecarga*: La ventaja en la observabilidad que adquieren las herramientas integradas en el sistema (simulación, SWIFI y cadenas de exploración) resulta un inconveniente al medir la sobrecarga que la herramienta supone. En este apartado, las herramientas externas son la que presentan una sobrecarga nula.

Sobre el *riesgo*: Las fuentes de radiación sí que presentan un riesgo tanto para el hombre como para los componentes del sistema que pueden ser dañados irreversiblemente. La inyección a nivel de pin y EMI tienen un riesgo, aunque muy bajo, sólo sobre los componentes del sistema. Con respecto a radiación láser, depende de la herramienta que se utilice. Las primeras aplicaciones de esta técnica eran totalmente destructivas. Las aplicaciones más recientes han demostrado ser menos agresivas que la radiación con fuentes radiactivas.

Sobre la *reutilización*: Ni la utilización de cadenas de exploración, ni SWIFI "runtime" o "pre-runtime" suelen ser técnicas genéricas. Sin embargo, existe una herramienta comercial **DEFI** [Gérardin 1986] basada en un emulador de EPROM para inyección "pre-runtime" que puede considerarse genérica. En general, se propone un nivel "bajo" en vez de nulo ya que existen técnicas que son reutilizables con varios sistemas, siempre que se basen en la misma arquitectura. Con respecto a simulación,

algunos lenguajes como VHDL presentan herramientas de inyección como tales, pudiendo ser genéricas.

Sobre el *potencial de automatización*: La tabla da una apreciación orientativa. Sin embargo, este parámetro depende de la herramienta y no de la técnica. Quizá las que presentan un potencial de automatización peor son aquellas que tienen un control temporal peor o una gran dificultad de aplicación.

Sobre el *coste de desarrollo*: Ninguna herramienta de inyección de fallos se puede calificar como económica. Las que presentan un menor coste de desarrollo son las herramientas basadas en SWIFI, ya que no necesitan material, sino un personal cualificado para realizar una programación eficiente. También es la técnica más rápida de implementar. Aunque como contrapartida, no es reutilizable, con lo que la amortización depende de los beneficios obtenidos sobre un sistema en particular. Por otro lado, también se considera el esfuerzo de desarrollo. Cuanto más compleja sea la herramienta y más facilidades se le atribuyan, mayor será el esfuerzo. Por otro lado, dichas facilidades aliviarán la dificultad de implantación y mejorarán la observabilidad. Por otra parte, a SWIFI se le atribuye un esfuerzo medio, pero la realidad es que es totalmente dependiente de la complejidad de la arquitectura del sistema.

Sobre el *coste de la infraestructura*: Las herramientas desarrolladas bajo la técnica de inyección física a nivel de pin, así como la técnica SWIFI de “real-time” son autocontenidas. Quiere decir que la propia herramienta aporta todos los recursos necesarios para su aplicación, lo que por regla general, encarece su coste de desarrollo. Por otra parte, la utilización de cadenas de exploración no presenta un coste de infraestructura bajo, porque por regla general, necesita algún recurso extraordinario como por ejemplo, placas específicas de evaluación proporcionadas por el propio fabricante y que no suelen ser económicas.

	Simulación a nivel de sistema		***HWIFI***					***SWIFI***		
	VHDL	***Otros lenguajes***	***Nivel de Pin***	***Radiación de partículas***	***EMI***	***Radiación láser***	***Cadenas de exploración***	***"Pre-runtime"***	***"Runtime"***	***"Real-Time"***
Nivel de abstracción	Estructural Comportamental	Comportamental	Prototipo	Prototipo	Prototipo	Prototipo	Prototipo	Prototipo	Prototipo	Prototipo
Naturaleza de los fallos	Simulación de fallos físicos y fallos en el software	Simulación de fallos físicos	Fallos físicos	Fallos físicos	Fallos físicos	Fallos físicos	Fallos físicos	Emulación de fallos físicos y fallos en el software	Emulación de fallos físicos y fallos en el software	Emulación de fallos físicos y fallos en el software
Accesibilidad	Completa sobre el modelo	Completa sobre el modelo	Entrada Salida	Contenida en semiconductor	Entrada Salida	Contenida en semiconductor	Registros incluidos en la cadena de exploración	RTL accesible por software	RTL accesible por software	RTL accesible por software
Control espacial	Alto	Alto	Alto	Bajo	Bajo	Alto	Alto	Alto	Alto	Alto
Control temporal	Alto	Alto	Medio	Bajo	Bajo	Medio	Alto	Alto	Alto	Alto
Multiplicidad	Fallos simples y múltiples	Fallos simples y múltiples	Fallos simples y múltiples	Indefinida	Indefinida	Fallos simples	Fallos simples y múltiples	Fallos simples y múltiples	Fallos simples y múltiples	Fallos simples y múltiples
Reproducibilidad de los experimentos	Alta	Alta	Media	Baja	Baja	Media	Alta	Alta	Alta	Alta
Duración de los experimentos	Alta	Alta	Baja	Media	Media	Media	Alta	Baja	Baja	Baja

Tabla 3.1. Comparación de diferentes técnicas de inyección de fallos

	Simulación a nivel de sistema		***HWIFI***					***SWIFI***		
	VHDL	***Otros lenguajes***	***Nivel de Pin***	***Radiación de partículas***	***EMI***	***Radiación láser***	***Cadenas de exploración***	***"Pre-runtime"***	***" Runtime "***	***" Real-Time "***
Observabilidad	Alta	A alto nivel	Media	Media	Media	Media	Alta	Media	Alta	Alta
Intrusión	Nula	Nula	Media	Baja	Media	Baja	Nula	Nula	Nula	Nula
Sobrecarga	-	-	Nula	Nula	Nula	Nula	Alta	Media / Baja	Alta	Nula
Medidas de tiempos	Alta	Alta	Alta	Media	Media	Alta	Alta	-	Alta	Alta
Riesgo	Nulo	Nulo	Bajo	Alto	Bajo	Medio / Alto	Nulo	Nulo	Nulo	Nulo
Reutilización de la herramienta	Alta	-	Alta	Alta	Alta	Alta	Baja	Baja	Baja	Alta
Potencial de automatización	Alto	Alto	Alto	Medio	Medio	Medio	Alto	-	Alto	Alto
Coste de desarrollo	Alto	Alto	Alto	Medio	Medio	Medio	Medio	Bajo	Bajo	Alto / Medio
Coste de la infraestructura	Medio / Alto	Bajo	Bajo	Alto	Alto	Alto	Medio	Bajo	Bajo	Bajo

Tabla 3.1. Comparación de diferentes técnicas de inyección de fallos

3.5. Resumen y Conclusiones del Capítulo

Este capítulo está dedicado a las técnicas de inyección de fallos. Primero, revisa las causas que favorecen la aparición de fallos físicos en los circuitos integrados y cómo se puede modelar su efecto mediante técnicas de inyección. Segundo, se aborda la descripción de la inyección de fallos como método de validación experimental y bajo la que se ha desarrollado un conjunto de técnicas diferentes según la forma de acceso e inyección de los fallos en el sistema. La tercera parte del capítulo revisa y actualiza el estado del arte sobre *inyección de fallos* y establece pautas para la comparación entre técnicas o herramientas de inyección. Esta comparativa se presenta al final del capítulo.

La búsqueda de una solución definitiva en la validación experimental de sistemas tolerantes a fallos ha conducido a la definición de diversas técnicas de inyección, materializadas en sus correspondientes herramientas. Dichas herramientas implementan, en mejor o peor medida, las características propias de la técnica elegida. Esta búsqueda ha perfilado la teoría que define la base, los conceptos y las formas de aplicación de la inyección de fallos sobre modelos y prototipos de sistemas, cuyo objetivo final es la obtención de su confiabilidad. Este capítulo trata de avanzar en la clasificación de los fallos físicos que pueden ocurrir en los sistemas actuales, así como aportar una estructuración al concepto de inyección de fallos a través de los conjuntos de descripción de los dominios de entrada y salida aplicables al proceso de validación.

Aunque existen muchas herramientas de inyección, ninguna se puede considerar totalmente completa. Todas tienen una serie de ventajas e inconvenientes, e incluso, orientaciones distintas. Existen algunos estudios comparativos entre herramientas basadas en la misma técnica o basadas en técnicas distintas. Hasta ahora, la conclusión de estos trabajos es similar, las herramientas son complementarias, ya que difícilmente reproducen los mismos experimentos. Para poder entender mejor cuáles son las diferencias entre técnicas, el capítulo aporta una lista de propiedades descriptivas aplicables a cualquier técnica o herramienta de inyección. Además, tras la revisión de los trabajos de investigación relacionados, se ha tratado de valorar estas propiedades en las técnicas de inyección de fallos más importantes publicadas hasta el momento como una posible comparación orientativa.

En adelante, este trabajo de tesis profundiza en la validación experimental de un sistema distribuido tolerante a fallos de comportamiento crítico. Utiliza una de las técnicas que mayor impulso está teniendo en la actualidad, la inyección física de fallos o HWIFI. En particular, la inyección a nivel de pin. A pesar de las ventajas de otras técnicas de inyección, la inyección física de fallos conserva su importancia por la representatividad de los fallos que inyecta. Mientras la emulación o simulación de fallos físicos requieren de un conocimiento previo del efecto de los fallos sobre el componente, la inyección física de fallos ataca directamente la raíz del problema. Es más, el avance tecnológico es una desventaja para técnicas de emulación o simulación, ya que implica la actualización continua de ese conocimiento previo a la aplicación de la técnica. Una actualización que experimentalmente se puede realizar utilizando HWIFI.

Capítulo 4. La Arquitectura TTA

Arquitectura de Disparo por Tiempo

La complejidad de los *sistemas empotrados* es cada vez mayor. El nivel de integración y las mejoras en la fabricación están permitiendo el desarrollo de complicados diseños de control y procesamiento con costes razonables. La reducción del tamaño de los circuitos VLSI y el incremento de los recursos integrables han hecho que, en ciertas áreas de desarrollo y fabricación, se esté avanzando con celeridad y planteado cambios a nivel electrónico difíciles hace unos años. Por ejemplo, en la industria del automóvil la tendencia actual viene marcada por el cambio de la centralización de los algoritmos de control en una única unidad por la distribución de dichos algoritmos en varias unidades que no estén concentradas en el mismo emplazamiento.

Aparte de las ventajas inherentes de un control distribuido frente a uno centralizado, existe además, la posibilidad de reemplazar componentes hidráulicos y mecánicos por componentes **meca-electrónicos**[43] (componentes electrónicos para el control mecánico). El objetivo de estos componentes es poder diseñar e implantar equipamientos seguros de asistencia a la conducción. Pero si estos equipamientos se diseñan utilizando sólo componentes mecánicos, la complejidad derivada es muy alta con respecto a los costes de producción. Por tanto, la solución sería incrementar la electrónica, asegurando su fiabilidad, y reducir la mecánica [Hammett 1999]. A los equipamientos que utilizan componentes electrónicos para el control mecánico se les denomina **sistemas guiados por cable** ("***x-by-wire***").

La forma de garantizar la fiabilidad y seguridad de un sistema electrónico es desarrollarlo bajo una arquitectura que demuestre una alta confiabilidad, tanto en ausencia como en presencia de fallos. La arquitectura que hasta el momento ha demostrado unos niveles de seguridad más altos es la arquitectura **TTA** ("**Time-Triggered Architecture**"), o Arquitectura de Disparo por Tiempo.

En este capítulo revisa la arquitectura TTA y los protocolos de comunicaciones basados en ella. Entre los protocolos analizados se encuentra el TTP, bajo el que se diseña el controlador de comunicaciones TTP^{TM}/C, objetivo de la validación experimental descrita y analizada en los siguientes capítulos.

4.1. Sistemas Guiados por Cable

En general, las ventajas de un control distribuido dentro del automóvil son mayores que las de un control centralizado por varias razones:

- Los dispositivos de entrada-salida, para sensorizar o para actuar sobre los

[43] Mechatronic Systems

elementos mecánicos, se encuentran ya de por sí distribuidos. Si el control inteligente de dichos dispositivos se localiza cerca de ellos, se minimizan los problemas relacionados con la conexiones.

- La partición de la carga de un sistema entre varios subsistemas tiene dos beneficios. Primero, la sobrecarga en cada subsistema es inferior a la que tiene un único sistema centralizado, mejorando los tiempos de ejecución. Segundo, se reduce el número de mensajes a enviar a través del canal de comunicaciones, por tanto, se reduce la utilización del canal.

- La topología del canal de comunicaciones puede variar. Existe la posibilidad de una topología de bus que requiere menos metros de cable que una topología de estrella.

- Finalmente, es posible implementar tolerancia a fallos en cada control inteligente, evitando que una avería local influya negativamente en el funcionamiento global del sistema.

Si nos referimos a sistema electrónico del vehículo como el conjunto de toda la electrónica contenida en él, un sistema guiado por cable es aquella parte orientada a un fin determinado. Los controles automáticos, que potencialmente pueden diseñarse como sistemas guiados por cable, son los llamados **controles de seguridad activa** destinados a la prevención de accidentes. No lo son los elementos de seguridad pasiva, destinados a mitigar las consecuencias del accidente. En la figura 4.1. se muestra un incremento exponencial de los controles de seguridad activa, utilizando el término *conducción autónoma* como máximo exponente (obtenido de [XByWire-DB-6/6-24]). Aunque no está claro lo que significará este término en el futuro, en el presente ya reconocemos una cantidad bastante significativa de controles de seguridad activa.

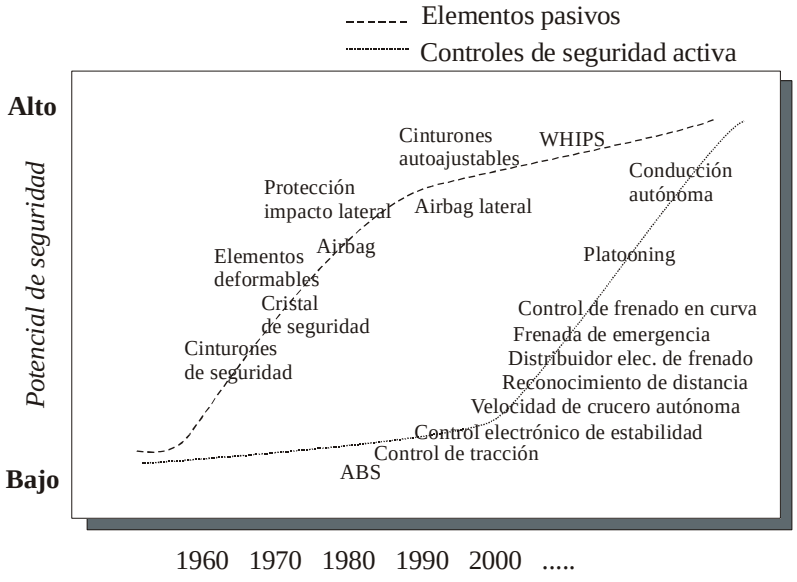


Figura 4.1. Crecimiento de los controles de seguridad en vehículos

Un sistema guiado por cable consta de varios elementos básicos:

- Sensores y actuadores asociados a dispositivos de entrada-salida.
- Los dispositivos de entrada-salida asociados a un elemento de control inteligente.

- Los elementos de control inteligente que, distribuidos en diferentes emplazamientos del vehículo, necesitan un canal de comunicaciones común.
- Un protocolo de comunicaciones que gestione el canal.

Todos estos elementos componen la arquitectura del sistema. Ya que los sistemas guiados por cable están orientados a aplicaciones de seguridad que deben considerarse críticas, la arquitectura que los soporte deberá ser en sí una arquitectura con alta confiabilidad, siendo de esperar que no se produzca nunca un estado que conlleve un riesgo para el usuario. Por tanto, en un estado normal de funcionamiento, libre de fallos, no se debería observar ningún modo de avería. En presencia de fallos, el sistema establece una cobertura para la detección de errores, sin que los mecanismos de tolerancia a fallos degraden el funcionamiento del dispositivo. En caso de producirse una avería que impida al sistema mantener su total funcionalidad, éste debería mantener una funcionalidad mínima (modo degradado de funcionamiento) del vehículo por tiempo suficiente como para llegar a un lugar de estacionamiento [XByWire-DB-6/6-24].

Por tanto, necesitamos una arquitectura cuyos atributos de fiabilidad y seguridad-inocuidad cumplan con lo establecido en el párrafo anterior. Las especificaciones deben contemplar, en la hipótesis de fallos, la no ocurrencia de averías catastróficas. Cada parte del sistema debe garantizar, individualmente, que no violará estas especificaciones ni en el dominio del tiempo ni del valor. Además, esta garantía también debe darse cuando las diferentes partes se unen para formar un único sistema. La integración del conjunto de elementos que lo componen no debe degradar o invalidar la confiabilidad que individualmente mantienen cada uno de dichos elementos. A esta propiedad se le denomina con el término inglés "***composability***" [Kopetz 2000] refiriéndose a la integración segura de todas las partes que componen el sistema.

4.2. División de Arquitecturas en Sistemas Distribuidos

Básicamente, podemos diferenciar dos tipos de arquitecturas: las arquitecturas de disparo por evento[44] y las de disparo por tiempo[45]. En una ET, la decisión de realizar un nuevo envío por parte de un nodo se toma ante la ocurrencia de un evento. En una TTA, se otorga un periodo de tiempo preestablecido a cada nodo, denominado ventana de transmisión, durante el cual sólo él puede y debe enviar algún mensaje. La asignación de tiempos es rotativa y todos los integrantes tienen conocimiento, al menos, de su ventana de transmisión. Por tanto, uno de los pilares de la arquitectura es la definición de una base de tiempos común que los sincronice.

En la figura 4.2. vemos un ejemplo de acceso al canal de comunicaciones en función de la arquitectura utilizada: ET o TTA. Existen tres nodos conectados al canal, nodos *A*, *B* y *C*. En el instante de tiempo t_{1ET} ocurre un evento en el nodo *A* que requiere del envío de una trama. En una ET, el nodo *A* accede al canal con un mínimo retardo, mientras que en una TTA el nodo *A* no puede comenzar el envío hasta la apertura de su ventana de transmisión en t_{2TTA}. En el instante de tiempo t_{2ET} ocurren dos eventos, uno en el nodo *B* y otro en el nodo *C*. En una ET es necesario establecer prioridades y/o resolución de colisiones ya que ambos nodos tratarán de acceder al canal al mismo tiempo. En una TTA, no existen prioridades ni se aceptan colisiones. En el caso de la figura, el nodo *C* será el primero en transmitir en t_{1TTA} debido a la proximidad del evento con la apertura de su ventana de transmisión, mientras que el nodo B debe esperar a t_{3TTA}, después del envío de la trama del nodo *A*. Según el esquema de la figura, en la arquitectura ET, la secuencia

[44] ET: Event-Triggered Architecture
[45] TTA: Time-Triggered Architecture

de acceso al canal sería: *Trama A*, *Trama B*, *Trama C* y *Trama A*. En la arquitectura TTA sería: *Trama C*, *Trama A*, *Trama B* y *Trama A*. No coinciden ni en secuencia ni en instantes de transmisión.

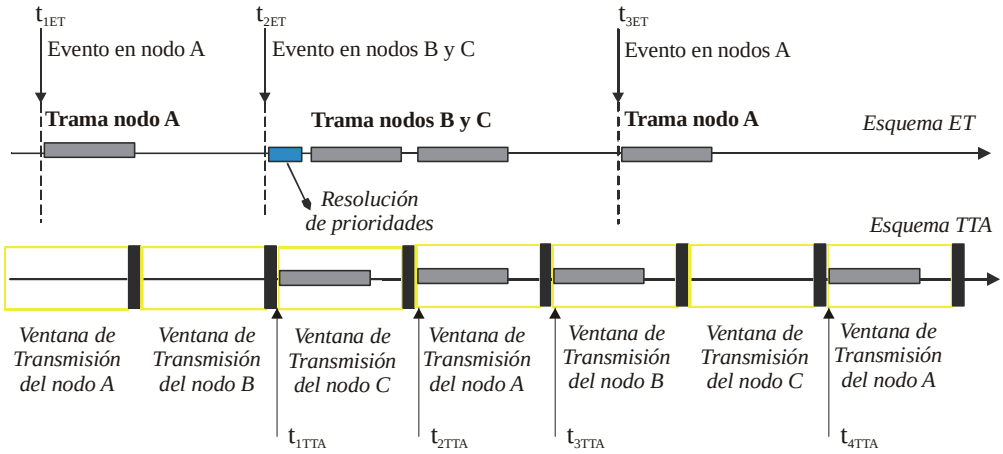


Figura 4.2. Ejemplo. Acceso al canal de comunicaciones según ET o TTA

Como vemos, una arquitectura ET es más flexible que una TTA, lo que supone mejores prestaciones en la utilización de sus recursos. Sin embargo, una TTA es una arquitectura más segura y fiable, lo que conlleva una mejor confiabilidad. Una arquitectura disparada por eventos desarrolla la detección de errores a nivel de nodo, pero no a nivel de sistema. Por el contrario, una arquitectura disparada por tiempos es capaz de detectar errores a dos niveles: a nivel de nodo y a nivel de sistema.

Modo de avería	***Arquitectura de disparo por evento***	***Arquitectura de disparo por tiempo***
Avería con parada	No define tiempo de latencia mínimo para reconocer la avería.	Asegura una latencia mínima y conocida tras la que todos los nodos perciben la avería. El sistema se mantiene consistente.
Averías bizantinas	No es consistente.	La integración del sistema (*composability*) le permite mantener su consistencia ante estas averías.
SOS	Se pueden dar en el dominio de valor.	Se pueden dar en el dominio de valor y de tiempo.
Suplantación de identidad	No define tiempo de latencia mínimo para reconocer la avería.	Provocaría una colisión. Asegura una latencia mínima de detección.
Transmisiones espurias	La consistencia del sistema depende de la implementación del protocolo.	Define normas de contención del error [Temple 1998] y dispositivos de comportamiento silencioso.
Pérdida de conexión	No define tiempo de latencia mínimo para reconocer la avería.	Asegura una latencia mínima de detección.

Tabla 4.1. Tratamiento de los modos de avería en sistemas distribuidos por ET y TTA

En la tabla 4.1. vemos el tratamiento que reciben los modos de avería[46] en sistemas distribuidos con ambas arquitecturas.

Mantener la consistencia del sistema ante cualquier circunstancia es fundamental para asegurar su funcionalidad, ya sea completa, o en el peor caso en modo degradado pero suficiente para cumplir con las normas de diseño establecidas por SAE[47] en su clase C. Por este motivo, la TTA es patrón de trabajo para las investigaciones realizadas por el FlexRay Consortium y el TTA Group que agrupan, entre otros, a fabricantes como Audi, Delphy Automotive System, Honeywell, PSA Peugeot-Citroen, TTTech, Volkswagen, BMW, Bosch Automotive Group, DaimlerChrysler, Ford, General Motors, Phylips Semiconductors, Texas Instruments, Cadence, etc.

Los puntos que la arquitectura TTA considera más importantes con respecto a las normas de diseño de **SAE** para la clase **C** son [Hedenetz y Belschner 1998] los siguientes:

- Una comunicación determinista que asegure tiempos de latencia preestablecidos.
- La integración del conjunto de elementos que componen el sistema, o la incorporación de nuevos elementos, no debe degradar o invalidar ninguna propiedad que individualmente mantengan cada uno de dichos elementos. Esta norma hace referencia a la integración segura del sistema[48].
- Debe dar soporte para elementos replicados.
- Se recomienda la distribución de los elementos de control inteligente replicados, evitando fallos en modo común (un mismo fallo que afecte a los dos elementos replicados). Este punto también hace referencia a las averías por proximidad, donde se tiene en cuenta que un impacto localizado en un punto del vehículo no debe afectar a ambas réplicas.
- Debe asegurar que ningún dispositivo con acceso al canal de comunicaciones genere transmisiones espurias monopolizando el canal, y en cualquier caso, debe garantizarse la continuidad del servicio ante este tipo de transmisiones.
- Debe existir una sincronización entre los nodos y una distribución de la carga. Por lo tanto, es requisito una base de tiempos común.
- Debe asegurar la consistencia del sistema. Todos los nodos conocen el estado de los demás nodos que comparten el canal. A cada elemento que comparte el canal se le denomina miembro del sistema. Los posibles estados de un miembro del sistemas son: activo, pasivo o desconectado. El mecanismo que proporciona un conocimiento del estado de todos los miembros se llama servicio de pertenencia. Se asegura una latencia mínima y conocida de reconocimiento de una nueva situación, por ejemplo, el cambio de estado de un miembro.
- Se recomienda soporte para la interconexión de redes.

4.3. Conceptos Básicos de la Arquitectura TTA

Existen muchas referencias donde se describe la arquitectura TTA [ver Vmars-TUWien], pero es en [Kopetz y Bauer 2002] donde se describe con más detalle el camino trazado en la investigación, desde los primeros trabajos orientados a aplicaciones críticas, hasta el actual concepto de la Arquitectura de Disparo por Tiempo TTA. Este trabajo está

[46] Definidos en el capítulo 2
[47] Society of Automotive Engineers
[48] Composability

subvencionado, entre otros, por el proyecto FIT – IST 1999-10748.

Los primeros trabajos de investigación sobre arquitecturas para sistemas distribuidos con alta confiabilidad aparecieron hace más de 30 años. El STAR [Avizienis *et at*. 1971] es un ejemplo práctico de los primeros trabajos publicados en este área. También lo son los proyectos de investigación SIFT [Wensley *et al*. 1978] y FTMP [Hopkins *et al*. 1978]. Diez años después, y como continuación de las investigaciones realizas en el SIFT y FTMP, aparecen las propuestas FTPP [Lala y Alger 1988] y MAFT [Kieckhafer *et al*. 1988], bases del sistema de control de vuelo AIRBUS [Traverse 1988]. En 1993 se publicó la primera referencia sobre SAFEbus [Hoyme y Driscoll 1993], arquitectura utilizada en el Boeing 777. Otros trabajos de interés sobre los principios básicos que debería sostener cualquier arquitectura con alta confiabilidad, son los presentados en [Laprie 1992], [Lala y Harper 1994] y [Avizienis 1997]. En Europa, el proyecto ESPRIT DELTA 4 [Kanoun *et al*. 1990] se centra en mecanismos de recuperación ante averías en sistemas distribuidos.

Con respecto a la arquitectura TTA, el primer proyecto de investigación es el proyecto MARS ("MAinteinable Real-Time System") [Kopetz y Merker 1985], dirigido por la Universidad de Viena en 1979. Este proyecto significó la colaboración de varias universidades, así como un gran número de informes entre 1982 y 1985. Uno de los campos sobre el que más se trabajó fue en la definición de un algoritmo de sincronización de los relojes locales internos que estableciera una base de tiempos común. Con el proyecto Europeo PDCS ("Predictably Dependable Computing Systems") se desarrolló el primer prototipo basado en TTA: el TTP^{TM}/C, que incorporaba este algoritmo de sincronización [Kopetz y Grünsteidl 1993].

Sin embargo, a principios de los años 90, la industria todavía no acepta la necesidad de desarrollar e incorporar esta nueva arquitectura, diferente de una típicamente centralizada. La primera colaboración entre industria y academia es con DaimlerChrysler en 1995. En 1998 se desarrolla el primer integrado que incorpora el protocolo de comunicaciones TTP basado en TTA bajo el proyecto ESPRIT TTA. En este mismo año se funda la empresa High-Tech Spinoff, que luego se convertirá en TTTech, para la introducción en el mercado de TTP^{TM}/C.

Los últimos proyectos relacionados con la validación y mejora del TTP son los proyectos Europeos IST **FIT**, NEXT TTA y SETTA. En Estados Unidos, los proyectos DARPA, MOBIES y NEST también han contribuido a la validación de la arquitectura TTA.

Tras TTP aparecen otros protocolos basados en TTA, como son TTCAN [Führer *et al*. 2002 y Hartwich *et al*. 2002], FlexRay [FlexRay 2002 y Rushby 2001 (a)] y más recientemente FTT-CAN [Calha y Fonseca 2002, Ferreira *et al*. 2002 (a) y (b)], de los que se hablará en el apartado 4.4.

4.3.1. Estructura

Los elementos constructores de la arquitectura son los siguientes:

- **El canal de comunicaciones**: La arquitectura TTA se basa en el principio de la redundancia física. Por tanto, el canal de comunicaciones es un canal replicado: dos líneas físicamente independientes. Un mensaje se puede transmitir por un canal o por los dos al mismo tiempo. Este segundo caso es recomendable para mensajes críticos, mientras que la no

replicación mejora el ancho de banda del canal.

- **Los nodos**: Son unidades inteligentes de control electrónico. Si detectan que no pueden mantener correctamente su funcionalidad dejan de transmitir, volviendo a reintegrarse en el canal de comunicaciones cuando se recuperen. Los nodos se encuentran replicados formando una Unidad Tolerante a Fallos o FTU. Ambos nodos tienen asignado un periodo de transmisión distinto, los dos tienen acceso independiente a ambos canales de comunicación y se mantienen en estado activo recibiendo y enviando mensajes, pero sin embargo, ejecutan la misma carga. Por tanto, en un estado libre de fallos, todos los mensajes se reciben duplicados, uno de cada réplica. Una estrategia básica de selección es la lectura aleatoria de uno de los dos mensajes. En presencia de fallos, si aseguramos que no se dan fallos en modo común, aunque una de las réplicas deje de transmitir (comportamiento silencioso), la otra réplica podrá enviar el mensaje. La figura 4.3. muestra las diferencias entre una conexión replicada y una TMR[49] que también ha sido frecuentemente utilizada en sistemas distribuidos [Siewiorek y Swarz 1992, Mitra y McCluskey 2001, Elnozahy *et al*. 2002, entre otros].

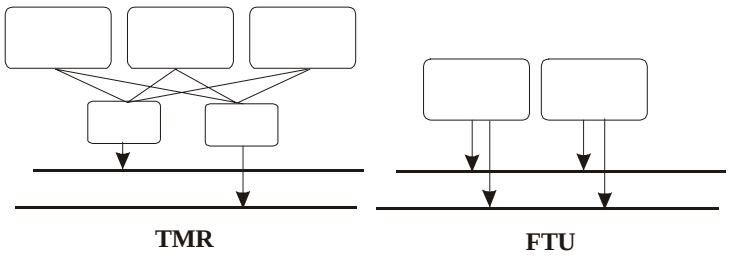


Figura 4.3. Diferencias entre un TMR y una FTU

- **El sistema de comunicaciones**: Un nivel más de estructuración. Se compone de elementos electrónicos pasivos y elementos activos involucrados en el protocolo de comunicaciones.

- **El guardián del bus**: Es un elemento con alimentación y reloj independientes. Es el encargado de mantener aislada la salida del nodo hacia el canal mientras no sea su turno de acceso. Solamente habilita físicamente dicha salida durante la ventana de transmisión. Este elemento debe garantizar la no ocurrencia de transmisiones espurias de un nodo durante la ventana de transmisión asignada a otro miembro del sistema. Existen dos posibles topologías de conexión para TTA: **topología de bus** y **topología de estrella**. La *topología de bus* implica un guardián local en cada nodo (figura 4.4.), mientras que la *topología de estrella* implica dos guardianes centrales, uno por cada canal (figura 4.5.).

- **El nivel de aplicación**: Incluye no sólo la aplicación, sino también el sistema operativo que gestiona el servicio de interconexión con el sistema de comunicaciones.

[49] Tripple Modular Redundant

- "Firewalls" o **barreras de contención**: Se utilizan para entrelazar niveles. Incluyen tanto un aislamiento físico, para evitar la propagación de fallos, como algoritmos de gestión. La memoria de mensajes se considera una barrera de contención entre el nivel de aplicación y el sistema de comunicaciones, mientras que el guardián del bus se considera otra barrera de contención entre el sistema de comunicaciones y el propio canal.

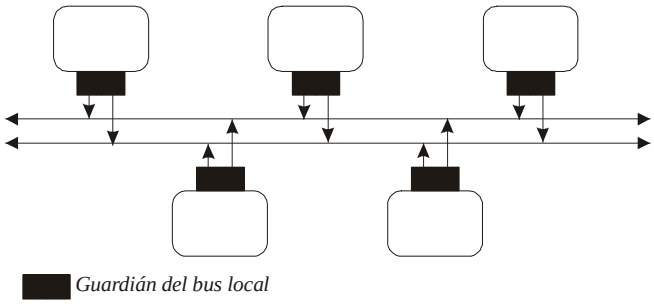


Figura 4.4. Topología de Bus

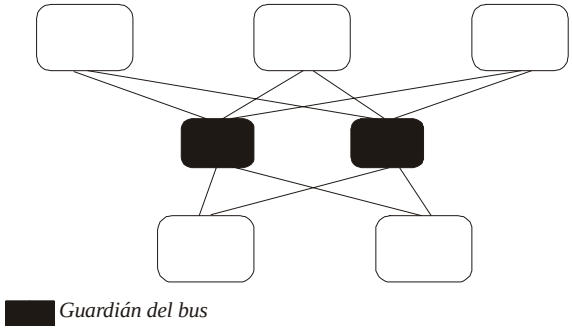


Figura 4.5. Topología de Estrella

4.3.2. El sistema de comunicaciones

Entre sus servicios se encuentran:

- **Acceso al canal de comunicaciones sin arbitraje**: Mantiene una tabla predefinida con la división temporal del canal, es por tanto una transmisión determinista. Todos los nodos tienen asignado un período o ventana de transmisión con un orden preestablecido. La suma de todos los períodos se denomina una vuelta completa de acceso o **TDMA**[50]. Se pueden definir diferentes TDMA. En cada uno de ellos, cada nodo envía un tipo de trama distinta. Como está obligado a transmitir siempre, cuando no tiene datos que transmitir emite una trama de sincronización. El conjunto de todos los TDMA definidos se llama **ciclo completo de transmisiones del sistema** (por ejemplo, ver figura 4.6.).

[50] Time Division Media Access

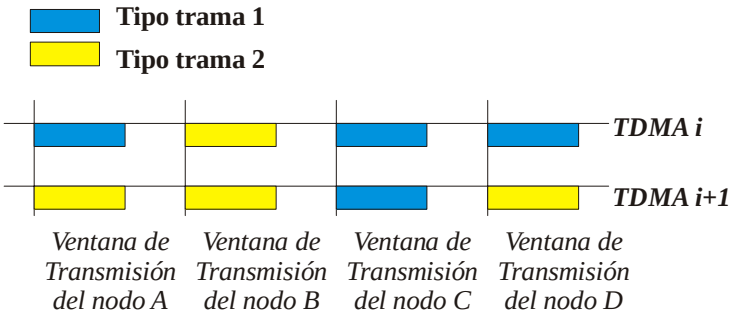


Figura 4.6. Ciclo completo de transmisiones de un sistema con dos TDMA

- **Base de tiempos común**: Algoritmo de sincronización permanente. Puede ser externa o interna. La ***sincronización externa*** necesita un nodo o miembro maestro cuya base de tiempos es referencia para el resto de los miembros del sistema. La ***sincronización interna*** se alcanza mediante un algoritmo que calcula la desviación temporal entre el momento en el que se espera una nueva trama y el momento en el que se recibe realmente, pudiendo ser desestimadas aquellas desviaciones que exceden (por exceso o defecto) un margen de desviación. La media de las desviaciones calculada durante un ciclo sirve como factor de corrección de la base de tiempos local.

- **Servicio de pertenencia**: En cada nodo se mantiene un **vector de pertenencia** que indica, con un bit o *flag*, si el nodo al que representa dicho bit se encuentra activo, o por el contrario, se ha desconectado debido a un error. Este vector representa la visión del nodo sobre el sistema. Cada vez que se transmite una trama, el nodo transmisor incluye en ella el vector de pertenencia. Todos los demás nodos comprueban si su vector coincide con el recibido. El algoritmo que define el servicio de pertenencia incluye los pasos que ha de llevar a cabo un receptor en caso de no coincidencia.

- **Reconocimiento implícito**: No existe campo de reconocimiento en las tramas. Cada nodo mantiene tres contadores: número de tramas válidas, número de tramas inválidas recibidas y silencios percibidos en un TDMA. Los valores de los contadores se ponen a cero cuando el nodo envía una trama y se comprueban en su siguiente ventana de transmisión, justo antes de un nuevo envío. El algoritmo comprueba que el contador de tramas válidas sea mayor que el contador de tramas inválidas o silencios percibidos. En caso contrario, asume un error y pasa a modo pasivo, permaneciendo en silencio. Una trama cuyo vector de pertenencia difiere del vector de pertenencia del receptor se considera como inválida. Por tanto, un nodo puede ser expulsado del canal si existe una mayoría que marca el *flag* de dicho nodo como inválido en su vector de pertenencia. Todos los nodos participan tanto en el servicio de pertenencia como en el de reconocimiento, independientemente de los datos contenidos en la trama.

- **Cambio de modo de operación**: Es posible operar en varios modos. Para cada modo se define una tabla de acceso al bus distinta, todas ellas almacenadas en memoria. Debido a que en una tabla de acceso al bus se contemplan las tramas a transmitir por todos y cada uno de los nodos del

sistema, un cambio de modo debe ser una acción consensuada, donde se coordina el cambio del esquema de acceso por el definido en la tabla correspondiente al modo seleccionado.

- **Encapsulamiento de la trama**: Define el formato de las tramas. No existe identificador destino o de trama ya que ésta se identifica por el momento en el que se ha recibido. La trama se recibe por todos los nodos, todos comprueban su validez y actualizan el vector de pertenencia.

- **Codificación de la trama**: Codifica la salida según un código conocido.

- **Códigos de protección**: La trama incluye un campo para códigos de detección de errores, típicamente un CRC[51]. El algoritmo de cálculo utiliza los bits que conforman la trama además de una serie de registros internos cuyo contenido no se incluye explícitamente en ningún campo de la trama.

- **Verificación de la trama**: Comprueba el instante de recepción de la trama (o no recepción en caso de detectar silencio en el bus), el encapsulamiento, la codificación, el código de protección, el vector de pertenencia y el modo de operación.

- **Gestión del los mensajes**: El sistema de comunicaciones genera tramas de transmisión en función de los valores almacenados en un área de memoria específica (memoria de mensajes). A su vez, cuando recibe un nuevo mensaje, el valor contenido en el campo de datos también es almacenado en la memoria de mensajes. Por tanto, dicha área de memoria debe ser accesible tanto desde el sistema de comunicaciones como desde el nivel de aplicación. Esta memoria específica se denomina CNI[52] y actúa como una memoria de doble puerto. Los algoritmos de escritura y lectura, así como los servicios que controlan la validez de los datos contenidos en dicha memoria, se agrupan en los mecanismos de gestión de la CNI.

- **Interconexión con el nivel de aplicación**: El sistema de comunicaciones es el responsable de la detección y contención de errores detectados en alguno de los servicios anteriores. Es capaz de tomar decisiones sobre su conexión o desconexión al canal. Sin embargo, tales decisiones deben ser comunicadas de inmediato al nivel superior o nivel de aplicación. Esta comunicación entre niveles se define como *servicio de interconexión con el nivel de aplicación.*

4.4. Protocolos de Comunicación basados en TTA

Destacamos los siguientes protocolos de comunicaciones basados en la arquitectura TTA: TTP, FlexRay, TTCAN y FTT-CAN.

TTP – Orientado a la seguridad, es un protocolo estrictamente estático. No sólo tiene un esquema predefinido de acceso al bus, sino que también las tareas propias del nivel de aplicación se realizan mediante un orden conocido. Este orden marca el instante de inicio

[51] Código Cíclico Redundante
[52] Controller Network Interface

y el tiempo máximo de ejecución de la tarea.

Según la hipótesis de fallos, el protocolo es capaz de tolerar dos eventos. Primero, una avería o desconexión de algún elemento activo (controlador o guardián de bus), siendo capaz de mantener la consistencia ante cualquier modo de avería. Segundo, un fallo en algún elemento electrónico pasivo o en el canal de comunicaciones.

El mínimo número de nodos que deben permanecer activos para asegurar una correcta sincronización de la base de tiempos es de cuatro. Con menos nodos, el sistema funciona en modo degradado.

FlexRay – Está orientado a la flexibilidad en vez de a la seguridad. Combina dos segmentos, uno para transmisión estática, de disparo por tiempo, y otro para transmisión dinámica, de disparo por evento. El tamaño de ambos segmentos es configurable. Durante la transmisión estática, utiliza un esquema predefinido de acceso al bus. Durante la transmisión dinámica, el acceso al bus se realiza por prioridades.

Según la hipótesis de fallos, el protocolo es capaz de tolerar una avería arbitraria en algún elemento activo, pero sólo está protegido el algoritmo de sincronización. No implementa ni servicio de pertenencia ni reconocimiento implícito.

El número mínimo de nodos que deben permanecer activos para asegurar la correcta sincronización de la base de tiempos debe ser superior a un tercio de los nodos que comparten el segmento estático.

TTCAN – Es una adaptación del estándar ISO 11898 CAN (“Control Area Network”) versión 2.0. Fija una matriz de acceso al bus, donde cada ventana de transmisión tiene un intervalo de tiempo predefinido. Existen dos tipos de ventanas de transmisión: fijas, donde el nodo que debe transmitir es siempre el mismo, o de arbitraje, donde todos los nodos pueden tratar de ganar el bus utilizando su identificador de trama (el identificador establece su prioridad).

Según su hipótesis de fallos, el protocolo sólo es capaz de tolerar transmisiones espurias coincidentes con ventanas de transmisión fijas. No tolera averías arbitrarias. Implementa códigos de protección del mensaje. Sin embargo, no se podrá retransmitir una trama detectada como errónea hasta la siguiente ventana de transmisión habilitada para el nodo. La sincronización de la base de tiempos es externa. Por tanto, al menos debe existir un nodo maestro que envíe mensajes de sincronización al resto de los nodos conectados.

FTT-CAN – (“Flexible Time-Triggered CAN”) Es una variación o mejora de TTCAN. El protocolo de comunicaciones se encuentra centralizado en un nodo maestro. La matriz de acceso al bus es variable, no fija. Es el nodo maestro el que reajusta dicha matriz en cada ciclo básico de acceso.

Según su hipótesis de fallos, el protocolo es capaz de tolerar transmisiones espurias coincidentes con ventanas de transmisión fijas y desconexiones de elementos activos. Implementa códigos de protección del mensaje y en este caso, la trama podrá ser retransmitida siempre que no exceda el periodo asignado a la ventana de transmisión. La sincronización de la base de tiempos es externa. Igual que TTCAN, al menos debe existir un nodo maestro que envíe mensajes de sincronización al resto de los nodos conectados.

4.4.1. Comparación entre diferentes protocolos de comunicaciones

Siendo protocolos de comunicación basados en la misma arquitectura, existen diferencias importantes que pueden afectar a la confiabilidad del sistema sobre el que se implementen. La tabla 4.2. muestra algunas diferencias entre los cuatro protocolos.

	TTP	***FlexRay***	***TTCAN***	***FTT-CAN***
Velocidad de transmisión	5 Mbit/s	10 Mbit/s	1 Mbit/s	1 Mbit/s
Máximo tamaño de trama	240 bytes	246 bytes	8 bytes	8 bytes
Topología	Bus y estrella	Estrella	Bus	Bus
Canal replicado	Si	Si	No	Si
Guardián del bus	Local e independiente para bus. Central para estrella (uno por canal)	Local pero no independiente para bus. Central para estrella (uno por canal)	No	Local
Tipo de sincronización	Interna	Interna	Externa	Externa
Acceso al bus	Estático	Dos segmentos de tamaño fijo, uno estático y otro con arbitraje	División de tiempos estática. Ventanas de transmisión con o sin arbitraje	División de tiempos dinámica controlada por el maestro. Ventanas de transmisión con o sin arbitraje
Diferencias básicas en los servicios	Servicio de pertenencia y reconocimiento implícito	Ninguno	Reconocimiento de trama	Reconocimiento de trama, servicio de pertenencia para nodos maestro, el maestro actúa de monitor de bus
Orientación industrial	Automoción y Aviónica	Automoción	Automoción	Control industrial
Integración segura del sistema[53]	Sí	No	No	No

Tabla 4.2. Comparación de protocolos basados en TTA

TTP – Se puede configurar como topología de bus o de estrella con canal replicado. Esta última configuración ofrece mejores resultados de fiabilidad. El guardián del bus tiene reloj y alimentación independientes.

La sincronización es interna, pero admite sincronización externa en caso de combinar varios sistemas, donde uno haría de maestro y el resto de esclavos.

No necesita identificador de trama o mensaje, por lo tanto, el formato de la trama se simplifica. El tipo y tamaño de una trama son constantes predefinidas. Sólo varían los valores contenidos en algunos campos como son los datos, el registro de estado, CRC,

[53] Composability

petición de cambio de modo, etc.

FlexRay – Se configura como una topología de estrella con canal replicado. Si se implementa con un guardián del bus local, éste no tiene ni reloj ni alimentación independiente.

La sincronización es interna, aunque se admite también sincronización externa. La sincronización sólo se realiza durante el segmento estático.

Al utilizar el mismo formato de trama en ambos segmentos, necesita introducir un identificador de trama. Por tanto, aunque el esquema de acceso al bus sea estático, el tipo de trama que transmite un nodo en el periodo asignado puede ser variable.

En el segmento dinámico necesita resolución de colisiones. En realidad, el protocolo evita colisiones asignando un retardo de acceso al bus distinto a cada nodo, según su prioridad. El retardo de acceso es el tiempo que tiene que esperar desde que el canal está libre hasta que puede comenzar una transmisión.

El protocolo no define ningún tipo de reconocimiento, ni implícito ni explícito.

TTCAN – Se configura con una topología de bus sin canal replicado. Se utilizan controladores estándar de CAN, por tanto, no existe guardián del bus y la definición de la matriz de acceso al canal de comunicaciones se realiza a nivel de aplicación. Cada nodo sólo conoce su o sus ventanas de transmisión, nunca la matriz de acceso completa.

La recepción y aceptación de una trama se gestiona, al igual que CAN, según el identificador de trama. Por tanto no se implementa el servicio de pertenencia en el que es necesario tener a priori un conocimiento completo de la matriz o patrón de acceso al bus.

La sincronización es externa, existe un nodo maestro que transmite un mensaje de referencia al inicio de cada ciclo básico. Dicho mensaje sirve para ejecutar el algoritmo de sincronización local en cada nodo. La transmisión del mensaje de referencia marca el inicio de un ciclo básico, con un patrón de acceso determinado. Un ciclo completo de transmisiones consiste en el conjunto de todos los diferentes ciclos básicos.

El formato de trama no varía respecto al definido en CAN. La única novedad es que si un nodo intenta ganar el bus en una ventana de arbitraje y no lo consigue, no lo intenta de nuevo (utiliza el mecanismos de *single-shot*). Así se evita que colisione con la siguiente ventana de transmisión que, con bastante probabilidad, será fija.

Los servicios de TTCAN son los propios del estándar.

FTT-CAN – Incorpora una topología de bus con canal replicado. A los controladores de CAN añade un guardián del bus.

Centraliza la mayor parte de las decisiones en un maestro de bus dedicado a las tareas de ajuste de la matriz de acceso y de supervisión del estado de las comunicaciones. El maestro envía un mensaje de referencia al inicio de cada ciclo básico. Además de utilizarse para tareas de sincronización, el mensaje de referencia indica la nueva distribución de los acceso al bus para ese ciclo. Este ajuste continuado de la matriz de acceso es el que mejora la flexibilidad del protocolo.

Pueden coexistir varios maestros de bus, pero sólo uno accede al canal. El resto se mantiene en estado pasivo, leyendo las tramas transferidas y ejecutando los algoritmos

correspondientes, pero sin envío de trama alguna. Cuando un maestro detecta un error interno, tiene un comportamiento silencioso, dejando libre su periodo de transmisión para otro maestro.

Una de las funciones del maestro es comprobar que cada nodo transmite durante la ventana fija asignada a dicho nodo. Por tanto, la transmisión durante estas ventanas es obligada. Si el maestro detecta una *no-transmisión*, ajusta la matriz de acceso para el siguiente ciclo básico.

Finalmente, el maestro activo implementa un servicio de pertenencia, pero restringido a los nodos maestros, mediante el envío de una trama remota. La contestación a esta trama indica si los nodos maestros en estado pasivo siguen manteniendo sus funciones básicas.

Aunque FlexRay es el algoritmo más parecido a TTP que podemos encontrar, no asegura la integración del sistema[11] puesto que no define ningún tipo de servicio. Este protocolo podría verse como una base, cuya definición se limita al acceso al bus, sobre la que se debe construir el verdadero protocolo (por ejemplo el Byteflight de BMW [Byteflight-spec 2001]). Por otra parte, TTCAN incorpora un acceso de disparo por tiempo desarrollado a nivel de aplicación, pero mantiene sus características de sistema de disparo por eventos. FTT-CAN mejora la consistencia del sistema introduciendo al maestro como monitor del bus. Sin embargo, esta consistencia sólo se asegura durante las ventanas de transmisión fijas. Un error durante una ventaja de arbitraje puede generar una inconsistencia cuya latencia de detección, por parte del maestro, no se encuentra definida en las especificaciones.

4.4.2. El controlador de comunicaciones TTP/C

El controlador TTP^{TM}/C, la C de clase C de SAE, se diseña bajo el protocolo de comunicaciones TTP basado en una arquitectura TTA. La figura 4.7. muestra una configuración básica de cuatro nodos con topología de bus. Todos los nodos tienen acceso directo al canal de comunicaciones y se mantienen constantemente en estado activo, salvo desconexión por avería, tanto si se encuentran replicados como si no.

El controlador de comunicaciones que alberga el protocolo TTP es el objetivo de la validación experimental llevada a cabo en el presente trabajo de tesis. Existe una previsión de implantación de sistemas con controladores TTP/C en vehículos comerciales aproximadamente sobre el 2006. Por tanto, se trata de un producto comercial, basado en un concepto innovador en el campo de la automoción.

Nuestro prototipo está formado por cuatro nodos. Cada uno consta de cuatro partes básicas (figura 4.8.): el controlador de comunicaciones, la interfaz de comunicaciones o CNI, el microcontrolador que alberga la aplicación[54] y los dispositivos de entrada-salida.

El controlador de comunicaciones TTP^{TM}/C se comporta como un conjunto de elementos con cierta relación. Estos elementos son la **Unidad de Control del Protocolo**, la **Interfaz de Comunicaciones**, el **Descriptor de Mensajes** y el **Guardián del Bus** (figura 4.9.), como elementos activos. Como elementos pasivos encontramos los canales de comunicaciones, los componentes pasivos y los tres osciladores para el guardián del bus, el controlador de comunicaciones y para el microcontrolador que soporta la aplicación.

[54] Denominado “host”

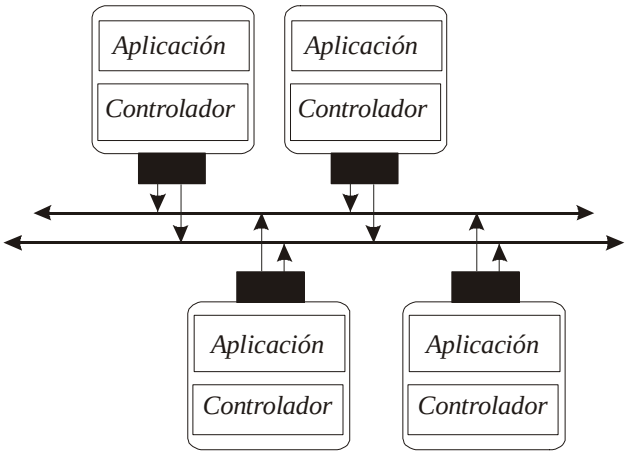


Figura 4.7. Configuración básica de bus con cuatro nodos

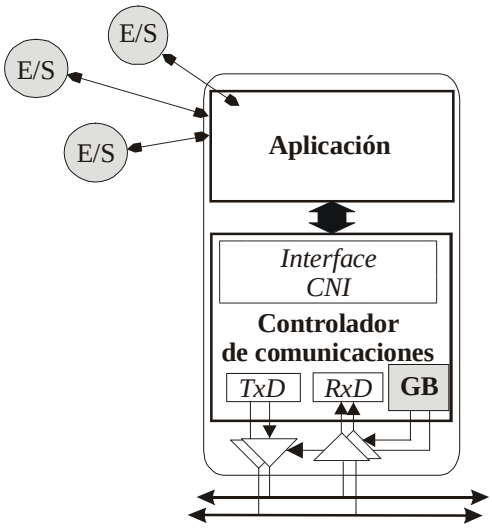


Figura 4.8. Detalle de un nodo TTP/C

4.4.2.1. La unidad de control del protocolo (PCU)

Es el núcleo del controlador. En las versiones C1 y C2 consiste en una unidad de procesamiento con búsqueda, decodificación y ejecución de las instrucciones pertenecientes al llamado microcódigo. En la versión C2S la estructura del controlador es totalmente hardware.

La unidad de control tiene asociados varios módulos, como son la unidad de control del tiempo, la unidad de cálculo y comparación del CRC, la unidad de test, la pila de registros internos y el tratamiento de entradas como la señal de *reset* y la entrada del oscilador.

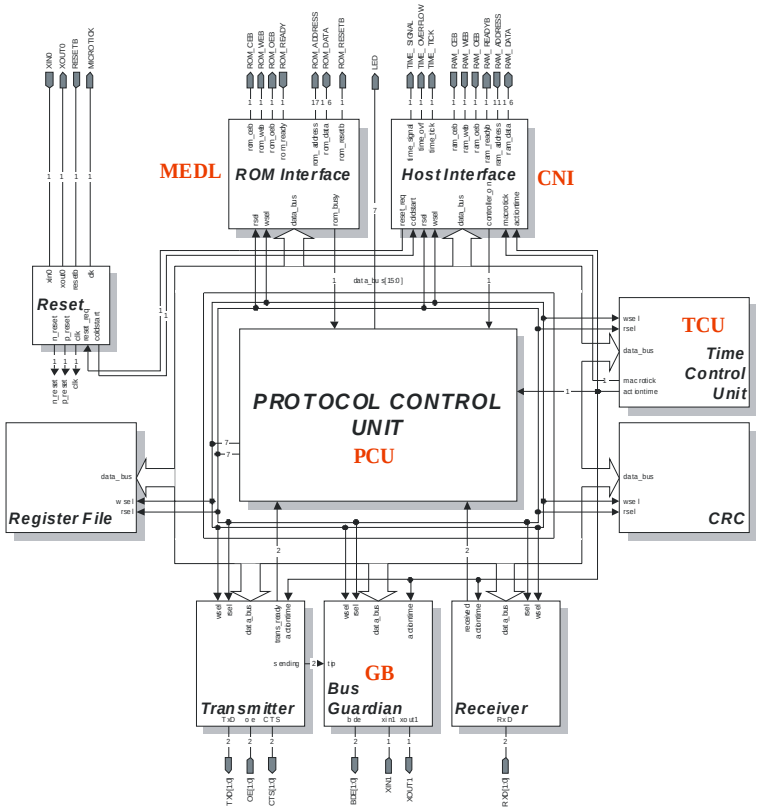


Figura 4.9. Representación por bloques de un nodo TTP/C

4.4.2.2. La interfaz de comunicaciones (CNI)

Es una memoria de doble puerto a la que tiene acceso tanto el controlador de comunicaciones (donde se encuentra integrada) como la aplicación. Se divide en dos áreas, el **área de mensajes** y el **área de registros de control y estado**.

El área de mensajes consiste en un conjunto de registros de 16 bits donde se almacenan los valores a enviar y valores recibidos durante el último ciclo completo de transmisiones del sistema. Cada variable que pueda formar parte de una trama tiene asignadas una o varias direcciones conocidas. El número de direcciones depende del tamaño de la variable. Si una variable se envía por ambos canales, el valor de dicha variable se almacena por duplicado de forma que el controlador construya cada trama utilizando posiciones de memoria diferentes. El tamaño total del área permite alojar hasta 240 bytes por cada canal.

El área de registros de control y estado consiste en un conjunto de registros de 16 bits que contienen información relevante sobre el estado de las comunicaciones y sobre la intercomunicación del controlador con la aplicación. Algunos de estos registros contienen información sobre el orden de acceso al bus, versión del protocolo, versión de la aplicación, etc. Los más destacables en importancia son los siguientes:

El registro de estado[55]: Ocupa de dos a ocho posiciones de memoria, dependiendo de la versión. Es importante puesto que se utiliza en el cálculo del CRC. Se descompone en un campo que indica el factor de corrección de la base de tiempos, otro campo que indica la ventana de transmisión actual y, finalmente, otro campo

[55] C-state

con el **vector de pertenencia** donde cada bit representa en binario el estado de uno de los nodos del sistema: miembro activo del sistema o desconectado.

Registro de habilitación de interrupciones: Ante la detección de un error, el controlador puede informar a la aplicación de dicha detección, siempre que se encuentre habilitada la interrupción correspondiente. Existe una señal conectada a una interrupción externa del microcontrolador. Sólo existe esta señal para indicar una detección, obligando al microcontrolador a acceder a otro registro de memoria donde encontrará información más detallada del evento: el Registro de Estado de las Interrupciones.

Testigo de vida del controlador: Mientras que el controlador de comunicaciones esté vivo debe actualizar este registro según cierto algoritmo. Existe una tarea a nivel de aplicación que lee dicho registro cada TDMA, asegurándose así de que el nodo aún sigue conectado al canal.

Testigo de vida de la aplicación: Al igual que antes, sirve para que el controlador se asegure de la funcionalidad de la aplicación, o si por el contrario debe realizar una desconexión voluntaria del canal.

Registros de tiempos: Los utiliza el sistema operativo para almacenar el tiempo de ejecución máximo de la tarea en proceso.

4.4.2.3. El descriptor de mensajes (MEDL)

Es una memoria estructurada para almacenar la tabla de acceso al bus. Identifica el periodo de acceso de cada nodo, tamaño de la ventana de transmisión, duración total de cada periodo y tiempo de espera entre dos transmisiones consecutivas. Tiene capacidad para almacenar varios modos de funcionamiento (cada modo implica una tabla), y con varias frecuencias de transmisión.

En la versión C1, el descriptor de mensajes se encuentra en una memoria externa al controlador. Para proteger la lectura de los datos desde el descriptor al controlador se utiliza un CRC por bloques de direcciones de grado 16: $x^{16} + x^{12} + x^{11} + x^{8} + x^{5} + x^{4} + 1$. En la versión C2, el descriptor se integra en el controlador aunque mantiene el mismo mecanismo de detección de errores.

4.4.2.4. El guardián del bus (GB)

En una topología de bus, existe un único guardián para ambos canales. A diferencia de la recepción, que permanece habilitada constantemente, el guardián del bus es el encargado de habilitar la transmisión sólo durante el periodo asignado en el TDMA.

En una topología de estrella existen dos guardianes centrales, uno por canal. En este modo, el guardián tiene tantas entradas y tantas salidas, físicamente independientes, como nodos conectados. En cada ventana de transmisión tan solo habilita una de sus entradas y todas sus salidas. La ventaja de esta topología es que el guardián puede ser inteligente, implementando algoritmos de control adicionales. El gran inconveniente es el número de cables que requiere y la dependencia de todo el sistema al correcto funcionamiento de, al menos, un guardián del bus.

Durante el tiempo de espera entre dos transmisiones consecutivas, ni la recepción ni la transmisión se encuentran habilitadas. A este periodo se le denomina **IFG** (“InterFrame

Gap"), es de tamaño fijo y se aprovecha para la actualización de los registros de control y estado.

4.5. Representatividad de los Fallos a Nivel de Pin en el TTP/C

El principal objetivo de la inyección física llevada a cabo sobre el TTP™/C es la de validar el comportamiento del controlador de comunicaciones en presencia de fallos, asegurando que no existe ninguna violación de la hipótesis de fallos ni en el dominio del tiempo ni en el de valor.

Cada tipo de inyección física valida el comportamiento del sistema ante un conjunto de fallos de determinadas características. Por eso es más interesante realizar diferentes tipos de inyección que limitarnos a una sola técnica. Uno de los posibles tipos de inyección física es la inyección a nivel de pin, basada en la idea de perturbar los circuitos integrados mediante la introducción de fallos a través de los pines o patillas del integrado, modelando tanto fallos externos al integrado como internos [Gil, Blanc y Serrano 2003]. Son ejemplo de fallos externos los producidos por ruido radiado que se transmite a través de los pines, fallos a nivel de soldadura o problemas en las líneas de conexión entre dispositivos, fallos debidos a los elementos pasivos (osciladores, conmutación de relés, *drivers* de transmisión, líneas de bus, terminadores, etc.), fallos propagados desde otros dispositivos, o inversiones en celdas de memoria o registros debidas a radiación. Los fallos internos son aquellos que, debido a errores del propio dispositivo, afectan a señales de control o a las líneas de transmisión de datos. Este tipo de inyección nos va a servir para validar la eficiencia de las barreras de contención de errores en el TTP/C: la interfaz de comunicaciones CNI y el guardián del bus.

El efecto más dañino de un fallo en el TTP/C es el de generar un error que se propague fuera de una zona de contención. La propagación del error fuera de esta zona violaría las especificaciones de diseño del protocolo de comunicaciones. En la figura 4.10. se delimitan las zonas de contención internas del controlador de comunicaciones TTP/C. La interfaz de comunicaciones CNI (memoria SRAM) hace las veces de barrera de contención entre el nivel de aplicación y el sistema de comunicaciones. El guardián del bus hace las veces de barrera de contención entre un nodo y el canal de comunicaciones.

Se va a dividir el análisis de los efectos de los fallos en dos partes. Por un lado se verán las diferentes alteraciones que se producen en la señal emitida, y luego se desarrollará un segundo enfoque en donde la señal es parte de un conjunto cuyo sincronismo determina un comportamiento concreto.

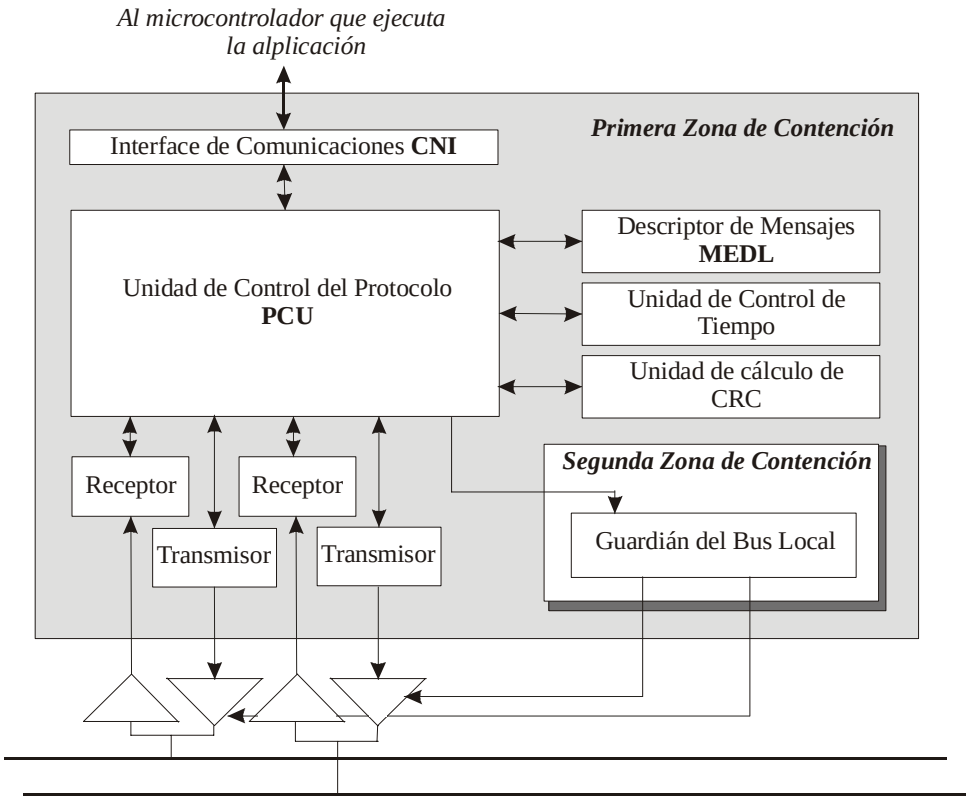


Figura 4.10. Zonas de contención y barreras del controlador TTP/C

4.5.1. Efecto de la inyección de fallos sobre un pin de entrada–salida

El forzado de un pin de entrada-salida a un valor determinado (inyección del fallo) puede provocar diferentes efectos según la señal inyectada. Los efectos pueden agruparse esquemáticamente en: la **generación de un pulso inexistente**, la **eliminación de un pulso existente** o la **variación de los márgenes temporales del pulso**. Para su aplicación a las líneas de entrada y salida del controlador TTP/C, se tendrán presentes dos consideraciones:

- Se dispone de una herramienta capaz de inyectar fallos con una duración inferior al pulso más pequeño que el controlador TTP/C pueda generar ≡ la mínima duración de un fallo transitorio.
- Se dispone de una herramienta capaz de inyectar fallos de duración dos veces superior a la máxima vuelta de acceso completa (TDMA) que se pueda programar ≡ fallo permanente.

4.5.1.1. Generación de un pulso inexistente

Generación de un pulso inexistente en una señal de control relativa a la transmisión-recepción de mensajes: Lo aplicaremos sobre señales relacionadas con la habilitación de la transmisión o recepción de mensajes. La figura 4.11. muestra un ejemplo de un pulso inexistente. La línea punteada indica el perfil real de la señal mientras que la línea continua dibuja el resultado de solapar la señal con el fallo.

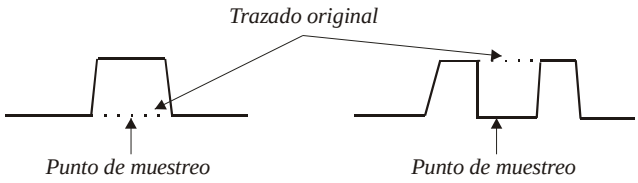


Figura 4.11. Generación de un pulso inexistente en una señal de control

Generación de un pulso inexistente en el bus de memoria: En cada TDMA, la aplicación calcula nuevos valores para las variables relacionadas con el algoritmo de control ejecutado por la aplicación. Estas variables deben almacenarse en la área de mensajes de la memoria del controlador a cada TDMA y formarán el campo de datos de la nueva trama. Cada variable tiene asignada una posición fija en memoria. La actualización de las variables en dicha memoria se realiza a cada TDMA, con tantos accesos consecutivos como posiciones implicadas haya. Por tanto, la variación de la duración del fallo puede afectar a la escritura (o lectura) de una o varias variables, tal como se muestra en la figura 4.12.

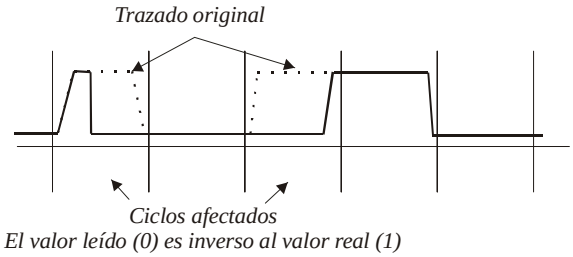


Figura 4.12. Generación de un pulso inexistente en el bus de memoria

Generación de un pulso inexistente en la señal de transmisión: La transmisión serie de la trama se ve afectada por el fallo, alterando algunos de los valores de los bits transmitidos.

4.5.1.2. Eliminación de un pulso existente

Eliminación de un pulso en una señal de control relativa a la transmisión-recepción de mensajes: El efecto del fallo es el mismo que provocaría una línea cortada o pegada permanentemente a un valor. La duración del fallo se considera permanente. La figura 4.13. muestra un ejemplo en el que se elimina durante dos TDMA el pulso relativo a la habilitación de la transmisión.

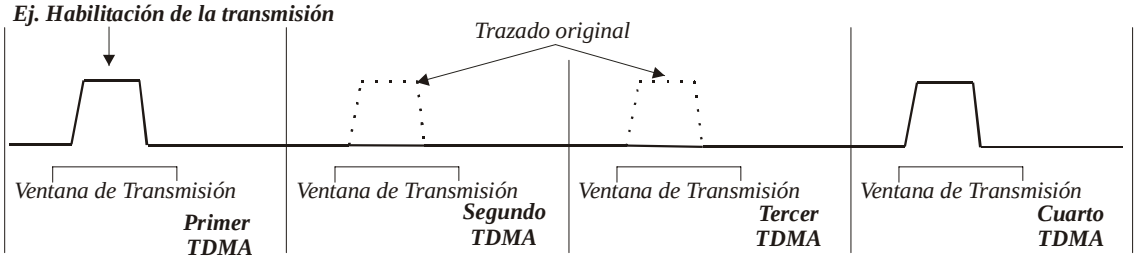


Figura 4.13. Eliminación de pulsos de una señal de control

Eliminación de un pulso en una señal de control de acceso a memoria: Se aplica en las señales de habilitación de lectura o escritura en la memoria del controlador o CNI, impidiendo su correcta actualización. La duración del fallo es equivalente al tiempo necesario para una escritura (o lectura) completa en memoria.

4.5.1.3. Variación de los márgenes temporales del pulso

Un fallo transitorio de duración mínima solapado al pulso real de una señal la puede acortar o prolongar según el valor lógico del fallo. En la figura 4.14. vemos como el flanco de bajada de la señal se retrasa o se adelanta debido al fallo.

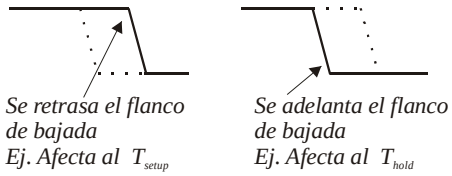


Figura 4.14. Variación de los márgenes temporales de un pulso

4.5.2. Impacto de los fallos sobre las barreras de contención

Una acción realizada por el controlador de comunicaciones durante el servicio del sistema implica la generación sincronizada de diferentes señales de entrada y salida. Por tanto, un fallo simple, aunque afecte sólo a una señal, repercute en el estado de la ejecución.

Para validar la fiabilidad de alguna de las barreras de contención debemos asegurar el correcto funcionamiento del sistema en presencia de fallos que afecten a cualquier señal relacionada con dicha barrera. Haremos, por tanto, dos análisis. El primero, para la validación de la fiabilidad del guardián de bus local. El segundo, para la evaluación del impacto de un fallo en la interfaz de comunicaciones; y en general, para verificar la aserción de la hipótesis de fallos del controlador de comunicaciones TTP/C.

4.5.2.1. Validación de la fiabilidad del guardián del bus local

Hasta el momento existen dos versiones comercializadas del controlador TTP/C, la C1 y la C2. En la C1, las señales de control del guardián del bus son:

- **BG** ("Bus Guardian"): Indica la duración máxima para la transmisión de una trama. Fuera de este pulso, la línea de transmisión debe permanecer a un valor recesivo.
- **OE** ("Output Enable"): Habilita o deshabilita los *drivers* de salida. En la versión C1, el transceptor utilizado es el 82C250 para CAN y permanece constantemente habilitado. En la versión C2, se utiliza un MAX3485 para RS485 y permanece habilitado sólo durante el pulso de OE.
- **CTS** ("Clear To Send"): Determina el tamaño real de la transmisión. Indica el inicio de la transmisión.
- **TxD** ("Transmisión"): Señal TTL de transmisión de datos por el canal de comunicaciones. Salida del controlador – entrada al *driver*.
- **RxD** ("Reception"): Señal TTL de recepción de datos por el canal de comunicaciones. Entrada en el controlador – salida del *driver*.

En la versión C2 las líneas de control son internas, quedando sólo accesibles TxD y RxD, así como el oscilador correspondiente al guardián del bus de 16MHz. Además, la entrada de *reset* asíncrona se encuentra disponible en ambas versiones. En la figura 4.15. se indican los puntos de inyección seleccionados.

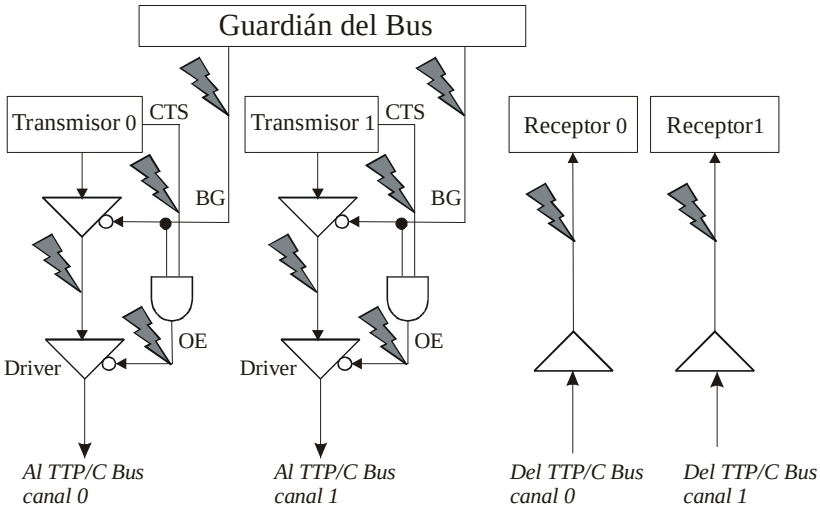


Figura 4.15. Puntos de inyección seleccionados respecto al guardián del bus

A ***nivel lógico***, los **fallos permanentes** en estas líneas modelan el pegado de una entrada o salida del controlador a un valor determinado, o bien que una línea permanece abierta, sin conexión. Con **fallos transitorios simples** se modelan retardos que varían los márgenes temporales de las señales de datos y direcciones, así como la degradación del oscilador. A su vez, también modelan fallos internos del controlador que derivan en pulsos de habilitación no esperados. Los **fallos transitorios conexos** se inyectan en ambos canales de comunicaciones. Internamente, el controlador maneja dos bloques de transmisión y dos de recepción (uno por canal). Para modelar fallos transitorios conexos de acoplamiento entre ambos canales, y fallos internos que afecten a la transmisión en general, se han utilizado fallos dobles que fuerzan ambos canales con el mismo periodo y valor.

En conjunto, a nivel de sistema, los ***modos de avería*** modelados son los relacionados con averías con **parada y reintegración**, **SOS**, **transmisiones espurias** y **pérdidas de conexión**.

En una avería con *parada* el momento más crítico será cuando la parada se produzca durante una transmisión ya iniciada. Esta avería se obtiene acortando la señal de habilitación de la transmisión. Además, cuando la parada implique una reintegración del nodo, dicha reintegración deberá ser segura para la funcionalidad del sistema, eliminando cualquier posibilidad de llevarlo a un estado no deseado.

Las averías *SOS* en el dominio del tiempo se obtiene con fallos transitorios en la señal de transmisión que ocurren en el límite de la apertura o cierre de la ventana de recepción. Aunque el sistema utiliza una base de tiempos común, ésta se calcula de forma interna en cada nodo, siendo necesario un algoritmo de resincronización constante. Por tanto, es normal que exista un margen de error y que la apertura de la ventana de recepción de los nodos tenga cierta desviación. Un fallo coincidente con dicha apertura será percibido por los nodos más rápidos, mientras que los lentos quizá abran su ventana cuando el fallo ya ha desaparecido. El caso contrario ocurrirá en el cierre de la ventana. Las averías *SOS* en el dominio del valor se crean al perturbar la señal de recepción en varios nodos. El nodo o nodos inyectados reciben una trama diferente a aquellos nodos que estén libres de fallos.

Las *transmisiones espurias* se obtienen con fallos transitorios de corta duración sobre la señal de transmisión.

Finalmente, *las pérdidas de conexión* se obtienen al eliminar el pulso de habilitación de la transmisión: la trama no se vuelca sobre el canal. La percepción por parte de los otros nodos es la de una desconexión del nodo inyectado.

4.5.2.2. Evaluación de la interfaz de comunicaciones

Aunque la CNI, o interfaz de comunicaciones, es interna al controlador TTP/C, existe una conexión externa con el microcontrolador que soporta la ejecución del algoritmo de control a nivel de aplicación. Esta conexión incluye las siguientes líneas:

- **CE** ("Chip Enable"): Habilita el acceso a la memoria de la CNI.
- **WE** ("Write Enable"): Para operaciones de escritura en la memoria de la CNI.
- **OE** ("Output Enable"): Para operaciones de lectura sobre la memoria de la CNI.
- **Bus de direcciones**: 12 líneas para direccionar 46 registros de estado y hasta 1.968 direcciones de memoria de mensaje. Estos registros almacenan valores relativos al estado de la aplicación y a eventos detectados en el canal de comunicaciones. En las direcciones de memoria asignadas a mensajes se almacenan las variables de la aplicación cuyos valores deben ser transmitidos como parte del mensaje.
- **Bus de datos**: 16 líneas de datos para acceso al área de mensajes de la CNI y a registros de estado.
- **TTICK**: Señal de entrada utilizada para calcular el ciclo de trabajo.

A ***nivel lógico y RT***, los **fallos permanentes** en las líneas de control modelan la degradación y desconexión de las líneas inyectadas. Los **fallos transitorios simples** modelan retardos que varían los márgenes temporales de las señales de datos y direcciones, así como la degradación del oscilador. Este efecto hace que el acceso de escritura en memoria (o lectura) se retrase, pudiendo violar el margen temporal máximo permitido. A su vez, un pegado efectivo sobre una señal de datos varía el valor el bit transmitido en ese momento, modelando la inversión de un bit en memoria. Los pegados sobre las señales de direcciones corresponden a errores de cálculo, llevando al controlador a direccionar una posición errónea. Para modelar **fallos transitorios conexos** se utilizan pegados dobles en líneas de datos que emulan cortos y acoplamientos.

Las principales causas de un error en la memoria SRAM de la CNI son tres. Primero, un error no detectado en el algoritmo de control que genera una variable errónea. Segundo, un fallo durante el acceso a memoria derivando en una actualización incorrecta. Tercero, un fallo dentro de la propia memoria debido a un evento externo, por ejemplo un SEU. Sea cual sea la causa, el ***modo de avería*** va a ser el mismo, se enviará un mensaje sintácticamente correcto pero semánticamente contendrá un valor erróneo.

Hasta el momento, a la hora de representar los fallos en memoria, se había aceptado el modelo de inversión simple como válido. Sin embargo, este modelo ya no es justificable, sino que debe cambiarse por el de inversión múltiple, teniendo en cuenta que:

- Los errores no detectados en el algoritmo de control no se pueden considerar como fallos en la CNI. Se necesitan estrategias de replicación o redundancia temporal para detectarlos. La replicación hardware es más costosa que la redundancia temporal[56] implementada dentro de la propia aplicación. En los trabajos presentados en [Ademaj 2003 y Aidemark *et al*. 2002] se

[56] "Timing redundancy": comportamiento redundante obtenido por repetición

comprueba que dicha redundancia temporal, implementada en forma de estrategias como la doble ejecución o triple ejecución con votación, son suficientes para obtener una buena cobertura de detección.

- No se puede justificar que un fallo ocurrido durante un acceso a memoria sólo afecte a un bit en la transmisión. Aparte de la cercanía entre pines que aumenta el riesgo de acoplamiento entre líneas a alta frecuencia, debemos tener en cuenta que la duración de un fallo puede ser mayor que la duración de un acceso simple a memoria. Si el fallo persiste durante el segundo acceso, aumenta el riesgo de que se produzcan múltiples errores. Este tipo de errores, donde se fuerza el valor de varios bits en la misma dirección (a 0 o a 1), se denominan **errores unidireccionales múltiples** [Pradhan 1980, Pradhan *ed*. 1996].
- Los llamados "soft errors", o errores reversible en memoria, causados por la radiación en una SRAM, pueden provocar el cambio lógico de más de una celda de almacenamiento adyacentes. Este cambio de valor, o inversión, permanece hasta que la posición de memoria afectada vuelva a escribirse con un nuevo valor. La diferencia con los errores unidireccionales es que, dependiendo de la estructura de la memoria y de la localización del fallo, los cambios pueden ser a niveles lógicos diferentes. Son **errores aleatorios múltiples**.

Tanto los errores unidireccionales como los aleatorios múltiples serán tenidos en cuenta en la validación de la CNI.

4.6. Resumen y Conclusiones del Capítulo

En la actualidad, el diseño de sistemas distribuidos tolerantes a fallos que den soporten a aplicaciones críticas es un reto para la investigación y el desarrollo de arquitecturas fiables y seguras que proporcionen el grado de confiabilidad requerido en productos cuyo correcto funcionamiento es esencial para evitar riesgos humanos o cuantiosas pérdidas económicas.

Entre estos productos, y de pronta aparición en el mercado, se encuentra el controlador de comunicaciones TTP^{TM}/C, basado en la arquitectura de disparo por tiempo TTA. Hereda, por tanto, las condiciones optimas que la arquitectura garantiza sobre comportamiento seguro ante cualquier tipo de fallo que acaezca en el sistema. Sin embargo, la aserción teórica de un comportamiento libre de inconsistencias no es suficiente ante la importancia del uso destinado para el producto: el desarrollo de controles de seguridad activa en sistemas guiados por cable ("*x-by-wire*"). De ahí, la importancia de la validación experimental del protocolo de comunicaciones. Este capítulo revisa la arquitectura TTA y los protocolos de comunicaciones basados en ella, entre ellos, el TTP bajo el que se diseña el controlador de comunicaciones TTP/C, objetivo de la validación experimental llevada a cabo mediante inyección de fallos a nivel de pin.

Los fallos físicos a nivel de pin permiten causar, de forma controlada, varios efectos sobre las señales de entrada y salida del prototipo, y más concretamente, sobre las líneas vinculadas a las barreras de contención de errores del controlador de comunicaciones TTP/C. Estas barreras de contención constituyen una parte fundamental en la arquitectura TTA para garantizar su fiabilidad.

Finalmente, se analiza la relación entre el conjunto de fallos inyectables y el impacto que se desea obtener en el sistema. Define, por tanto, el dominio de entrada de la validación experimental en el controlador ante fallos a nivel de pin. El conjunto de fallos

incluye el detalle de su localización, modelo y persistencia, mientras que el impacto alude a los modos de avería que observaremos y ante los cuales el protocolo de comunicaciones deberá mantener un estado consistente. En general, el capítulo presenta la base para configurar el entorno de validación.

Capítulo 5. AFIT- La Herramienta de Inyección
Adaptación para Sistemas Distribuidos

En este capítulo se describe la herramienta de inyección de fallos físicos a nivel de pin utilizada en la validación del TTP^{TM}/C. Se trata de una herramienta genérica, aplicable sobre diferentes tipos de sistemas y configuraciones, específicamente adaptada a sistemas distribuidos.

5.1. Evolución de la Herramienta de Inyección de Fallos

La primera versión de la herramienta de inyección **AFIT** ("Advanced Fault Injection Tool") [Gil 1992], diseñada por el grupo de investigación de Sistemas Tolerantes a Fallos (GSTF), del departamento de Informática de Sistemas y Computadores de la Universidad Politécnica de Valencia, consiste en una herramienta modular cuya interacción con el usuario se realiza a través de un PC. La segunda versión, [Gil *et al*. 1997 y Martínez *et al*. 1999], mantiene la misma estructura, pero consigue aumentar la frecuencia de inyección hasta 40MHz. Esta frecuencia determina la duración del fallo transitorio mínimo en 25ns.

AFIT se divide en cinco módulos interconectados: módulo de **sincronización y disparo**, módulo de **temporización**, módulo de **activación**, módulo de **lectura de eventos** y finalmente, el módulo de **potencia** con las **puntas de inyección de alta velocidad** (ver figura 5.1.) [Gil, Blanc y Serrano 2003].

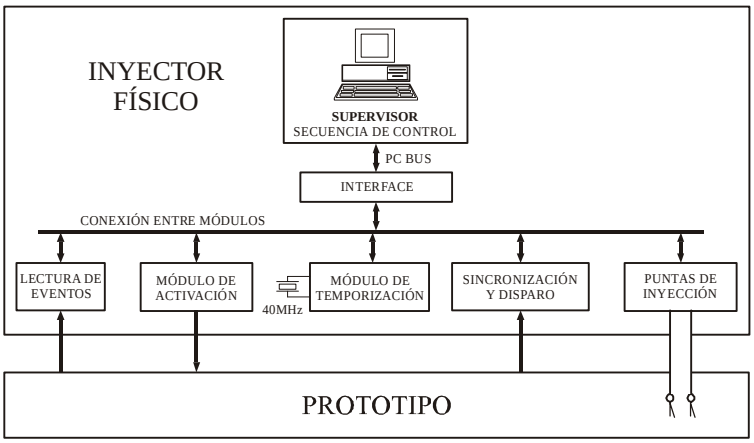


Figura 5.1. División modular de AFIT

La evolución de los sistemas, en términos de integración y velocidad, fuerzan la evolución de las herramientas de inyección. Algunas técnicas, como por ejemplo la inyección por inserción, actualmente son difíciles de aplicar. Sin embargo, la técnica de inyección de fallos por forzado a nivel de pin sigue siendo apropiada para los nuevos diseños. Además, teniendo en cuenta que AFIT puede forzar fallos transitorios desde

25ns hasta fallos permanentes, hace que esta herramienta se proponga para la validación de diferentes sistemas tolerantes a fallos en proyectos de investigación propios y en colaboración con otros grupos de investigación.

Así, en [Blanc 1998] se mejoran los módulos de **temporización** y **potencia**, rediseñando también las **puntas de inyección de alta velocidad** para aumentar el número de puntas de inyección independientes en el forzado de fallos permanentes, transitorios e intermitentes de hasta 50MHz. La herramienta se simplifica, ya que el disparo de la inyección y la lectura de eventos son externos en esta nueva versión. El nuevo inyector se construye en 1998 y se utiliza para validar con fallos físicos un nodo de DICOS ("Distributed Control System") [Campelo *et al*. 1999], un sistema de control industrial desarrollado en el mismo grupo de investigación de Sistemas Tolerantes a Fallos [Campelo 1999]. Los resultados de los experimentos llevados a cabo sobre DICOS mediante inyección de fallos a nivel de pin se comparan con los resultados obtenidos mediante otra herramienta SWIFI, de tipo "runtime" con recursos software (**SOFI** [Campelo 1999]), destacando las diferencias básicas entre ambas técnicas de inyección y resolviendo el problema de la unificación de las medidas obtenidas con diferentes herramientas para el cálculo global de la cobertura de detección [Blanc 2000 y Blanc *et al*. 2001].

Tanto en el trabajo de validación de DICOS como en el planteamiento inicial de la validación del TTP^{TM}/C [Blanc *et al*. 2002 (a)], se destaca que la utilización de varias técnicas de inyección para validar un mismo sistema no sólo es compatible, sino que también es recomendable. Debido a que ninguna técnica de inyección de fallos cubre por completo todos los requisitos de la validación, sino que cada técnica abarca un subconjunto determinado del domino de entrada (*fallo* × *activación*), podemos decir que las técnicas son complementarias. Sin embargo, para poder aplicar un criterio uniforme a los resultados obtenidos con distintas técnicas, es necesario un dominio de salida (*lecturas* × *medidas*) equivalente en todas las campañas. Durante el proyecto FIT[57], se observa la necesidad de diseñar un nuevo módulo para la *sincronización entre prototipo y herramienta* y para la *lectura de eventos*, especialmente trabajando con sistemas distribuidos de tiempo real [FIT 2002, Blanc *et al*. 2002 (b), Blanc y Gil 2003]. Este módulo, que se presenta íntegramente en este capítulo, se diseña como una herramienta independiente que podrá ser utilizada junto con cualquier herramienta de inyección de fallos aplicable a prototipos.

5.1.1. Descripción modular de la herramienta

En este apartado se revisan las características más importantes de los módulos de AFIT [Gil *et al*. 1997, Martínez *el al*. 1999, Gil, Blanc y Serrano 2003]:

5.1.1.1. Módulo de sincronización y disparo

Este módulo determina el comienzo de un experimento de inyección. Cada experimento implica la inyección de un fallo simple o de varios fallos al mismo tiempo (fallo múltiple), en un instante determinado de la función del sistema, cuya activación se implementa en dos bloques (figura 5.2.): **detector de la palabra de disparo** y generación del **retardo programable**. Tras programar una palabra de disparo *TW* y habilitar la inyección *IE*, el bloque *detector de la palabra de disparo* comienza el muestreo de un subconjunto de líneas (buses de direcciones o datos, señales de control, etc.), para determinar cuál es el estado interno del dispositivo. Cuando el estado interno coincida

[57] Fault Injection for TTA

con la palabra de disparo *TW*, tras un *retardo programable*, se genera **el disparo de la inyección** *IT* que activa el módulo de temporización.

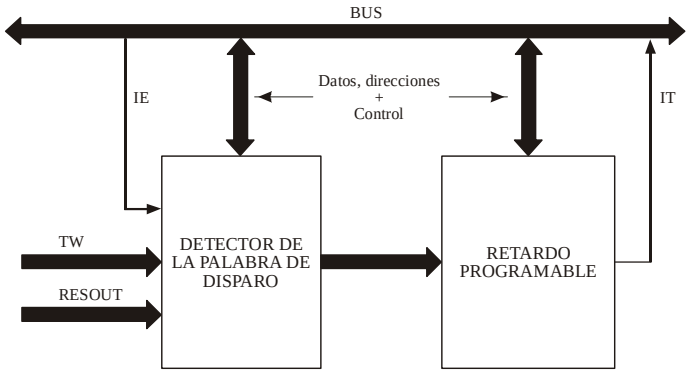


Figura 5.2. Módulo de sincronización y disparo

5.1.1.2. Módulo de temporización

El módulo de temporización es el generador de las frecuencias de inyección. Según la persistencia del fallo deseada, se generará un pulso permanente, una secuencia de pulsos intermitentes o un único pulso transitorio. Tanto el periodo de la inyección como la duración del fallo son programables para fallos intermitentes. Para fallos transitorios se programa la duración del pulso.

El fallo se genera tras la recepción del *disparo de la inyección IT* desde el módulo de sincronización y disparo, y la señal que genera este módulo es **la señal de inyección** *AI*, dirigida al módulo de potencia (figura 5.3.).

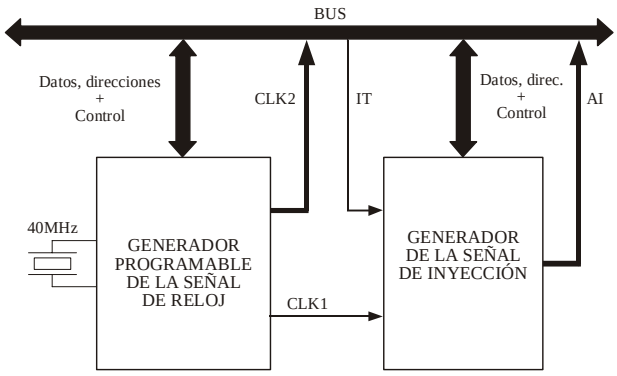


Figura 5.3. Módulo de temporización

5.1.1.3. Módulo de activación

Para poder evaluar correctamente el efecto de los fallos sobre un dispositivo es necesario asegurar que, en el momento de la inyección, el dispositivo (y el sistema en general) se encuentra en un estado conocido. Este módulo (figura 5.4.) se encarga de llevar al sistema hasta dicho estado, utilizando un vector de activación *AV*. El vector se forma a partir de un generador programable de señales de inicialización *RESIN* y el módulo de activación que fuerza valores predeterminados en las salidas *OAS*.

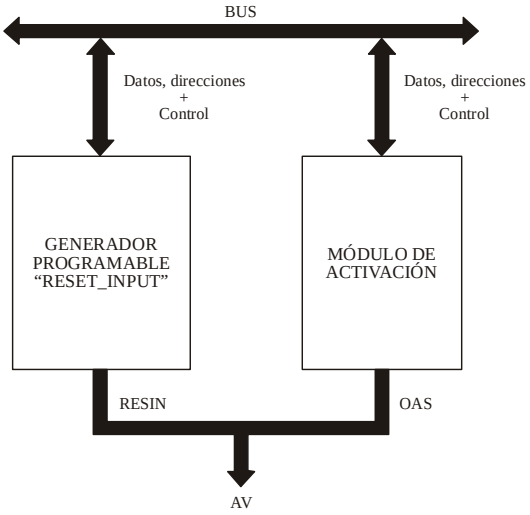


Figura 5.4. Módulo de activación

5.1.1.4. Módulo de lectura de eventos

El módulo de lectura de eventos es quien observa la respuesta del sistema. Está sincronizado con el módulo de temporización. Es capaz de medir diferentes tiempos de latencia y almacenar la traza de un subconjunto de señales preestablecidas desde el momento de la inyección del fallo. Esta traza será analizada a posteriori para la validación del comportamiento del sistema en presencia de fallos.

5.1.1.5. Módulo de potencia

El módulo de potencia se divide en dos bloques (figura 5.5.): **las puntas de inyección de alta velocidad** y el circuito de **detección de la efectividad del fallo**.

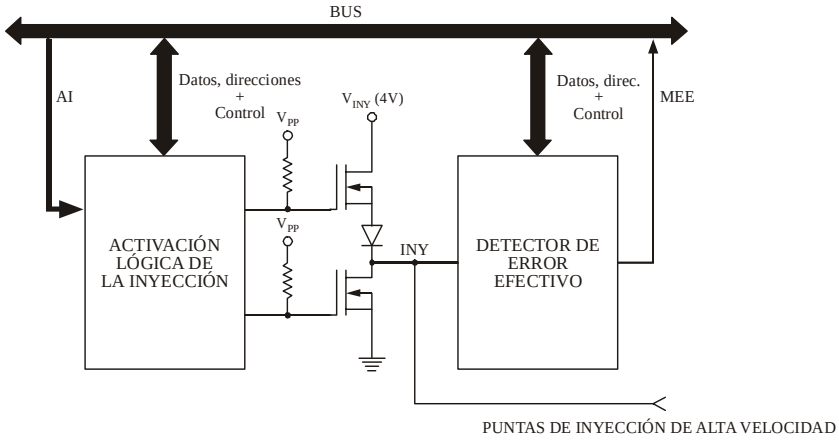


Figura 5.5. Módulo de potencia

Existen 40 *puntas de inyección*. En cada experimento se habilita una punta de inyección, seleccionada automáticamente, para forzar un pegado simple a 0 o a 1. La persistencia del fallo inyectado depende de la *señal de inyección AI*, generada en el módulo de temporización.

El circuito de *error efectivo* es necesario para determinar si el fallo ha forzado un nuevo valor en la señal seleccionada o no, indicándolo con la salida *MEE*. Por ejemplo, si

se fuerza un pegado a 0 sobre una señal cuyo valor lógico ya era 0, el fallo no es efectivo.

5.1.2. Mejoras realizadas en AFIT

La versión de AFIT diseñada en 1998, y presentada totalmente en [Blanc 1998], mantiene la estructura modular original, pero en su forma simplificada. Sustituye el módulo de sincronización y disparo, así como el módulo dedicado a la lectura de eventos, por un analizador lógico comercial de altas prestaciones y controlable a través de un canal serie de entrada. También se mejoran las etapas de temporización y potencia y se implementa una nueva interfaz entre el usuario y la herramienta.

Tanto la herramienta AFIT como el analizador lógico se encuentran conectados a un PC, donde el usuario puede diseñar el conjunto de experimentos que quiere realizar. Los experimentos se llevan a cabo de forma automatizada, sin intervención del usuario. Este conjunto de experimentos constituyen la campaña de inyección. La duración de un experimento es baja, permitiendo realizar un número elevado de experimentos en un tiempo razonable.

5.1.2.1. Nuevo módulo de temporización

El módulo de temporización cuenta con el **canal de programación** y la entrada de **disparo de la inyección**. Al comienzo de cada experimento es necesario programar este módulo a través del canal de programación con los valores que describen el fallo, como son su persistencia y el tipo de pegado (a 0 o a 1), así como el lugar del circuito donde se quiere inyectar. Cada experimento de la campaña puede inyectar un fallo de diferentes características.

Existe una coordinación entre los distintos módulos, tal como se muestra en la figura 5.6. dirigida desde el PC o **supervisor**. Durante cada experimento se ejecuta una secuencia de control que fija el orden de las acciones realizadas por la herramienta sobre el dispositivo. Esta secuencia, implementada en el *supervisor*, se repite tantas veces como experimentos deba contener la campaña de inyección, pudiendo alterar las características del fallo en cada experimento.

El *canal de programación* une la aplicación, que ejecuta la secuencia de control de la herramienta, con la propia herramienta a través de un **bus de expansión**[58]. La aplicación se ejecuta en el supervisor y sirve de interfaz con el usuario. Esta misma aplicación puede forzar las **puntas de activación** de la herramienta en un momento dado, conduciendo al sistema a un estado conocido. Existen cuatro *puntas de activación*, que se pueden programar individualmente, para forzar un 0 o un 1 lógico durante un tiempo programable sobre la entrada del dispositivo seleccionada (normalmente entrada de *reset*).

El *disparo de la inyección* se puede considerar como una acción compleja, compuesta por tres tareas: decisión, reconocimiento y tratamiento. Se entiende por decisión la planificación de cómo será la señal que el módulo va ha reconocer como disparo de la inyección. El reconocimiento es el tiempo desde que la herramienta se encuentra en espera de esta señal hasta que dicha señal sea detectada. El tratamiento es el

[58] El diseño de la herramienta incluye una tarjeta de expansión del bus ISA del PC sobre el que corre la aplicación. Existe una conexión directa entre la herramienta y la salida de la tarjeta de expansión mediante un cable plano de 32 líneas. Esta conexión es a lo que se ha denominado como *bus de expansión*.

tiempo de respuesta de la herramienta. Esta respuesta se traduce en un evento interno que activará el funcionamiento de los otros módulos implicados: *la señal de inyección*.

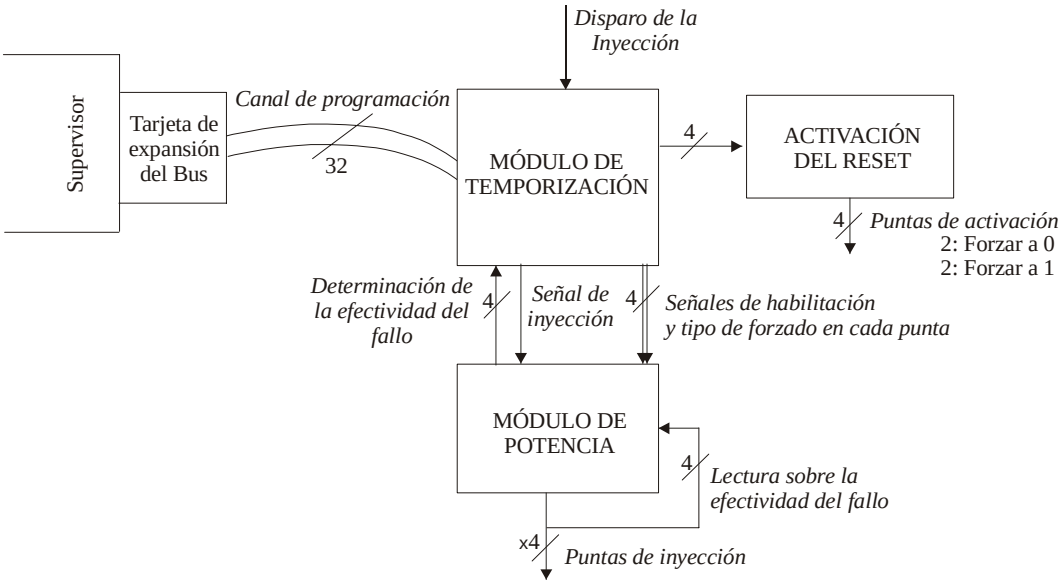


Figura 5.6. Esquema general de conexiones

La mayor parte del nuevo módulo de temporización se basa en lógica programable con funcionamiento síncrono. Se consigue así una disminución del retardo de tratamiento del disparo a 12ns [Blanc 1998]. Tras el reconocimiento del disparo se genera la señal de inyección, consistente en un pulso negativo de excitación, entrada del módulo de potencia, cuya duración y periodo equivale a la del fallo deseado.

5.1.2.2. Nuevo módulo de potencia

El módulo de potencia se diseña con cuatro inyectores independientes, con sus cuatro circuitos de error efectivo. La señal de inyección se desdobla, por cada inyector, como entrada a un circuito formado básicamente por dos transistores P123 Polyfet (“Vertical-Lateral Double Diffuse MOS”). Estos transistores presentan tiempos de conmutación, de subida y de bajada muy bajos, además de una capacidad de entrada bastante inferior que los *power* MOSFET, permitiendo reducir la intensidad de base y por tanto, el consumo y el retardo. El retardo máximo de esta etapa es de 25ns [Blanc 1998], un total de 37ns desde el disparo de la inyección hasta que el forzado es efectivo. La selección del tipo de pegado en cada inyector es independiente. Con respecto a la versión anterior, se incorpora la posibilidad de generar fallos múltiples, pudiéndose forzar un fallo por inyector[59].

La conexión física entre una punta de inyección y el dispositivo a inyectar se realiza a través de las **sondas de inyección**. La herramienta implementa dos inyectores con sondas directas y otros dos inyectores con sondas a través de zócalo, el cual permite distribuir la salida entre 32 puntas de inyección, donde sólo se selecciona una por experimento. El zócalo es de gran utilidad para inyección sobre buses, donde la localización del fallo se distribuye sobre el conjunto de líneas que lo forman. Permite acoplar la herramienta al bus sin necesidad de moverlo durante toda la campaña. Al principio de cada experimento de la campaña se programa la línea del bus sobre la que se

[59] La multiplicidad es x4

inyectará el fallo para que en el zócalo se habilite la salida correspondiente (ver figura 5.7.).

El circuito de error efectivo se activa si en el momento de la inyección el valor de la señal seleccionada es diferente al valor del forzado.

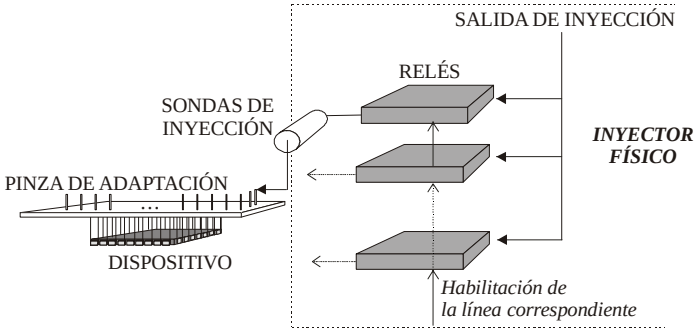


Figura 5.7. Zócalo de conexión

5.1.2.4. Terminadores para las puntas de inyección de alta velocidad

Las herramientas de inyección física de fallos presentan, entre sus inconvenientes, el riesgo de perturbaciones no deseables en el dispositivo bajo prueba. En el caso de la inyección física a nivel de pin, la capacidad parásita de la sonda puede impedir el correcto funcionamiento del dispositivo. Para disminuir esta capacidad parásita se han diseñado y añadido **terminadores** al final de las sondas, adecuando la conexión de la herramienta a líneas en las que antes no era posible una conexión directa.

Los terminadores tiene una doble salida, dependiendo del tipo de forzado que se quiera llevar a cabo (a 0 o a 1). Se basan en diodos de pequeña señal 1N4148 (ver figura 5.8.) y se añaden al final de las sondas directas (ver figura 5.9.), o en el caso del zócalo, será necesario un terminador por línea en la pinza de adaptación (ver figura 5.10.).

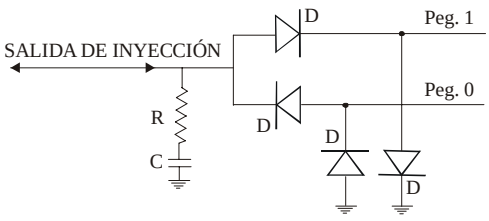


Figura 5.8. Terminadores

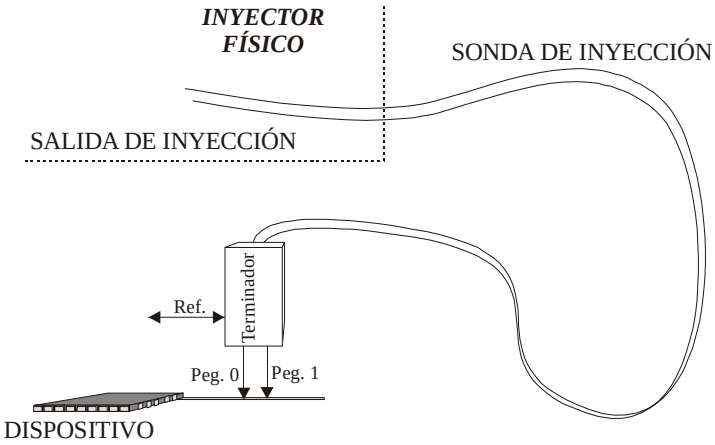


Figura 5.9. Terminadores en las sondas directas

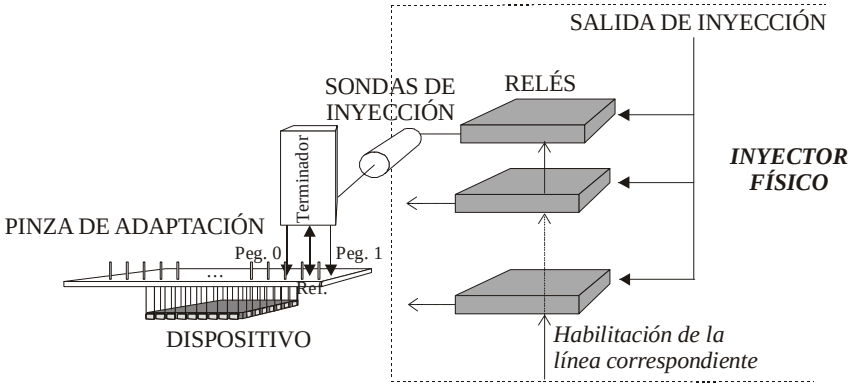


Figura 5.10. Terminadores en los zócalos

5.2. Adaptación a Sistemas Distribuidos

La complejidad de los sistemas tolerantes a fallos distribuidos de tiempo real, y especialmente de los sistemas empotrados, es cada vez mayor, así como la necesidad de validarlos experimentalmente. Debido a las restricciones de funcionamiento que estos sistemas de tiempo real presentan, y a su propia condición de distribuidos, son necesarias nuevas soluciones dentro de la validación experimental que no impliquen una sobrecarga en la función del sistema y que resuelvan los problemas relacionados con la integración de la herramienta en el sistema sin que sea necesario detener o retrasar la ejecución de la carga de trabajo. También es necesario resolver la lectura dinámica de los distintos eventos que ocurren durante un mismo intervalo de tiempo, aunque en emplazamientos físicamente distanciados.

La inyección física a nivel de pin se aplica externamente al sistema a validar, sin que su integración signifique una sobrecarga o alteración en la carga de trabajo. En relación a la lectura dinámica de los eventos, es esencial desarrollar un método que nos permita obtener observaciones útiles sobre el comportamiento del sistema completo y no sólo de la parte inyectada. En este trabajo se presenta una solución para la monitorización de sistemas distribuidos, que consiste en una combinación de recursos hardware y software, y cuya mayor ventaja reside en la posibilidad de obtener lecturas de cada uno de los nodos que componen el sistema mediante mecanismos físicamente independientes,

evitando pérdidas o alteración de la información obtenida de cada nodo. Además, un almacenamiento ordenado e igualmente dinámico de las lecturas en una base de datos global va a permitir el procesamiento posterior de esta información para el cálculo de medidas.

Tanto la herramienta de inyección física como el monitor tienen un funcionamiento automático, lo que hace posible generar muestras con los resultados de los experimentos sin intervención del usuario.

5.2.1. Procesos implicados en la inyección de fallos

Para cualquier técnica de inyección de fallos, junto con el proceso de **inyección** coexisten otros dos procesos implicados en la validación: la **adquisición de datos**[60] y el **análisis de los resultados**[61]. Los tres procesos forman el entorno de validación (ver figura 5.11.).

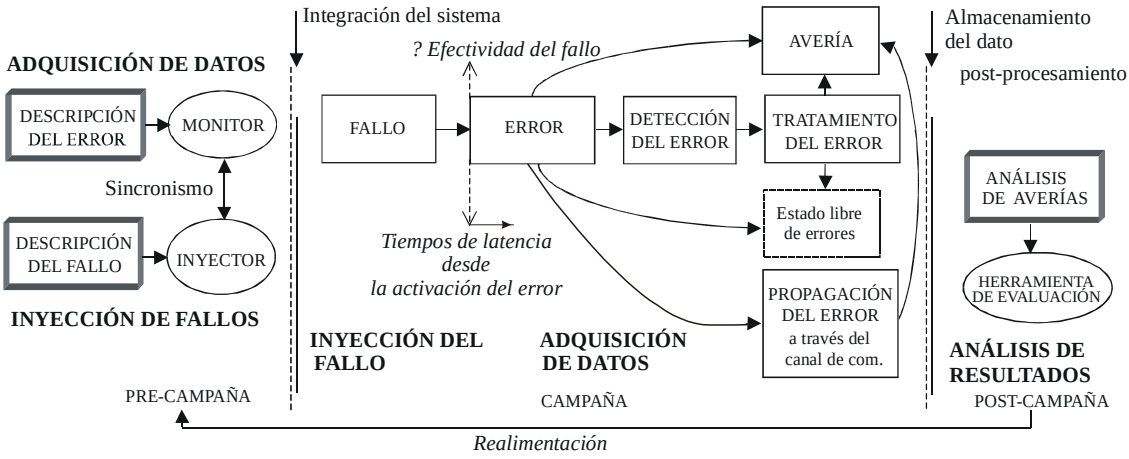


Figura 5.11. Estructura del entorno de validación

Una técnica de inyección de fallos, materializada en una herramienta, es capaz de inyectar un conjunto conocido de fallos. Por tanto, el sistema será validado frente a ese conjunto de fallos y en función de las lecturas que se obtengan de su comportamiento. Los resultados de la validación dependen de las lecturas, o datos adquiridos, que trazan la activación de los mecanismos de detección y contención de errores, haciendo necesaria una sincronización activa entre la herramienta de inyección y la herramienta de adquisición, especialmente si se llevan a cabo tareas relacionadas con la *eliminación de fallos*.

La complejidad de la herramienta de adquisición depende de la información necesaria para validar o evaluar el sistema. Las lecturas esperadas en cada experimento abarcan desde la descripción del fallo hasta la última reacción del sistema; es decir: la efectividad del fallo, la activación del error, la detección del error, los mecanismos implicados en la detección, el diagnostico y contención del error, la recuperación después del fallo o la propagación del error y el modo de avería. La *efectividad del fallo* tiene especial importancia en la inyección física, determinando si un fallo ha causado la perturbación esperada y puede activar un error. Sin embargo, no todos los errores son activados, ya que la redundancia no intencionada o el enmascaramiento de fallos pueden

[60] La adquisición de datos incluye la lectura de eventos como parte del proceso.
[61] Esta división se puede encontrar en [Martínez *et al*. 1999, Gil, Blanc y Serrano 2003 y Blanc y Gil 2003]

devolver al sistema a un estado libre de errores sin la necesaria activación de los mecanismos de detección. El almacenamiento estructurado de las lecturas, junto con ciertas medidas sobre tiempos de latencia, son parte del proceso de adquisición de datos.

Para poder obtener una muestra de resultados en un tiempo viable, y que sea suficientemente representativa del comportamiento del sistema, sería deseable que los procesos de inyección de fallos y de adquisición de datos fueran automatizables. El tiempo dedicado a ambos procesos, cuando trabajamos con un prototipo del sistema, es muy pequeño. Sin embargo, el análisis de los resultados es un proceso que consume gran cantidad de tiempo y que normalmente incluye cálculos que requieren la muestra completa. Por ese motivo, llevando a cabo un almacenamiento estructurado de las lecturas, el análisis de los resultados se puede realizar después de la campaña de inyección. Durante este tercer proceso, pueden detectarse carencias en los procesos de inyección de fallos y adquisición de datos, que deberán ser rectificados en una nueva campaña (realimentación en el proceso de validación).

En un sistema distribuido, los tres procesos se aplican sobre el sistema completo. Aunque el fallo se localice en un nodo, la posible propagación del error a través del canal de comunicaciones determina la importancia de observar el comportamiento de todos los nodos conectados al canal. El proceso de inyección de fallos se realiza con la herramienta AFIT, sincronizada con un **monitor para sistemas distribuidos**, cuyo diseño e implementación se detalla en el siguiente apartado.

5.2.2. Un monitor para sistemas distribuidos

El monitor, que se presenta como aportación en este trabajo de tesis, diseñado para sistemas distribuidos, combina recursos hardware con tareas software. El monitor permite generar una base de datos estructurada durante la campaña de inyección que incluye, para cada experimento, las características del fallo, su efectividad, las lecturas sobre el comportamiento de los nodos y el tiempo de latencia desde la inyección hasta la percepción de la ocurrencia de algún evento.

Podemos dividir su estructura en tres partes. Primero, **el monitor**[62], externo al sistema a observar. Segundo, **las tareas software** ejecutadas en los nodos a monitorizar, cuyo objetivo es adquirir información del estado interno del nodo y transferirla al monitor. Tercero, **la secuencia de control del monitor** ejecutada desde el PC **supervisor**.

La figura 5.12. representa la conexión entre AFIT, el monitor, el PC supervisor y el sistema a validar (un *cluster*[63] del TTP).

El PC *supervisor*, a través de un bus de expansión, se encarga de la secuencia de control, además de gestionar el volcado de las lecturas obtenidas en cada experimento en la base de datos global implementada en el propio supervisor.

Es deseable una sincronización activa entre el proceso de inyección de fallos y el de adquisición de datos. En nuestro entorno de validación, tanto AFIT como el monitor son controlados desde un mismo supervisor. De hecho, el supervisor ejecuta una única secuencia de control con acciones dirigidas a ambas herramientas. Esta integración en el control de ambos procesos permite que parámetros como el inicio o final del experimento sean variables conocidas dinámicamente por ambas.

[62] Llamado SCCircuit en la documentación del proyecto FIT (“Fault Injection for TTA”)
[63] Conjunto de nodos que comparten el canal de comunicaciones

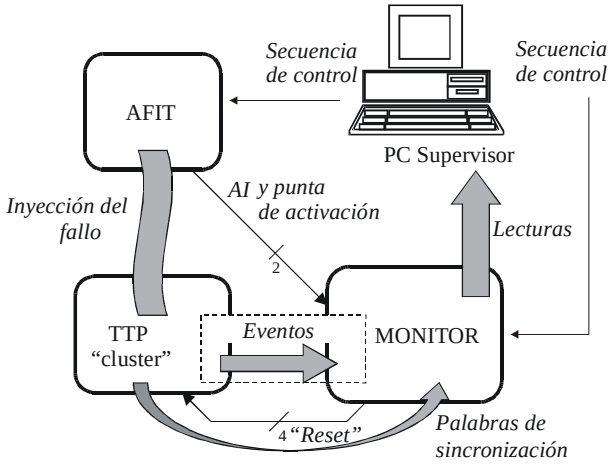


Figura 5.12. Conexión esquemática AFIT-Monitor

El objetivo de la *puntas de activación* diseñadas en AFIT es conducir al sistema a un estado conocido al inicio de cada experimento. Una de las puntas de activación de AFIT se ha derivado al monitor y hace de puente hacia las entradas de *reset* de los controladores de comunicaciones de los TTP nodos. Durante un experimento, el sistema podría alcanzar un estado no seguro a partir del cual es impredecible determinar si habrá y cuál será el siguiente estado seguro y conocido. Por tanto, se entiende que en una campaña de inyección de fallos automatizada el estado inicial del sistema[64] sea el forzado por las *puntas de activación*. Sin embargo, en un sistema en el que un nodo puede alcanzar diferentes estados internos, la posibilidad de retrasar el inicio del experimento hasta que el nodo haya alcanzado un estado concreto es muy conveniente. Por ejemplo, en la primera versión del TTP/C, el prototipo C1, según su hipótesis de fallos, el estado inicial del nodo no se encuentra especificado como tolerante a fallos. Por tanto, en este estado no se puede validar ningún mecanismo de detección de errores, siendo conveniente retrasar el experimento hasta que el nodo alcance un estado normal de ejecución. Para estos casos, es deseable una sincronización a mayor nivel entre el sistema a validar y la herramienta de inyección. En nuestro entorno se consigue dicha sincronización mediante las palabras de sincronización.

El monitor dispone de una conexión en paralelo de 8 bits por cada nodo. Estos 8 bits forman la **palabra de sincronización**. Tras activar las *puntas de activación,* la siguiente sentencia de la secuencia de control es esperar la detección de todas las *palabras de sincronización,* teniendo en cuenta un tiempo de espera máximo. Cada palabra de sincronización es una combinación de 8 bits que el nodo escribe en un puerto de salida designado al alcanzar el estado deseado, y que mantiene durante cierto periodo de tiempo. Para iniciar el experimento, el monitor espera a que todas las palabras se fijen en la conexión correspondiente, evento que es transmitido al supervisor y, a su vez, a la herramienta de inyección.

El disparo o instante de inyección se puede seleccionar de tres formas. Puede activarse desde el supervisor, como una sentencia más de la secuencia de control. Puede activarse directamente desde el monitor, al detectarse las palabras de sincronización, con

[64] En el TTP, como en otros protocolos de comunicaciones, el estado inicial del sistema es aquél en el que cada nodo reconoce su conexión al canal y mediante tramas específicas de inicialización, adquiere información sobre el sistema, como por ejemplo, el número de nodos conectados, su estado y orden de transmisión. Una vez superado el estado inicial, el nodo comienza la ejecución de sus tareas y el envío de mensajes de datos, según la política de transmisión implementada.

o sin retardo. Finalmente, puede utilizarse un disparo externo a ambas herramientas. Un ejemplo de esta última opción es la utilización de un analizador lógico comercial, que tiene a favor la posibilidad de definir un disparo por condición con suficiente nivel de detalle.

La figura 5.13. muestra el detalle del monitor.

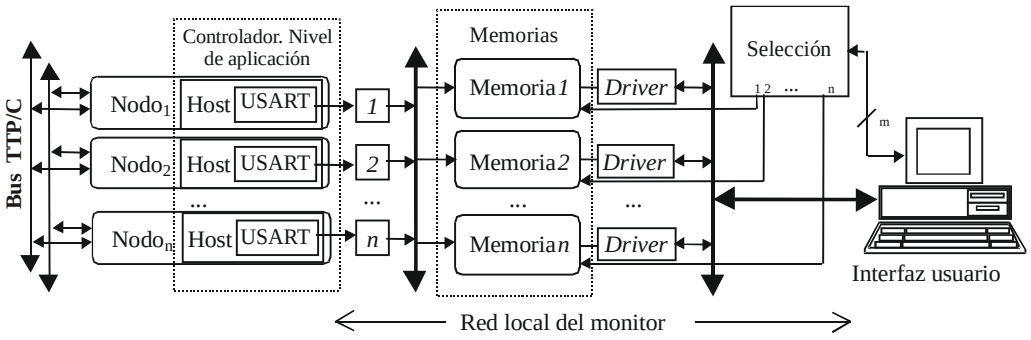


Figura 5.13. Red local del monitor para sistemas distribuidos

En cada nodo se implementa una o varias *tareas software*, que se ejecutan sólo esporádicamente y a causa de la detección de cierto evento. Su condición de *tareas esporádicas* tiene como objetivo causar la menor sobrecarga posible en la carga de trabajo del nodo. Ante un evento predeterminado, el nodo ejecutará la acción o acciones implementadas en la tarea para obtener de información relativa a su estado interno. Dicha información se transfiere al monitor a través de la USART[65].

El monitor mantiene una conexión dedicada con cada nodo del sistema que consiste en un enlace serie, permanentemente habilitado, sobre el que el nodo es libre de transmitir datos en cualquier instante de tiempo dentro de los límites temporales del experimento. Cada nodo genera sus propias lecturas, por tanto, el monitor recibe en paralelo las lecturas de todos los nodos y las almacena en memorias independientes junto con una marca que indica el momento de su recepción, para volcarse al final del experimento en la base de datos global. La marca indica el tiempo transcurrido desde la inyección del fallo hasta la recepción de la lectura.

El circuito del monitor se diseña utilizando microcontroladores con memoria interna, para la gestión de la **red local**. Esta red tiene tres partes básicas, las conexiones dedicadas con los nodos, los segmentos de memoria interna destinados al almacenamiento de la información transferida por cada nodo y el bus local de conexión entre el monitor y el PC *supervisor* (ver figura 5.13.).

La duración total de un experimento puede ser inferior a 1 segundo, aunque depende principalmente del tiempo de observación requerido tras la inyección del fallo, lo que posibilita obtener muestras representativas del comportamiento del sistema con un alto grado de confianza. El número total de experimentos realizados puede ser muy elevado, por lo que una estructuración correcta de la base de datos global que almacena los resultados de cada experimento simplifica el proceso de *análisis de resultados*. Los datos que se obtienen en cada experimento son los siguientes:

- **Relativos al fallo**: Se refieren a la localización del fallo o la línea o líneas inyectadas, tanto para fallos simples como múltiples; la persistencia del fallo, diferenciando entre permanentes, intermitentes o transitorios; tipo

[65] Universal Synchronous Asynchronous Receiver Transmitter

de pegado que se aplica a cada línea inyectada; condición que activa el disparo de la inyección y efectividad del fallo.

- **Relativos a las lecturas obtenidas**: Cada lectura incluye el identificador del nodo que la genera; el contenido de los datos transferidos por el nodo y el tiempo desde que se inyecta el fallo hasta que se recibe la lectura. Este último parámetro, en un sistema con arquitectura TTA, permite calcular la distancia en ciclos de acceso o en TDMA desde que se origina el fallo hasta que cada nodo percibe una avería, siendo posible observar si el error activado ha sido correctamente contenido dentro del nodo inyectado, o si por el contrario, se ha propagado a través del canal de comunicaciones.

Tanto el monitor como la herramienta AFIT son genéricas, y por tanto, pueden utilizarse en la validación de sistemas basados en arquitecturas diferentes. Sin embargo, la secuencia de control, que tienen en cuenta las diversas acciones que se deben realizar sobre el sistema a validar, depende del objetivo y orientación de la campaña (eliminación o predicción de fallos).

Como consideración adicional, se debe tener en cuenta que la sobrecarga que una tarea software pueda suponer dependerá de la arquitectura del sistema. Por ejemplo, en una arquitectura TTA, la ejecución de cada tarea implementada a nivel de aplicación, e incluso, las tareas específicas de lectura o escritura sobre la interfaz de comunicaciones, tienen un patrón temporal fijo establecido a priori. A cada tarea se le asigna un instante de inicio y un periodo máximo de ejecución. Por lo tanto, es posible introducir tareas de monitorización en los lapsus temporales no asignados a ninguna tarea, en los que la carga se mantiene ociosa, consiguiendo una sobrecarga temporal nula en la carga de trabajo. Estas tareas de monitorización se implementan para la comprobación de registros de estado y análisis de los valores transmitidos a través del canal de comunicaciones.

Por otro lado, la mayoría de los microcontroladores utilizados actualmente en algoritmos de control para automoción están capacitados para el tratamiento de excepciones. Estas excepciones, generadas a raíz de un error, son eventos extraordinarios. La implementación de tareas como parte del tratamiento de la excepción no supone una sobrecarga adicional a la carga de trabajo ejecutada en el modo normal de funcionamiento especificado en el sistema.

5.3. Resumen y Conclusiones del Capítulo

En este capítulo se describen las ampliaciones y modificaciones realizadas sobre la herramienta de inyección física de fallos a nivel de pin, AFIT (“Advanced Fault Injection Tool”), para su utilización con sistemas distribuidos. Esta herramienta genérica es el medio que permitirá inyectar fallos físicos en prototipos de diferentes sistemas como parte de su validación experimental.

Se ha puesto especial énfasis en las mejoras realizadas sobre los módulos de temporización y potencia. Por un lado, se han reducido los retardos de propagación de las señales internas de control de la inyección, llegando a obtener un máximo de 37ns de retardo desde el disparo de la inyección hasta que el forzado es efectivo. Por otro lado, se ha capacitado a la herramienta para inyectar fallos múltiples, disponiendo de sondas de conexión directa o a través de zócalos de gran utilidad para la inyección sobre buses. Además, se dota a AFIT de nuevos terminadores para las sondas que eviten el riesgo de perturbaciones no deseadas en el dispositivo bajo prueba.

La aportación más importante del capítulo es la adaptación realizada para sistemas distribuidos. Se trata de una nueva herramienta, también genérica, diseñada para resolver el proceso de adquisición de datos con herramientas de inyección que trabajen sobre un prototipo del sistema o el sistema real. La herramienta, diseñada y construida durante el proyecto de investigación FIT, combina un monitor hardware con tareas software implementadas según la arquitectura del sistema a observar. La sincronización entre AFIT y el monitor diseñado es de gran importancia para la adquisición de tiempos de latencia. Esta sincronización se consigue mediante la utilización de una única interfaz que supervisa la secuencia de control ejecutada en ambas herramientas.

Capítulo 6. Validación del Protocolo de Comunicaciones

La fiabilidad del protocolo de comunicaciones TTP se basa en la correcta especificación del controlador de comunicaciones TTP^{TM}/C. Sin embargo, en la validación de un sistema distribuido bajo aquellos criterios requeridos por las aplicaciones que debe soportar, no sólo intervienen los atributos de cada miembro que comparte el canal de comunicaciones, sino también la integración segura del sistema garantizando que su composición final al unir sus diferentes partes no degrade o invalide ninguna propiedad que individualmente mantenga cada miembro. Por tanto, la confiabilidad de un sistema distribuido, especialmente en los sistemas guiados por cable, no se basa sólo en la capacidad de sus nodos de detectar, contener y aislar errores antes de que se propaguen, sino que también deben asegurar la continuidad del servicio en situaciones de avería. Estas situaciones o modos de avería se han clasificado en[66]: averías bizantinas, suplantación de otra identidad, averías con parada, transmisiones espurias, averías SOS y pérdida de conexión. Este capítulo se centra en las averías con parada, transmisiones espurias, averías SOS y pérdidas de conexión generadas mediante inyección de fallos a nivel de pin.

Las averías con parada se definen como el tipo de avería más común en un sistema distribuido. Ocurre cuando un nodo o miembro del sistema se desconecta temporalmente del canal de comunicaciones al detectar un error atribuible al propio nodo, comportamiento esperado en componentes "*fail-silence*" como el TTP/C. Aún verificando el correcto comportamiento del nodo que realiza la parada, en la validación también se debe verificar que la desconexión temporal de un nodo es tolerada y el servicio ofrecido mantiene la función del sistema. Además, la reintegración del nodo tras la parada no debe interferir en los nodos activos provocando la desconexión de alguno de ellos. En este capítulo se describen los experimentos que analizan el protocolo TTP ante la parada temporal de un nodo y su posterior reintegración.

La inyección física a nivel de pin con AFIT puede realizar inyecciones controladas que generen transmisiones espurias. Los fallos pueden ser coincidentes con la transmisión de alguna trama o no coincidentes, atribuyéndose a fallos del propio canal. La seguridad que ofrece el protocolo TTP, o cualquier protocolo de comunicaciones, depende de si todas las averías que puedan ocurrir en el sistema son benignas y nunca catastróficas. Una avería catastrófica deriva de una inconsistencia provocada por la falta de consenso entre los miembros del sistema. En este sentido, las transmisiones espurias son la principal causa de la falta de consenso, y más concretamente, aquellas transmisiones que se transforman en averías SOS.

Las averías SOS son un caso particular de averías asimétricas en sistemas de disparo por tiempo. Se producen cuando la recepción del mensaje está delimitada por un patrón

[66] Apartado 2.4.3.

de tiempo o de valor, diferenciando entre SOS en el dominio del tiempo y SOS en el dominio del valor. Cuando la recepción está delimitada por un patrón temporal, donde un mensaje determinado se espera en un instante determinado, una trama que llegue demasiado pronto o demasiado tarde con respecto a ese instante será considerada como incorrecta. Esto implica que el transmisor de la trama y sus receptores deben estar sincronizados, manteniendo un ajuste perfecto de la base de tiempos común. Aún incluyendo ciertos márgenes de error en el ajuste, no sería recomendable validar un sistema crítico asumiendo que no va a existir desviación entre los relojes locales que marcan la apertura, instante de recepción o transmisión y cierre de las ventanas de tiempo de cada nodo. Es precisamente esta desviación, la que puede propiciar averías SOS en el dominio del tiempo. Por ejemplo, una trama podría recibirse dentro del intervalo de tiempo esperado por un nodo mientras que para otro nodo se recibiría demasiado pronto o demasiado tarde. Otro ejemplo sería una transmisión espurias de corta duración o *pulso espurio* precedente a la trama y en el límite de la apertura de las ventanas de recepción, donde sólo algunos nodos detectarán el pulso, mientras que otros abrirán su ventana cuando éste haya desaparecido. Las averías SOS en el dominio del valor más frecuentes ocurren cuando toda o parte la señal analógica recibida no alcanza un nivel lógico bien definido. La interpretación del valor lógico de la señal puede ser diferente en cada nodo ocasionando una conversión asimétrica de los valores de la trama o avería SOS. En este capítulo se analiza el comportamiento observado en el sistema ante averías SOS producidas por fallos a nivel de pin respecto a posibles situaciones de inconsistencia. Del mismo modo, la inyección física a nivel de pin con AFIT proporciona los recursos necesarios para validar el protocolo TTP ante la pérdida de conexión de uno o más nodos. Avería que podría significar la pérdida de consistencia del sistema.

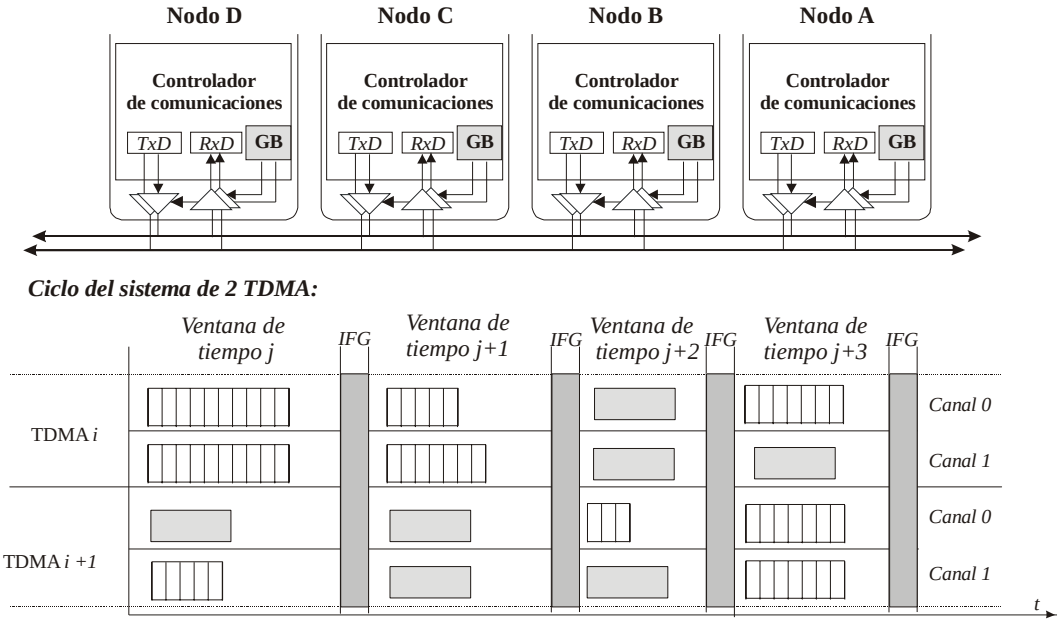
El capítulo se estructura de la siguiente forma. En la introducción se definen algunos conceptos que se utilizan a lo largo del capítulo. En la segunda parte se explican brevemente los servicios del protocolo que pueden tener incidencia en el comportamiento del sistema ante fallos. La tercera parte se centra en la descripción y análisis de los resultados. Finalmente, se concluye el capítulo.

6.1. Introducción

La rigidez de la arquitectura TTA, en cuanto a distribución del tiempo se refiere, hace necesaria una planificación del ciclo completo de transmisiones del sistema (o ciclo del sistema) como parte de las tareas de configuración e implementación que realiza el usuario[67]. Un ejemplo de esta planificación sobre un sistema con cuatro nodos es el que se muestra en la figura 6.1.

El orden de transmisión establecido en el ejemplo es: *C* es **sucesor** de *D*, *B* es *sucesor* de *C*, *A* es *sucesor* de *B* y *D* es *sucesor* de *A*. Otro concepto que se utiliza en la descripción de algunos algoritmos es el de **sucesor del sucesor**. Por ejemplo, *B* sería el sucesor del sucesor de *D*. La distribución del tiempo para envío de mensajes entre un nodo y sus sucesores no es necesariamente homogénea. Así pues, el intervalo de tiempo (o **ventana de tiempo**) asignado al nodo *D* puede ser mayor que el asignado a *B*. Dicho intervalo de tiempo es la **ventana de transmisión**. En una *ventana de tiempo* sólo un nodo tiene permiso para transmitir, todos los demás obligatoriamente reciben la trama en lo que entendemos como una **ventana de recepción**. Además, entre dos ventanas de tiempo consecutivas existe un intervalo durante el cual ni se transmite ni se atiende el canal. Éste es el IFG (“InterFrame Gap”).

[67] Se entiende por usuario aquél que lleva a cabo la planificación del sistema “*x-by-wire*” (o algoritmo de control correspondiente) sobre un protocolo de comunicaciones determinado (por ejemplo TTP)



Ventana de tiempo j en TDMA i
Nodo *D*:
Transmite una trama de datos de 10 bytes replicada
Nodo *A, B,* y *C* escuchan los canales 0 y 1

Ventana de tiempo j+1 en TDMA i
Nodo *C*:
Transmite una trama de datos de 12 bytes entre ambos canales
Nodo *A, B,* y *D* escuchan los canales 0 y 1

Ventana de tiempo j+2 en TDMA i
Nodo *B*:
Transmite tramas de sincronización
Nodo *A, C,* y *D* escuchan los canales 0 y 1

Ventana de tiempo j+3 en TDMA i
Nodo *A:*
Transmite una trama de datos de 7 bytes sin replicar
Nodo *B, C* y *D* escuchan los canales 0 y 1

Ventana de tiempo j en TDMA i+1
Nodo *D*:
Transmite una trama de 5 bytes sin replicar
Nodo *A, B,* y *C* escuchan los canales 0 y 1

Ventana de tiempo j+1 en TDMA i+1
Nodo *C:*
Transmite tramas de sincronización
Nodo *A, B,* y *D* escuchan los canales 0 y 1

Ventana de tiempo j+2 en TDMA i+1
Nodo *B*:
Transmite una trama de datos de 3 bytes sin replicar
Nodo *A, C,* y *D* escuchan los canales 0 y 1

Ventana de tiempo j+3 en TDMA i+1
Nodo *A*:
Transmite una trama de 7 bytes replicada
Nodo *B, C* y *D* escuchan los canales 0 y 1

Figura 6.1. Ejemplo de planificación en el controlador TTP™/C

La validación del protocolo de comunicaciones supone la validación de los servicios del protocolo. Antes de comenzar la descripción de las campañas de inyección realizadas y de los resultados obtenidos, en el siguiente apartado se revisan los conceptos más relevantes sobre los servicios del TTP implementados en el TTP™/C.

6.2. Servicios del Protocolo

Un nodo se puede representar mediante una estructura de capas. En la figura 6.2. se ha representado dicha estructura incluyendo los servicios mínimos relacionados con cada una. Esta estructura compete principalmente al controlador de comunicaciones, aunque la interconexión con la aplicación implica dos capas más desarrolladas en el microcontrolador que soporta la aplicación, la capa de tolerancia a fallos y la propia del algoritmo de control.

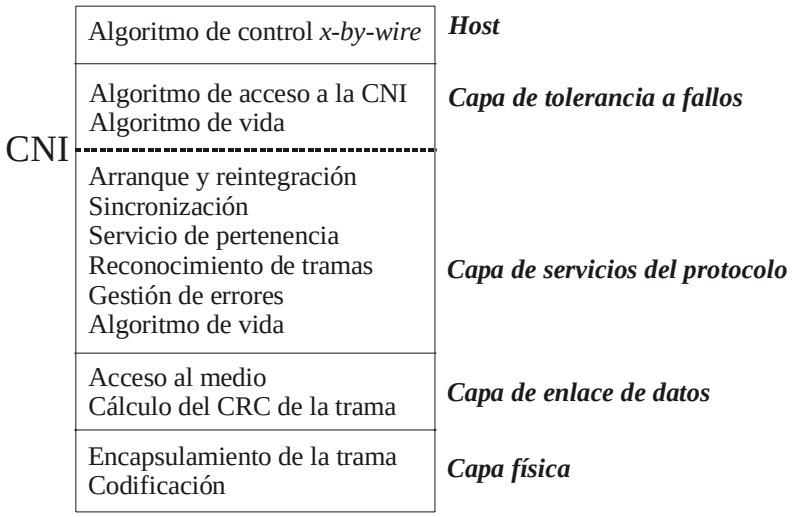


Figura 6.2. Estructuración de las capas de un nodo TTP/C[68]

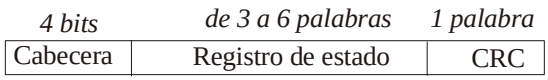
6.2.1. Capa física

Gestiona el acceso al medio, así como el encapsulamiento y la codificación de la trama. Aunque el protocolo no determina ningún encapsulamiento o codificación concreta, sí determina tres requisitos:

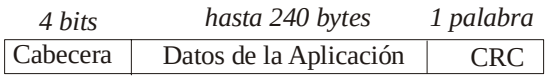
1. Un bus TTA debe estar formado por dos canales físicamente independientes, con sus respectivos niveles físicos.
2. El medio utilizado debe permitir el envío de mensajes del tipo "broadcast".
3. Los límites del retardo de propagación deben ser conocidos.

6.2.2. Capa de enlace de datos

Esta capa gestiona el acceso al medio. Ya que todos los nodos están obligados a transmitir una trama en el periodo asignado, existen dos tipos de tramas: **trama de datos** y **trama de sincronización**:



Trama de Sincronización



Trama de Datos

Figura 6.3. Formato de las tramas en TTP

Una *trama de sincronización I* no contiene datos. Se compone de una cabecera, el registro de estado del nodo y el CRC de la trama (figura 6.3.) que se calcula utilizando los bits de la cabecera y del registro de estado.

Una *trama de datos N* se compone de una cabecera, de los datos (hasta 240 bytes) y el CRC (figura 6.3.) que se calcula utilizando los bits de la cabecera, de los datos y del registro de estado, aunque este último no se envíe explícitamente.

[68] Versión 1.1 de Noviembre de 2003

Al inicio de cada trama se añade un carácter especial de sincronización.

En esta capa también se implementa el **cálculo del CRC**. Utiliza un polinomio generador de grado 16: $x^{16} + x^{13} + x^{12} + x^{10} + x^9 + x^4 + x + 1$. Lo más destacable es que siempre incluye el registro de estado del nodo en el cálculo del CRC.

El cálculo y comprobación del CRC permiten identificar una trama como **trama válida** o **trama inválida**. Una *trama válida* será aquella cuyo encapsulamiento, codificación y cálculo del CRC sean correctos. Cualquier otro caso se considera como *trama inválida*. Debido a que el contenido del registro de estado se utiliza en el cálculo del CRC, este mecanismo de detección de errores no sólo es útil para detectar patrones de error en los bits de la trama, sino que también invalida tramas enviadas por nodos con diferente vector de pertenencia o diferente modo de funcionamiento.

6.2.3. Capa de servicios del protocolo

Diferencia entre servicios de comunicación, servicios críticos y servicios de alto nivel. Entre los servicios de comunicación destacan el arranque y reintegración del controlador, así como la sincronización de la *base de tiempos* común. Entre los servicios críticos, los más importantes son el servicio de pertenencia, la gestión del reconocimiento implícito de las tramas con detección de segregación en subgrupos[69], la gestión de errores y el algoritmo de vida. Finalmente, los servicios a alto nivel incluyen los cambios de modo y la sincronización utilizando una base de tiempos externa si fuera necesario.

6.2.3.1. Arranque y reintegración

Podemos diferenciar entre un **arranque** completo del sistema o una **reintegración**. En el *arranque*, todos los nodos parten desde el mismo estado inicial. Cada nodo envía una o varias tramas de sincronización para ajustar la base de tiempos. Una vez ajustada, se pasa a un estado intermedio durante el cual se *escucha el canal* para comenzar a ejecutar los servicios de comunicación necesarios. Durante una comunicación estable y normal, se dice que el controlador se encuentra en el estado *activo*. Sin embargo, debido a errores internos o del propio canal, dicha comunicación puede desestabilizarse, pudiendo el controlador pasar a dos estados dependiendo del tipo de error detectado (figura 6.4.):

> **Estado pasivo**: El error detectado es leve. El controlador se mantiene a la escucha pero no procesa ningún mensaje. En el momento que el error sea solventado, el controlador tratará de ganar de nuevo su pertenencia.
>
> **Congelado**[70]: El error detectado es grave. Si la aplicación no se encuentra bloqueada puede reiniciar de nuevo al controlador. Para ello debe acceder a una posición específica de memoria y actualizar un valor[71]. Si la aplicación se encuentra bloqueada, será necesario reiniciar primero la propia aplicación. Una de las primeras tareas del sistema operativo será la de acceder a dicha posición de memoria y actualizarla.

Tanto el estado pasivo como la congelación de actividades suponen tareas de reintegración.

[69] Clique detection
[70] Se utiliza el término *congelación* o *estado congelado* por similitud al original “freeze state”
[71] *Flag* CO de la CNI

6.2.3.2. Algoritmo de sincronización

La información contenida en el descriptor de mensajes MEDL de cualquier nodo incluye la división temporal del canal de comunicaciones en las diferentes ventanas de tiempos, qué nodo tiene asignada cada ventana, qué tipo de trama se transmitirá en cada ventana, su longitud y cuál es el instante de tiempo, relativo a la apertura de la ventana, donde se espera comenzar a recibir dicha trama.

La sincronización de los relojes locales con respecto a la base de tiempo común se basa en el cálculo del **factor de corrección** que se aplica al reloj local en cada TDMA. El factor de corrección se calcula estimando la desviación media existente entre el reloj local y los relojes de los nodos que han enviado tramas válidas durante el último TDMA. La desviación existente entre el reloj local y el reloj del nodo que transmite una trama corresponde a la diferencia entre el instante de tiempo en el que se espera recibir la trama y el instante de tiempo en el que ésta se hace efectiva. Dicha diferencia debe ser inferior a cierto valor preestablecido para que sea incluida en el cálculo del factor de corrección. En caso contrario, la trama se desestima y se considera como trama inválida. Además, es posible desestimar también hasta dos tramas cuyos instantes de recepción tengan una desviación máxima con relación a la esperada, una por exceso y otra por defecto, siempre que el número de tramas restantes sea mayor de dos. La desviación media de las restantes debe ser inferior a $\Pi / 2$, siendo Π la *precisión del reloj* definida por el usuario.

6.2.3.3. Servicio de pertenencia

Todas las tramas llevan implícita o explícitamente el registro de estado del nodo. Este registro incluye el vector de pertenencia que se actualiza de la siguiente forma:

1. Después de transmitir una trama, el nodo compara su vector de pertenencia con el transmitido por su sucesor. Si coinciden[72], termina el algoritmo.
2. Si no coinciden, espera a recibir una segunda trama, la del sucesor de su sucesor. El nodo realiza una segunda comparación de su vector de pertenencia con el transmitido en esta segunda trama. Si coincide, marca la trama recibida de su sucesor como inválida y termina el algoritmo.
3. Si no coincide, compara los vectores de pertenencia de las dos tramas recibidas. Si son iguales, el nodo se marca a sí mismo como miembro no activo.
4. Si no son iguales, el nodo marca como miembros no activos a los nodos transmisores de ambas tramas.

6.2.3.4. Reconocimiento implícito

Cada nodo mantiene tres contadores: número de tramas válidas, inválidas o silencios percibidos en el bus[73]. Después de transmitir, el nodo inicia a cero los tres contadores.

[72] Si la trama recibida es de tipo I, el registro de estado se envía explícitamente y se puede realizar una comparación directa. Si por el contrario la trama es de tipo N, el registro de estado habrá sido empleado para el cálculo del CRC pero no se envía explícitamente. En este caso, el receptor asume que el registro de estado del nodo transmisor debe ser idéntico al suyo, y lo utiliza para realizar la comprobación del CRC, dependiendo de cuyo resultado se dará la trama como válida o no. Una trama válida equivale a la coincidencia de los vectores de pertenencia.

[73] La transmisión de una trama en la ventana de tiempo asignado es obligatoria. Además, todos los nodos están obligados a recoger y analizar la trama que se transmite en cada ventana (ventanas de

Cuando se recibe una trama válida, se incrementa el primer contador; si la trama es inválida, el segundo, y si no se recibe trama alguna cuando se esperaba recibir, el tercero. El reconocimiento implícito a una transmisión está ligado a la detección de una segregación del conjunto de todos los nodos del sistema en varios subconjuntos. La segregación es consecuencia directa de una falta de consenso a la hora de decidir si la última trama que transitó por el canal de comunicaciones fue válida o no. El protocolo TTP califica en sus especificaciones a la segregación como un estado inseguro que puede infringir la hipótesis de fallos. Por tanto, el protocolo trata de evitar la segregación mediante un algoritmo denominado "*clique avoidance*".

> **Versión v0.1.** El nodo, antes de su siguiente transmisión, realiza una comparación: si el número de tramas válidas recibidas no es estrictamente mayor que la suma de las inválidas más silencios percibidos, asume que ocurrió un error con la última trama que envió.
>
> **Versión v1.0. y v1.1.** El contador de tramas válidas se inicia a uno si al ejecutar el servicio de pertenencia el nodo permanece como miembro activo. Los contadores no se actualizan mientras el servicio de pertenencia esté en proceso. Además, antes de la siguiente transmisión el nodo realiza una comparación: si el número de tramas válidas recibidas no es estrictamente mayor que las inválidas asume que ocurrió un error con la última trama que envió. En el arranque o reintegración, el contador de tramas válidas se inicia a dos[74].

El reconocimiento implícito requiere que más de la mitad de los nodos conectados al canal reconozcan individualmente la validez de la trama. Ante la falta de consenso, en la que algunos nodos aceptan la trama mientras que otros la rechazan, es la mayoría la que gana y la que impone su criterio al resto de los nodos, manteniendo así la consistencia del sistema y evitando la segregación.

6.2.3.5. Gestión de errores

El controlador puede detectar cuatro tipos de error, dependiendo de si son errores derivados de algún servicio del protocolo de comunicaciones, de la pérdida de pertenencia como miembro activo del sistema, si son derivados de la interconexión con el nivel de aplicación o si son errores internos de funcionamiento.

- Un error en el *protocolo de comunicaciones*[75] corresponde a la detección de un error relativo a las comunicaciones. Son los derivados del algoritmo de sincronización, del reconocimiento de la trama o de la pérdida de conexión. Este último ocurre cuando el controlador supone que es el único nodo activo del sistema. Este tipo de errores dejan al controlador en un estado en el que se congela temporalmente cualquier actividad.

- La *pérdida de pertenencia*[76] como miembro activo del sistema se gestiona de forma diferente a cualquier otro error derivado de los servicios del protocolo. El servicio de pertenencia está oriento a detectar errores ocurridos durante la última transmisión del nodo. En caso de error, el controlador pasa a estado

recepción) a excepción de la suya propia. Por tanto, la percepción de un silencio significa que no se ha recibido ninguna trama durante una ventana de recepción.

[74] Esta modificación se decidió tras obtener los resultados de las inyecciones realizadas con VFIT en el TTP/C-C1 [Blanc *et al*. 2004].

[75] Protocol error

[76] Membership service

pasivo y trata de reintegrarse lo antes posible.

- Un error derivado de la *sincronización con el nivel de aplicación*[77] indica un problema en la aplicación o con el sistema operativo. Si la aplicación no es capaz de mantener su funcionalidad, el controlador pasa a un estado pasivo. En el momento que la aplicación se restablezca, tratará de reintegrarse activamente en el sistema lo antes posible.

- Un *error interno* puede detectarse por ciertos mecanismos internos implementados a tal efecto[78]. El más importante es el temporizador de guardia, pero también incluye los errores debidos a la comprobación de un CRC incorrecto en el descriptor de mensajes o un error detectado por el guardián del bus. Este tipo de errores dejan al controlador en un estado en el que se congela temporalmente cualquier actividad.

6.2.3.6. Algoritmo de vida

Se utiliza para realizar una comprobación periódica del estado de la aplicación. Si la aplicación hubiera quedado bloqueada por cualquier motivo, el controlador debería parar la transmisión, puesto que los datos a transferir son probablemente inválidos.

Este algoritmo designa dos registros del *área de registros y control de estado* de la CNI cuyo valor debe ser actualizado periódicamente siguiendo cierta estrategia.

1. Un registro es el destinado al testigo de vida del controlador de comunicaciones. El otro registro es el destinado al testigo de vida de la aplicación.
2. En cada TDMA la aplicación debe actualizar su testigo. El momento de la actualización se define por programa, pero debe ser antes de la ventana de transmisión asignada.
3. La aplicación lee el testigo de vida del controlador, invierte su valor y escribe el resultado como su propio testigo de vida.
4. Durante el IFG, el controlador debe comprobar y actualizar ambos testigos. Primero comprueba que la aplicación ha actualizado correctamente su valor. Si es correcto, limpia el valor del registro a cero. Si no es correcto, para la transmisión y pasa a estado pasivo. Segundo, actualiza su propio registro utilizando un valor modificado de los 16 bits más bajos del registro de estado.

6.2.4. Capa de tolerancia a fallos

Esta capa se implementa a nivel de aplicación en el microcontrolador que ejecuta el algoritmo de control. Gestiona el intercambio de mensajes entre el controlador de comunicaciones y la aplicación, así como la actualización periódica del registro de la CNI destinado al testigo de vida de la aplicación.

En los prototipos TTP/C-C1 y C2, el acceso a la CNI para el intercambio de mensajes entre el controlador de comunicaciones y la aplicación se realiza siguiendo un algoritmo de acceso a memoria **NBW** (“Non-Blocking Write Protocol”):

1. Se designan dos registros en memoria. Uno de lectura para el controlador

[77] Host error
[78] Built-In-Self Error, BIST

y de lectura/escritura para la aplicación. Otro de lectura para la aplicación y lectura/escritura para el controlador.
2. El valor contenido en ambos registros indica si se está realizando una escritura en ese momento y por parte de quién.
3. Si la aplicación está escribiendo en memoria, el controlador debe esperar, si es posible, a que ésta acabe. Si no puede esperar, activará un error de protocolo.
4. Si el controlador está escribiendo en memoria, la aplicación debe esperar, si es posible, a que éste termine. Si no puede esperar, el sistema operativo generará un error.
5. Si durante el acceso de la aplicación a memoria la escritura se invalida por alguna causa, la aplicación escribirá en su registro un valor predefinido de error.
6. Antes de realizar un acceso a memoria, el controlador debe leer el registro de la aplicación. Si contiene un error, pasará a estado pasivo.

6.3. Comportamiento del Protocolo TTP ante Averías

Después de haber realizado un rápido barrido por los servicios implementados en el controlador TTP™/C nos centramos en la validación del protocolo de comunicaciones. La inyección de fallos a nivel de pin facilita la observación del comportamiento del protocolo de comunicaciones TTP ante diferentes tipos de avería mediante la introducción deliberada de un conjunto de fallos cuya ocurrencia afecta directamente a la transmisión o recepción de mensajes.

Los fallos se localizan en las líneas de control, línea de recepción y línea de transmisión para el prototipo TTP/C-C1 y en el oscilador del guardián del bus y líneas de recepción y transmisión para el prototipo C2. Las observaciones y resultados derivados de estos experimentos son tratados en [Blanc *et al*. 2002 (b) y Blanc *et al*. 2004].

6.3.1. Averías con parada y reintegración

Si un nodo detecta una avería interna, ya sea debida a un error en la aplicación o a un error en el protocolo de comunicaciones, cualquier intento de transmisión o recepción será abortado por el nodo hasta el restablecimiento de su funcionalidad. El momento más crítico será cuando la *avería con parada* se produzca durante una transmisión ya iniciada. Además, ésta puede implicar un intento posterior de reintegración, que dependiendo del estado en el que se estabilice el controlador (figura 6.4.), deberá efectuarse mediante tramas de arranque en frío[79] o mediante la observación del canal de comunicaciones.

Para que un nodo pueda reintegrarse de nuevo es necesario que alcance el estado de *activo*. Sólo existen dos posibles transiciones que lleven al controlador a un estado de actividad normal. Son el arranque en frío y la observación del canal desde el estado *pasivo* (figura 6.4.). El arranque en frío se considera al efectuarse una reiniciación completa del nodo. Normalmente la iniciación del nodo se asocia al encendido o puesta en marcha del sistema, aunque también ocurre cuando la avería interna que provocó la parada dejó al nodo en el estado de *congelado*[80], como por ejemplo, la detección de un error de protocolo o un error interno. La transición al estado *pasivo* se debe o bien a la pérdida de sincronismo entre protocolo de comunicaciones y la aplicación, o bien a la pérdida de pertenencia del nodo con respecto al sistema. Una vez restablecido el

[79] Cold start frames
[80] Freeze state

sincronismo con la aplicación, el nodo observa el canal de comunicaciones e intenta su reintegración transmitiendo en la misma ventana que ocupaba antes de la detección del error. Mediante validación, es importante asegurar que en ambos casos la reintegración del nodo es segura y no conlleva ninguna avería asociada.

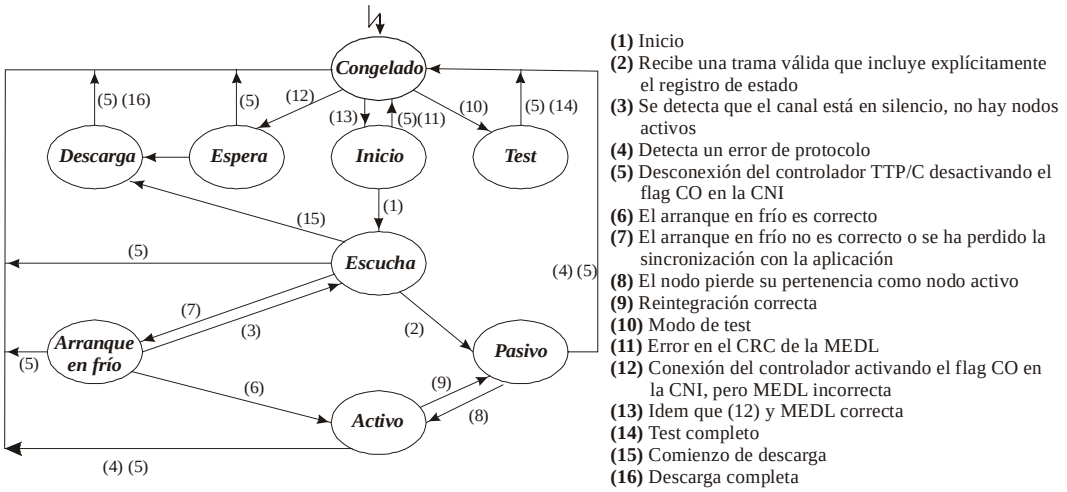
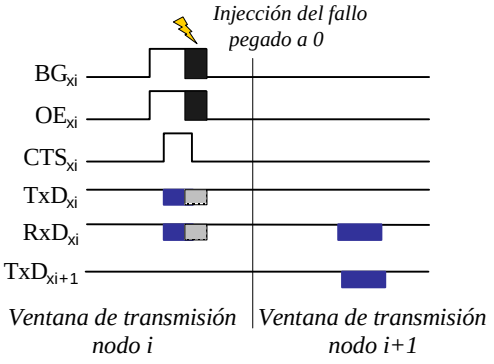


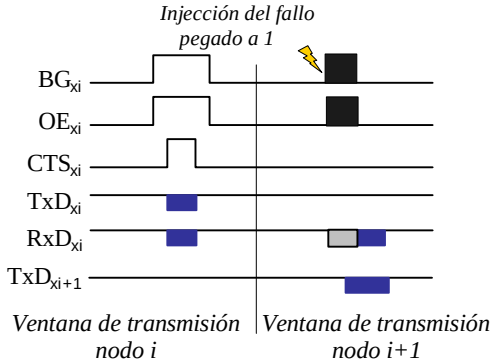
Figura 6.4. Estados y transiciones del controlador TTP/C

6.3.1.1. Parada durante una transmisión ya iniciada

Una vez habilitada la transmisión, las señales de control BG ("Bus Guardián") y OE ("Output Enable") se mantienen habilitadas durante un periodo superior al necesario para el volcado de la trama en el canal e inferior a la duración completa de la ventana de transmisión. La variación de los márgenes temporales de cualquiera de estas señales mediante fallos de pegado, a 0 o a 1, localizados en las líneas BG0 y BG1 o OE0 y OE1, tiene como consecuencia la interrupción del volcado de la trama (ver figuras 6.5. y 6.6. izquierda) o la habilitación de la transmisión fuera de la ventana de tiempo asignada (ver figuras 6.5. y 6.6. derecha), caso en el que se favorece la aparición de fallos espurios coincidentes con la ventana de transmisión de otro nodo.

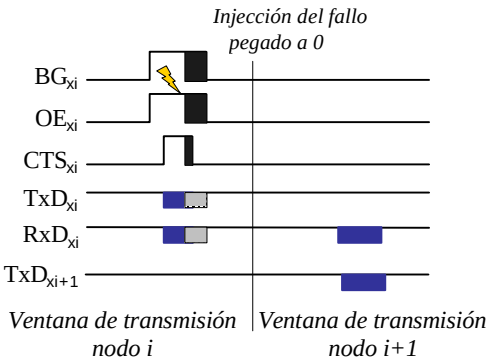


El fallo acorta el pulso de la línea BG interrumpiendo el volcado de la trama en el canal.

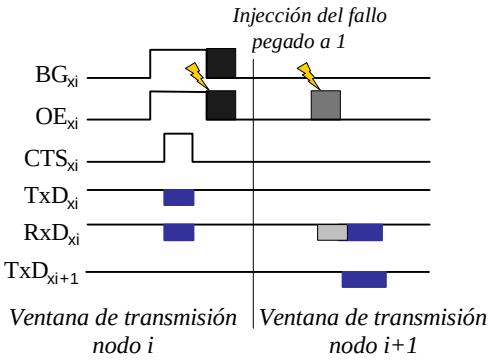


Habilita la transmisión fuera de la ventana asignada. Posibles fallos espurios en la recepción.

Figura 6.5. Representación de los fallos inyectados sobre la línea BG de control de la transmisión en el canal$_x$



El fallo acorta el pulso de la línea OE interrumpiendo el volcado de la trama en el canal.



En negro: El fallo alarga el pulso de la línea OE
En gris: Habilita la transmisión fuera de la ventana asignada. Posibles fallos espurios en el canal.

Figura 6.6. Representación de los fallos inyectados sobre la línea OE de habilitación de la transmisión en el canal$_x$

En las figuras 6.5. y 6.6. se representa el efecto de un fallo simple inyectado sobre las líneas BG$_x$ o OE$_x$, pudiendo ser x cualquiera de los dos canales del bus. Mientras el fallo se localice sólo en uno de los dos canales se tolera, ya que la trama afectada es marcada como *trama inválida* y se utiliza la trama volcada sobre el otro canal libre de fallos. Sin embargo, cabe resaltar las siguientes consideraciones:

- Se ha observado que los pegados simples en estas líneas pueden producir fallos transitorios conexos, con el mismo efecto que un fallo doble, que afecta internamente a ambos canales. Esta observación se extrae de los experimentos que implican la habilitación de la transmisión fuera de la ventana de tiempo asignada.

- Si el fallo habilita la transmisión durante la recepción de una trama, se producen fallos espurios en la línea de recepción del nodo inyectado, desvirtuando la trama recibida que queda marcada como *trama inválida*. Si el fallo afecta a ambos canales, las dos tramas recibidas, una por cada canal, quedan marcadas como inválidas y el nodo inyectado modifica su vector de pertenencia excluyendo al nodo transmisor. En ese momento, el nodo inyectado adquiere una visión del estado del sistema minoritaria, debido a que el resto de los nodos no modifican su vector de pertenencia al no afectar el fallo a su recepción. Todas las tramas que el nodo inyectado reciba a partir de ese momento son marcadas como inválidas ya que, al contener un registro de estado distinto al suyo, el cálculo del CRC detecta siempre un error. Según el *reconocimiento implícito* implementado en el protocolo, antes de que el nodo comience la siguiente transmisión debe verificar que su visión del estado del sistema coincide con la mayoría[81]. En caso de no coincidencia, el nodo detecta un error de protocolo y pasa a un estado de congelación de actividades.

- El reconocimiento implícito no siempre puede detectar una situación de minoría. Si la transmisión se realiza antes de medio TDMA desde la

[81] Clique avoidance

aparición del fallo, no existen suficientes tramas inválidas recibidas como para que el nodo perciba que su visión del sistema no es mayoritaria. Por tanto, transmite, aunque su transmisión no tiene ningún efecto negativo puesto que es marcada como inválida por todos los receptores y, en menos de un TDMA, el nodo ejecuta el *servicio de pertenencia* detectando su exclusión por parte de la mayoría de los nodos y pasando a un estado pasivo.

Durante los experimentos realizados con cuatro nodos y pegados simples se detectó que 10 de cada 500 fallos efectivos activan un error de protocolo, independientemente de la línea afectada. Estos errores son causados por fallos transitorios conexos derivados de la generación de un pulso inexistente. No se detectó ningún error con el servicio de pertenencia.

Con fallos que supongan la interrupción del volcado de la trama en ambos canales o la transmisión de una trama no correcta (fallos espurios), es el servicio de pertenencia el que detecta el error ocasionado por el fallo. El nodo pasa a un estado pasivo tras la detección.

Los fallos dobles coincidentes con la ventana de transmisión del nodo provocaron, en todos los casos, la exclusión del nodo al ejecutar el servicio de pertenencia, quedando en un estado pasivo, mientras que los coincidentes con alguna recepción provocaron un error de protocolo, dejando al nodo congelado.

6.3.1.2. Reintegración del nodo

Según las especificaciones del protocolo, un nodo sólo se reintegra en el sistema a través del estado de arranque en frío[82] cuando, durante un tiempo preestablecido, no recibe ninguna trama válida. Por lo tanto, este supuesto se producirá cuando todos los nodos conectados al canal de comunicaciones comiencen la reintegración al mismo tiempo. Al inicio de cualquiera de los experimentos de inyección física de los llevados a cabo se fuerza la línea de *reset* de todos los nodos del sistema. Nunca se ha observado que dicha acción provoque la exclusión o la incorrecta reintegración de alguno de ellos.

Para cualquier otro supuesto de reintegración, el nodo escucha primero el canal de comunicaciones. Tras la recepción de alguna trama que incluya explícitamente el registro de estado del transmisor (tramas de tipo I de sincronización) o de alguna trama de arranque en frío (existe otro nodo más rápido que ya ha comenzado la reintegración desde el estado de arranque en frío) el nodo pasa a estado pasivo.

Por tanto, una premisa básica no especificada para permitir la reintegración de un nodo es la de establecer por ciclo al menos una trama de tipo I transmitida por un nodo diferente al que trata de reintegrarse. Como el número de nodos mínimo especificado para mantener la base de tiempos sincronizada es de dos, para un sistema de n nodos sería posible reintegrar al mismo tiempo hasta n-2 nodos, siempre y cuando n-1 envíen una trama I por alguno de los canales y ciclo.

Bajo esta premisa, los cuatro nodos del sistema implementado transmiten al menos una trama de tipo I por ciclo. Las pruebas realizadas incluyen la reintegración desde el estado pasivo debido a una pérdida de pertenencia (detección de un error en el algoritmo de vida o transmisión de una trama inválida), o bien la reintegración desde el estado de escucha[83] (ver figura 6.4.) de un solo nodo o de dos nodos al mismo tiempo. En el primer

[82] Cold start state
[83] Listen state

caso, a excepción de un error de implementación en el algoritmo de vida que se detallará en el capítulo siguiente, en una muestra de casi 5.000 fallos efectivos no se detectó que la reintegración de un único nodo supusiera la exclusión de otro nodo del sistema. En el segundo caso, partiendo desde el estado de escucha, la reintegración de un nodo no provoca ninguna colisión. Sin embargo, sólo en 4 de cada 1.500 experimentos se alcanza la reintegración en un TDMA, siendo la parada y proceso de reintegración transparente al resto del sistema. Por tanto, es recomendable la replicación de aquellos nodos que soporten algoritmos de control críticos, ya que si el nodo sufre una avería con parada, el tiempo de restablecimiento puede ser superior al ciclo completo de transmisiones del sistema.

Por otro lado, la reintegración de dos nodos tampoco provoca exclusión o pérdida de sincronismo entre los nodos operacionalmente activos, aunque tampoco en este caso la parada resulta transparente, todo lo contrario. En los experimentos realizados, se utilizó un nodo con osciladores de peor precisión que el resto. Cuando dicho nodo comenzaba la reintegración al mismo tiempo que otro, se observó en un alto número de experimentos que el nodo generaba un error de protocolo antes de ganar el canal, cuya explicación radica en la ejecución del algoritmo de sincronización de la base de tiempos aún estando en estado pasivo. En dicho algoritmo, de todas las tramas recibidas se descartan hasta dos cuyos instantes de recepción tengan una desviación máxima con relación a la esperada, una por exceso y otra por defecto. La desviación media de las restantes debe ser inferior a $\Pi / 2$ (la mitad de la precisión definida por el usuario, en nuestro caso 2,4μs). Si la desviación es mayor, el nodo activa un error de protocolo. Por tanto, en un sistema con tan solo dos nodos operacionales y otros dos en estado pasivo, las tramas recibidas por los nodos operacionales no son descartables (son de obligada utilización en el algoritmo de sincronización) y la probabilidad de que tengan una desviación mayor que $\Pi / 2$ aumenta con osciladores de baja precisión. Para evitar estos problemas, las especificaciones proponen realizar la sincronización de la base de tiempos utilizando sólo las tramas de los nodos con osciladores similares llamados “relojes maestro”[84], al resto se les denomina “relojes esclavo”[85]. Lo que no se contempla en las especificaciones es que esta solución, para resolver el problema, también implica, al igual que la replicación, el aumento del número de nodos. Por ejemplo, cuando entre los *n*-2 nodos en estado pasivo se encuentra un nodo con reloj esclavo, sigue siendo posible que su algoritmo de sincronización obtenga una desviación media de las dos tramas recibidas superior a $\Pi / 2$, desviación que se va corrigiendo a medida que aumenta el número de nodos operacionales. Además, si los *n*-2 nodos en estado pasivo son relojes maestro y los dos nodos operacionales son relojes esclavo, la reintegración es inviable, reduciendo el número de nodos que pueden mantenerse en estado pasivo al mismo tiempo a *n*-1 relojes maestro. Con todo esto, parece más interesante la replicación de los nodos críticos independientemente de la precisión de sus osciladores.

6.3.2. Transmisiones espurias y averías SOS

La inyección de fallos a nivel de pin es capaz de generar transmisiones espurias sintáctica o semánticamente incorrectas. Los fallos pueden ser coincidentes con la transmisión de alguna trama, causando una alteración en la misma, o no coincidentes, atribuyéndose a fallos del propio canal o de los componentes pasivos del nivel físico.

El análisis de las transmisiones espurias se podría dividir en dos partes. En primer lugar, validar el guardián del bus local. En segundo lugar, validar el protocolo de comunicaciones.

[84] Master clocks
[85] Slave clocks

Cuando un fallo interno causa un error en el controlador de comunicaciones que pueda suponer el envío de una trama fuera de su ventana de transmisión o fuera de los márgenes temporales correctos de la ventana asignada, es el guardián del bus el que debe evitar la transmisión y aislar el error. En caso de no evitar la transmisión, en el peor de los casos se produciría una colisión en ambos canales. Si la colisión se produce en un solo canal, la trama afectada se marca como inválida y se utiliza la réplica enviada por el otro canal, siempre que la trama se envíe replicada[86]. Si no estamos ante una colisión, sino ante un envío fuera de los márgenes temporales correctos de la ventana asignada, el nodo debería ser excluido al ejecutar el servicio de pertenencia.

6.3.2.1. Transmisiones espurias sobre un canal

Una transmisión espuria, en relación a la transmisión de un nodo concreto, puede ocurrir en tres momentos diferentes: durante la ventana de transmisión del nodo, durante el IFG ("InterFrame Grap") o espacio entre dos ventanas y durante la ventana de transmisión de otro nodo.

En el primer caso, tanto si el fallo coincide con la transmisión de la trama (caso (a) figura 6.7.) o después de ésta, en la mayorías de las ocasiones se obtiene una trama sintácticamente incorrecta, siendo el nodo transmisor excluido como nodo activo por el algoritmo de pertenencia. Sólo en un bajo porcentaje de las ocasiones (1 sobre 1.000) la trama sigue siendo sintácticamente correcta, aunque semánticamente incorrecta, y el error no es detectado por el cálculo del CRC. Podemos decir, por tanto, que la cobertura de detección de tramas erróneas debidas a fallos en el canal o componentes pasivos es del 99,9%.

Un fallo dentro de los límites del IFG (caso (b) figura 6.7.) no tiene efecto ninguno sobre el sistema. En el protocolo TTP, durante el IFG los nodos no atienden los posibles cambios de nivel en el canal de comunicaciones. En principio este tipo de fallos podría parecer irrelevante. Sin embargo, como se verá en el apartado 6.3.2.3, esta característica del protocolo puede producir averías SOS en el dominio del tiempo.

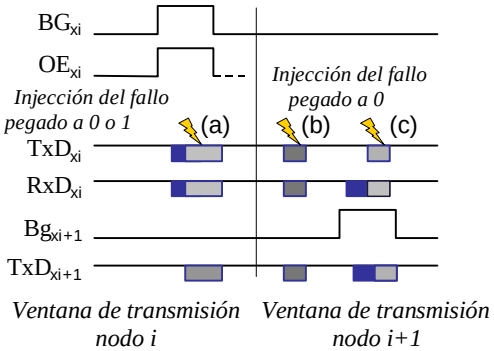


Figura 6.7. (a) Fallo coincidente con la transmisión de la trama; (b) fallo coincidente con el IFG; (c) fallo coincidente con una ventana de transmisión diferente a la asignada

Un fallo coincidente con una ventana de transmisión diferente a la asignada (caso (c) figura 6.7.) no debería tener incidencia en el sistema conteniéndose en el nodo gracias a la barrera del guardián del bus. Durante los experimentos realizados se observó la dependencia que presenta el correcto funcionamiento del guardián del bus local de los

[86] En el TTP no es obligatorio el envío de las tramas replicadas. Es posible enviar dos tramas distintas, una por cada canal, con el fin de optimizar la utilización del bus.

componentes pasivos del nivel físico. Por ejemplo, cuando la salida TTL del controlador se conecta al transceptor PCA82C250, la vía de salida del TTP nodo se mantiene habilitada indefinidamente. Cualquier variación de tensión en dicha vía provocada por un fallo se proyecta sobre el canal de comunicaciones colisionando con la trama en curso. Además, el nodo transmisor de la trama colisionada es excluido del canal como erróneo, no siendo el origen del fallo. Si el fallo es intermitente o suficientemente largo para afectar a dos ventanas de tiempo diferentes, las consecuencias son peores, pudiendo producirse la desconexión en cadena de todos los nodos[87]. Para evitar el mal funcionamiento del guardián del bus en su tarea de contención, es necesario que los componentes utilizados en el nivel físico de comunicaciones puedan ser directamente controlados y habilitados por el guardián local en una topología de bus. No ocurre lo mismo en una topología de estrella, donde el guardián de bus central hace las veces de filtro.

6.3.2.2. Transmisiones espurias conexas

Al igual que en los experimentos de inyección realizados sobre las líneas de control BG y OE, también en esta ocasión se detectaron errores de protocolo causados por fallos transitorios conexos derivados de pegados simples en la línea de transmisión, aunque a diferencia del caso anterior, los fallos eran coincidentes con la ventana asignada al nodo.

También es posible simular fallos conexos con AFIT mediante pegados dobles. Estos fallos son los más agresivos puesto que conducen a la exclusión segura del nodo como miembro activo. Siempre que uno de los dos canales de comunicación se mantenga libre de fallos, aunque una trama fuera rechazada como inválida, quedaría la otra trama para la ejecución del algoritmo de sincronización, pertenencia y reconocimiento implícito. Básicamente, el impacto se reduce al nivel de aplicación que no recibirá el valor de las variables esperadas durante ese ciclo. Sin embargo, el rechazo de ambas tramas debido a un fallo conexo hace que el reconocimiento implícito de la trama transmitida sea negativo y por tanto el nodo excluido. Estos fallos ponen a prueba la fiabilidad del protocolo de comunicaciones.

Durante la validación del protocolo se ha observado que la apertura de las ventanas de transmisión de los nodos y el inicio de la transmisión de cada trama son puntos débiles en la arquitectura TTA, ya que puede generan situaciones no especificadas. Estas situaciones son averías SOS ("Slightly-Off-Specifications")[88] en el dominio del tiempo derivadas de transmisiones espurias conexas (localizadas en ambos canales).

6.3.2.3. Observación de averías SOS en el dominio del tiempo

Se ha observado que son averías críticas aquellas transmisiones espurias que ocurren entre la apertura de una ventana de tiempo y el instante en el que se espera comenzar a recibir la trama correspondiente. En la figura 6.8. para la versión 0.1 del 1 de Febrero de 1999 y la figura 6.9. para las versiones 1.0 del 4 de Julio de 2002 y 1.1 del 19 de Noviembre de 2003, se muestran los parámetros implicados en el algoritmo de sincronización. Los instantes $t_{AT\ s}$ y $t_{AT\ r}$ marcarían el inicio de la transmisión de una trama o de la recepción de una trama respectivamente. Sin embargo, teniendo en cuenta la existencia de retardos, tanto en las aperturas de las ventanas como el retardo propio del medio físico, se indica con $t_{AT'\ s}$ y $t_{AT'\ r}$ el inicio esperado para el comienzo de una transmisión o de una recepción respectivamente. Por tanto, la apertura de las ventanas de

[87] El término empleado en el proyecto FIT para hacer referencia a este efecto es "cluster shutdown"
[88] Apartado 2.4.3.3.

tiempo debe ser anterior a $t_{AT\ s}$ en el caso del transmisor y $t_{AT\ r}$ en el caso del receptor. Es entre dicha apertura y $t_{AT'\ s}$ o $t_{AT'\ r}$ donde se ha observado que la ocurrencia de transmisiones espurias puede dar lugar a averías críticas para el protocolo de comunicaciones TTP.

Observando las diferencias entre la primera versión y la modificación posterior se puede pensar que ésta es debida a la nueva orientación del algoritmo para funcionar con un guardián del bus central, en cuyo caso la trama no se transmitiría en el instante predefinido, sino que se aplica un retardo que favorece la apertura de las ventanas de recepción antes de que comience el envío. Con un guardián del bus central, este retardo no lo estimaría directamente el transmisor, sino el propio guardián del bus. Por otro lado, se amplia $\Delta_{rw\ r}$ a $\pm 2\Pi$, bajo la suposición de que con una topología de estrella el retardo entre la transmisión y recepción será mayor que con una topología de bus. Esta diferencia se debe, no sólo al incremento del cableado ya previsto con Δ_{prop}, sino a que la señal recibida por la estrella, antes de ser enviada a los receptores, es rastreada bit a bit con el fin de detectar posibles valores indeterminados definidos como *tensiones de entrada fuera de los rangos establecidos para fijar un nivel lógico acorde con la tecnología utilizada*. A las acciones descritas llevadas a cabo por el guardián del bus y orientadas a asegurar la recepción completa de la trama por todos los nodos y a eliminar posibles valores indeterminados debidos a fallos en la transmisión se les denomina "***signal reshaping***".

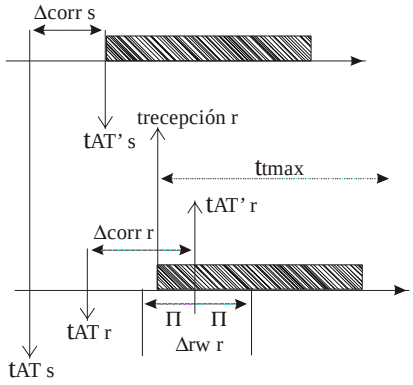


Figura 6.8. Cronograma de transmisión – recepción de una trama en v0.1 de 1999

- $t_{AT\ s\ (emisor)\ o\ r\ (receptor)}$ indica el instante de transmisión o recepción predefinido para trama y ciclo.
- $t_{AT'\ s\ (emisor)\ o\ r\ (receptor)}$ indica el instante de transmisión o recepción retrasado según un retardo conocido para trama y ciclo.
- $t_{recepción\ r}$ indica el instante de recepción real para trama y ciclo.
- t_{tMAX} indica el tiempo máximo necesario para la transmisión de la trama.
- $\Delta_{rw\ r}$ indica el intervalo en el que se espera comenzar a recibir la trama.
- Δ_{corr} indica el posible retardo máximo del canal de comunicaciones.

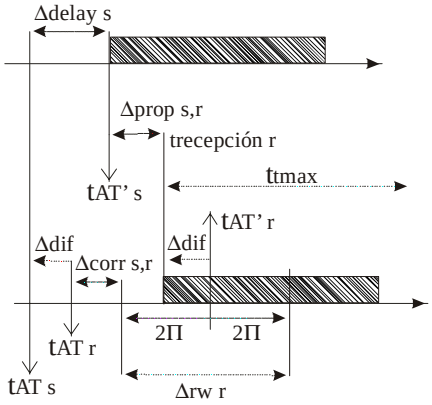


Figura 6.9. Cronograma de transmisión – recepción de una trama en v1.0 de 2002

- $t_{AT\ s\ (emisor)\ o\ r\ (receptor)}$ indica el instante de transmisión o recepción predefinido para trama y ciclo.
- $t_{AT'\ s\ (emisor)\ o\ r\ (receptor)}$ indica el instante de transmisión o recepción retrasado según un retardo conocido para trama y ciclo.
- $t_{recepción\ r}$ indica el instante de recepción real para trama y ciclo.
- t_{tMAX} indica el tiempo máximo necesario para la transmisión de la trama.
- $\Delta_{rw\ r}$ indica el intervalo en el que se espera comenzar a recibir la trama.
- Δ_{dif} indica la diferencia en las apertura de las ventanas de recepción.
- $\Delta_{corr\ s,r}$ indica el factor de corrección aplicable al retardo esperado desde que la trama debería ser enviada hasta que realmente se envía. Este factor se predefine en la MEDL y puede valer 0.
- $\Delta_{delay\ s}$ indica el retardo que fuerza el emisor de una trama desde que debe transmitir hasta que la transmite realmente. El objetivo de este retardo es garantizar que todas las ventanas de recepción estén abiertas cuando la trama sea volcada al canal.
- $\Delta_{prop\ s,r}$ indica el retardo de propagación del canal entre emisor y receptor. Se debe cumplir que :

$$\Delta_{prop\ s,r} + \Delta_{delay\ s} = \Delta_{corr\ s,r} + 2\Pi \quad \text{siendo} \quad \Delta_{corr\ s,r} \geq 0 \qquad (6.1)\ [\text{TTP 2004}]$$

Durante los experimentos, se observó la existencia de ciertos escenarios favorables para la ocurrencia de averías críticas. Los escenarios son los siguientes:

- En las figuras 6.8. y 6.9. se aprecia que existe un margen de error de ±Π o ±2Π en el comienzo de la recepción de la trama. Aunque $t_{AT'\ r}$ sea el instante de recepción esperado, una trama cuyo comienzo se detecte entre $t_{AT'\ r} - \Pi$ y $t_{AT'\ r} + \Pi$ (o $t_{AT'\ r} - 2\Pi$ y $t_{AT'\ r} + 2\Pi$) será aceptada y considerada como válida siempre que cumpla con el encapsulamiento, codificación y se compruebe el CRC. Por tanto, el primer escenario plantea la ocurrencia de una transmisión espuria recibida entre la apertura de la ventana de recepción y $t_{AT'\ r} - \Pi$ ($t_{AT'\ r} - 2\Pi$) (ver figura 6.10.).
- El segundo escenario plantea la ocurrencia de una transmisión espuria recibida entre $t_{AT'\ r} - \Pi$ ($t_{AT'\ r} - 2\Pi$) y el cierre de la ventana.
- El tercer escenario es una extensión del primero donde se considera la ligera desviación temporal existente entre los relojes locales.
- El cuarto escenario es una extensión del segundo, donde también se considera la ligera desviación temporal existente entre los relojes locales.

El tercer y cuarto escenario surgen debido a considerar que los algoritmos utilizados

para implementar cualquier *sincronización interna*[89] de la base de tiempos común llevan inherente una desviación relativa de los relojes locales. Esta ligera desviación de los relojes locales hace que las ventanas de recepción no se abran exactamente en el mismo instante de tiempo, pudiendo ocurrir averías SOS en el dominio del tiempo ante la ocurrencia de ciertas transmisiones espurias. Por ejemplo, un pulso espurio transmitido entre la apertura de la ventana de transmisión y $t_{AT'\,s}$, puede que sea recibido tan solo por algunos nodos, los más rápidos, cuya ventana de recepción se abra antes de desaparecer la perturbación, mientras que los nodos más lentos, cuya ventana se abra después de desaparecer la perturbación, no tendrán registrada la ocurrencia de la avería (ver figura 6.13.). Esta avería SOS es una percepción asimétrica de un evento, donde se puede perder el consenso del sistema al marcar, parte de los nodos, como erróneo al nodo transmisor mientras que otra parte entiende que la transmisión ha sido correcta. En el caso del TTP/C, la falta de consenso se evita mediante una estrategia NGU donde la visión mayoritaria es la que establece el estado del sistema independientemente de qué o quién produjo la transmisión espuria.

Primer y tercer escenario: Se recibe un pulso espurio antes de $t_{recepción\ r}$ (instante real de recepción de la trama) y no coincidente con $\Delta_{rw\ r}$ (margen de error para el comienzo de la recepción) (ver figuras 6.8, 6.9. y 6.10.). El escenario se describe en [Blanc *et al*. 2002 (b)] donde ya se advierte de la dificultad de reproducirlo con una herramienta de inyección física debido a la alto control temporal que se requiere. Para analizarlo se utilizó VFIT [Baraza *et al*. 2002] como herramienta de simulación.

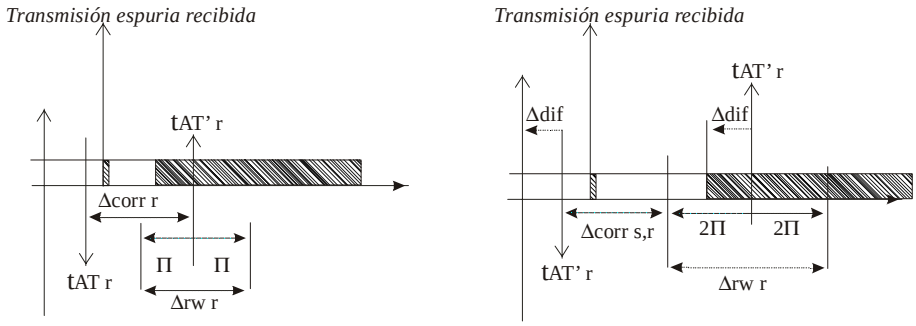


Figura 6.10. Transmisiones espurias fuera de $\Delta_{rw\ r}$

Al detectar el pulso espurio fuera de $\Delta_{rw\ r}$, el algoritmo de sincronización del receptor interpreta que la nueva trama esperada ha llegado fuera de tiempo y la rechaza. Al detectar la trama real, el algoritmo de sincronización vuelve a ejecutarse. Esta vez rechaza la trama real por considerarla una segunda recepción fuera de tiempo. En total se incrementa en dos el contador de tramas inválidas en una misma ventana de tiempo.

En un protocolo donde no se limite el incremento de los contadores por ventana de tiempo pueden ocurrir situaciones no deseadas. Por ejemplo, en la versión v0.1. se observó una avería SOS crítica en un sistema con cuatro nodos. Los nodos receptores del pulso espurio, al ejecutar el algoritmo de reconocimiento implícito antes de su propia transmisión y comparar el número de tramas inválidas recibidas con el número de tramas válidas, detectan un error de protocolo que congela su transmisión:

número de tramas inválidas = 2; *número de tramas válidas = 2*

En caso de que los tres nodos receptores dejen de transmitir, el nodo transmisor de la trama real, no pudiendo mantener su funcionalidad solo, también se desconecta

[89] La diferencia entre sincronización externa e interna se comentan en el apartado 4.3.2.

obedeciendo a un error de protocolo. En resumen, la transmisión espuria provoca la **desconexión de todos los nodos del sistema**[90], avería no contemplada en las especificaciones.

Con más de cuatro nodos, en la versión v0.1, la comparación del número de tramas inválidas con el de tramas válidas en los nodos receptores no genera un error de protocolo:

número de tramas inválidas = 2; *número de tramas válidas = 3*

El nodo transmisor es rechazado como miembro activo y pasará a un estado pasivo al ejecutar su algoritmo de pertenencia, tal como se contempla en las especificaciones.

En la versión 1.1. es más difícil observar una desconexión de todos los nodos del sistema en este escenario. En esta versión, el contador de tramas válidas se incrementa en uno si tras ejecutar el servicio de pertenencia el nodo permanece como miembro activo. Por tanto, en un sistema con cuatro nodos, los contadores quedarían:

número de tramas inválidas = 2; *número de tramas válidas = 3*

Para que se produjera una desconexión general, alguna trama más debería ser rechazada como inválida o sería necesario observar un silencio durante una ventana de tiempo, en cuyo caso el sistema estaría funcionando en modo degradado, con menos de cuatro nodos transmitiendo correctamente. Por tanto, este primer escenario sería propicio para una avería crítica en un sistema funcionando en modo degradado, provocando la desconexión de todos los nodos en un TDMA.

Por otro lado, el tercer escenario puede producir situaciones inesperadas como la representada en la figura 6.11. Esta figura representa un sistema con cinco nodos: *E* es el nodo transmisor, *D* es el nodo más lento y no percibe la perturbación, mientras que *C*, *B* y *A* sí la perciben. Ante la falta de consenso debería aplicarse una estrategia NGU por mayoría permaneciendo activos *C*, *B* y *A*. Sin embargo, en función del orden de transmisión, pueden ocurrir casos como el ejemplo de las figuras 6.11 y 6.12 donde es la minoría la que prevalece.

El ejemplo de la figura 6.11 representa un sistema con cinco nodos. Durante la transmisión del nodo *E* se produce una transmisión espuria que afecta a ambos canales. Sin embargo, la perturbación sólo es detectada en TDMA *i* por los nodos *C*, *B* y *A*. Los nodos *C* y *B* incrementan su contador de tramas no válidas[91] en 2. El nodo *A* aún espera el reconocimiento de su último envío.

La ventana de transmisión *j+1* corresponde al nodo *D*. Su contador de tramas válidas es estrictamente mayor que la suma de tramas no válidas (5 válidas y 0 no válidas), por lo tanto transmite. Además, con la trama transmitida por el nodo *D* en *j+1* se reconoce implícitamente la trama enviada por el nodo *E*, el cual incrementa su contador de tramas válidas en 1.

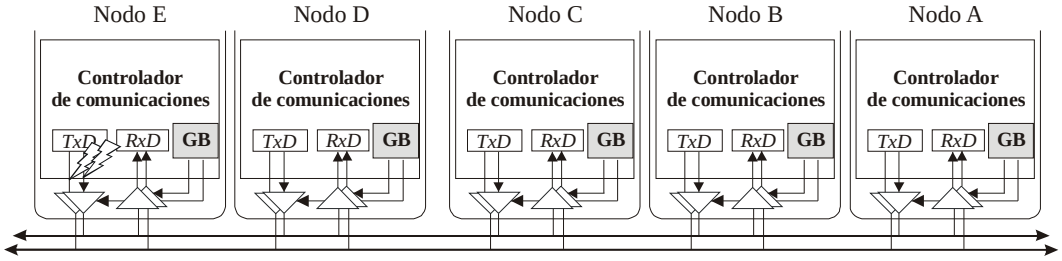
Las ventanas de transmisión *j+2*, *j+3* y *j+4* corresponden a los nodos *C*, *B* y *A* respectivamente. Tanto el nodo *C* como el nodo *B* incumplen el requisito impuesto para transmitir en el que el contador de tramas validas debe ser estrictamente mayor que la suma de tramas no válidas, pasando por tanto a estado pasivo sin llegar a transmitir. Al no

[90] Cluster shut-down

[91] En la versión v1.1. se diferencia entre tramas recibidas como inválidas y recibidas como incorrectas, siendo estas últimas aquellas con un CRC incorrecto.

transmitir, los silencios observados en el canal por el nodo *A* no le permiten obtener el reconocimiento a su último envío, pasando a estado pasivo sin llegar a transmitir de nuevo.

En TDMA *i+1* los nodos *C*, *B* y *A* estarán en estado pasivo y tratarán en volver a transmitir en ese mismo TDMA.



	Nodo E	Nodo D	Nodo C	Nodo B	Nodo A
TDMA *i - ventana j*					
Contador de tramas válidas	0	5	3	2	0
Contador de tramas no válidas	0	0	2	2	
TDMA *i - ventana j+1*					
Contador de tramas válidas	1	0	3	2	0
Contador de tramas no válidas	0	0	3	3	
TDMA *i - ventana j+2*					
Contador de tramas válidas	2	0	3	2	0
Contador de tramas no válidas	0	0	3 *Silencio*	3	
TDMA *i - ventana j+3*					
Contador de tramas válidas	2	0		2	0
Contador de tramas no válidas	0	0	*Pasivo*	3 *Silencio*	
TDMA *i -ventana j+4*					
Contador de tramas válidas	2	0			0
Contador de tramas no válidas	0	0	*Pasivo*	*Pasivo*	*Silencio*

Figura 6.11. Ejemplo del tercer escenario con cinco nodos

La figura 6.12. representa un sistema con siete nodos. En este ejemplo, sólo los nodos *D*, *C*, *B* y *A* detectan la perturbación. Los nodos *F* y *E* incrementan sus contadores de tramas válidas tras la recepción de la trama de *G*, mientras que *D*, *C* y *B* incrementan la suma de tramas no válidas.

En *j+1* y *j+2*, los nodos *F* y *E* transmiten sus respectivas tramas fijando la transmisión de *G* como correcta en sus vectores de pertenencia, lo que lleva al nodo *A* a pasar a estado pasivo al ejecutar el algoritmo de pertenencia.

En *j+3*, *j+4* y *j+5*, los nodos *D*, *C* y *B* incumplen el requisito impuesto para transmitir en el que el contador de tramas válidas debe ser estrictamente mayor que la suma de tramas no válidas, pasando por tanto a estado pasivo sin llegar a transmitir. A partir del TDMA *i+1*, los nodos *D*, *C*, *B* y *A* intentarán su reintegración.

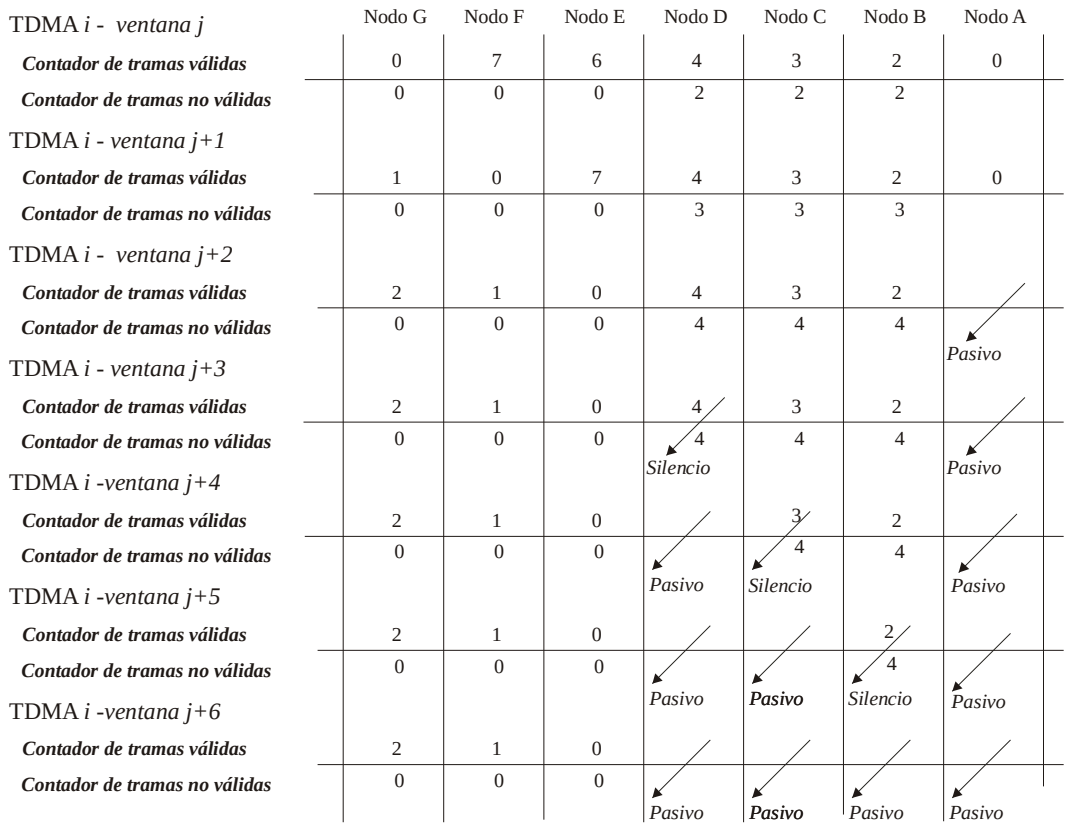


Figura 6.12. Ejemplo del tercer escenario con siete nodos

El problema de la avería señalada proviene de la doble ejecución del algoritmo de sincronización en una misma ventana de tiempo. El intervalo Δ_{cs} entre dos ejecuciones del algoritmo es el siguiente:

$$R_{int} \geq \Delta_{cs} > maxcorr \,.\, ((frMTs + 1)\,\Delta_{MT} + \Delta_{mt}) \qquad (6.2)\ [TTP]$$

R_{int} depende de las características de los componentes pasivos del nivel físico:

$$(\Pi - 2\varepsilon) / 4\rho \geq R_{int} \qquad (6.3)\ [TTP]$$

Siendo ε la diferencia entre el retardo de propagación mínimo y máximo del canal y ρ la precisión del oscilador.

Es el límite inferior el que nos interesa conocer para ratificar que se puede realizar una doble ejecución del algoritmo dentro de la misma ventana de tiempo. Siendo:

$$0 \leq maxcorr \leq (\Pi / 2\Delta_{mt}) \qquad (6.4)\ [TTP]$$

Δ_{mt} marca el periodo de la señal de reloj; mientras que Δ_{MT}[92] marca el periodo de transmisión formado por varios Δ_{mt}; finalmente, *frMTs* es un número asignable de MT, siendo por defecto 1[93]. Según la condición 6.4. el límite de Δ_{cs} marcado por *maxcorr* puede variar entre:

$$\Delta_{cs} > 0 \qquad (6.5)$$

[92] Macroticks
[93] No todas las implementaciones del controlador soportan un valor mayor que 1

$$\Delta_{cs} > (\Pi / 2\Delta_{mt}) ((frMTs + 1)\ \Delta_{MT} + \Delta_{mt}) \quad (6.6)$$

Cuando *maxcorr* es mínimo, el intervalo entre dos ejecuciones está dentro de los límites de una misma ventana de tiempo, pudiendo ocurrir la avería descrita. Cuando *maxcorr* es máximo:

$$\Delta_{cs} > \frac{\Pi}{2}\left[((\ frMTs\ +1)\cdot\frac{\Delta_{MT}}{\Delta_{mt}})+1 \right] \quad (6.7)$$

Para que la segunda ejecución del algoritmo fuese después de la transmisión de la trama, se debería cumplir que:

$$\frac{\Pi}{2}\left[((\ frMTs\ +1)\cdot\frac{\Delta_{MT}}{\Delta_{mt}})+1 \right] \geq \Delta_{corr\ r} + \mathbf{t_{tMAX}} \text{ para la versión 0.1} \quad (6.8)$$

$$\frac{\Pi}{2}\left[((\ frMTs\ +1)\cdot\frac{\Delta_{MT}}{\Delta_{mt}})+1 \right] \geq \Delta_{corr\ s,r} + (2\Pi - \Delta_{dif}) + \mathbf{t_{tMAX}} \text{ para la versión 1.0} \quad (6.9)$$

sii $\Delta_{corr\ s,r} > 0$. Cuando $\Delta_{corr\ s,r} = 0$ el fallo ocurre dentro de $\Delta_{rw\ r}$.

Sin embargo, estas condiciones no se cumplen necesariamente. Por ejemplo, en la implementación utilizada durante los experimentos, los valores empleados eran:

$$\Pi = 4{,}8\ \mu s;\ frMTs = 1;\ \Delta_{MT} = 1\ \mu s;\ \Delta_{mt} = 100\ ns$$

Con una trama de 1.940 bits de tamaño máximo permitido a 1Mbps[94], el intervalo entre dos ejecuciones del algoritmo de sincronización será inferior a t_{tMAX}.

Segundo y cuarto escenario: Los escenarios se ha observado para ambos prototipos con topología de bus, el TTP/C-C1 y C2, siendo de difícil reproducción con otras técnicas de inyección distintas a la inyección de fallos a nivel de pin, especialmente el cuarto escenario que implica la desviación de los relojes locales debido a causas físicas.

Cuando ocurre un evento que perturba el canal después de $t_{recepción\ r}$, la trama es rechazada en un 99,9% de los casos por un encapsulamiento incorrecto o en la comprobación del CRC[95]. Sin embargo, si se recibe un pulso espurio antes de $t_{recepción\ r}$, pero coincide con Δ_{rwr}, será tomada como parte de la trama y en conjunto (perturbación y trama) marcadas como una única recepción inválida.

Si todos los nodos reciben el pulso espurio y todos marcan como inválida la recepción, el consenso se mantiene en el sistema (segundo escenario). Sin embargo, el cuarto escenario es favorable para provocar la falta de consenso. La figura 6.13. muestra un posible ejemplo. Sólo los nodos *A* y *B* reciben el pulso espurio y marcan la trama como inválida, tras lo cual, el sistema se encuentra dividido en dos subgrupos, uno formado por *A* y *B*, y otro formado por *C*, *D*, *E y F*.

Se trata de una avería crítica que el TTP/C intenta salvar mediante una estrategia NGU por mayoría para evitar la falta de consenso. Sin embargo, no en todos los casos lo consigue. Por ejemplo en la versión 1.1, en un sistema con número par de nodos, si la

[94] Soporta hasta 5Mbps
[95] Experimentos descritos en el apartado 6.3.2.1.

mitad reciben el pulso espurio y la otra mitad (que incluye al transmisor) no, ningún subgrupo es mayoría y todos se desconectan del canal.

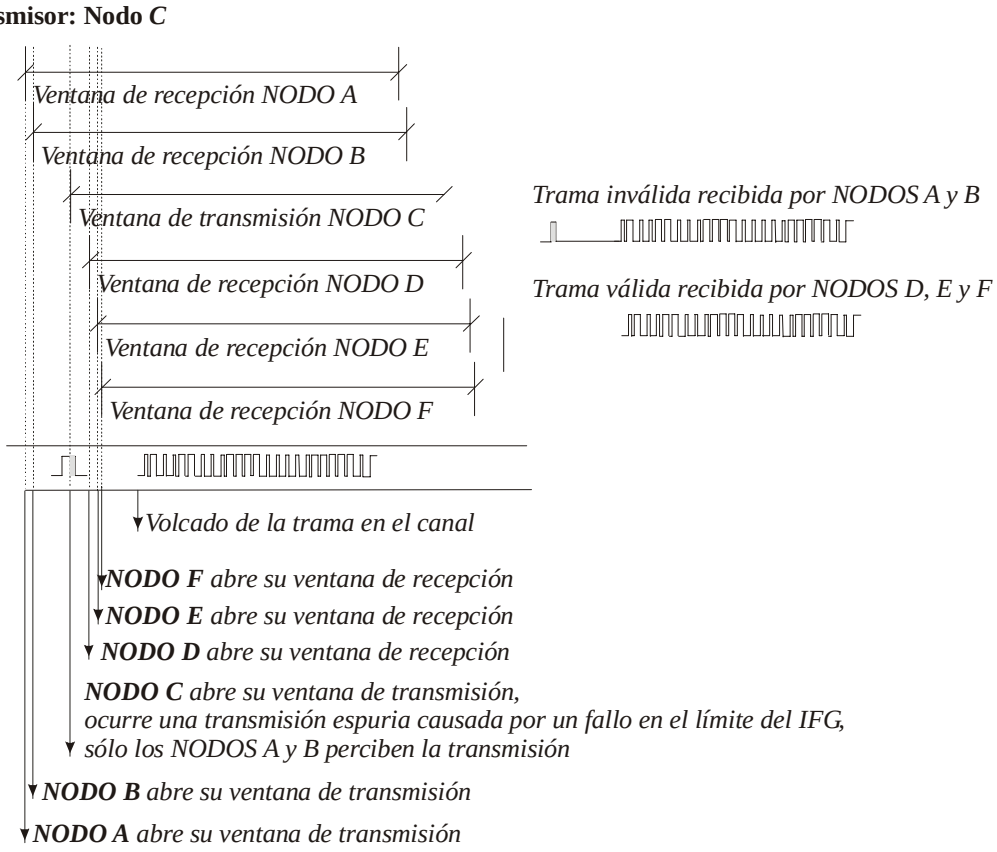


Figura 6.13. Escenario de una avería SOS en el dominio del tiempo

6.3.2.4. Averías SOS observadas con otras técnicas de inyección

Se observó un escenario de avería SOS con SWIFI [Ademaj 2000 (a)], que también se asume con radiación de partículas α [Sivencrona *et al*. 2003 (b)], en el prototipo TTP/C-C1. El escenario se detectó en el transcurso del proyecto de colaboración europeo FIT y se supone corregido en el nuevo prototipo TTP/C-C2 en la versión 1.1. En este caso se describe el efecto de un fallo sobre el MTCT[96] (registra el factor de corrección del reloj local con respecto a la base de tiempo global). El algoritmo de sincronización original, implementado en la versión 0.1, fijaba la relación entre Δ_{mt} y Δ_{MT}[97] igual para todos los nodos, sin tener en cuenta que la precisión de los osciladores puede ser distinta. Se demostró que un fallo sobre el MTCT (la inversión de un bit) tenía por consecuencia la aceleración en las transmisiones del nodo inyectado con una recepción próxima al límite de $t_{AT'\,r} - \Pi$, pero aún dentro de $\Delta_{rw\,r}$. A cada nueva transmisión del nodo inyectado, los relojes locales se corregían desplazándose incorrectamente hacia el periodo establecido por dicho nodo. A partir de cierto ciclo de funcionamiento, la distancia entre los relojes locales del nodo más rápido (nodo afectado por el fallo) y el nodo más lento era superior al límite establecido. Se consideró una avería SOS debido a que en vez de ser excluido del canal el nodo más rápido lo era el nodo más lento, cuyo funcionamiento había sido correcto y libre de errores hasta el momento su exclusión consensuada. En [Ademaj 2000

[96] Macrotick correction term

[97] Δ_{mt} marca el periodo de la señal de reloj; mientras que Δ_{MT} marca el periodo de transmisión formado por varios Δ_{mt}

(a)] se propone que la relación entre Δ_{mt} y Δ_{MT} se fije independientemente en cada nodo en función de la precisión de su oscilador. Teóricamente dice alcanzar una reducción de las averías de hasta un 90%. En sus experimentos la reducción alcanzada fue del 71,42%.

6.3.2.5. Observación de averías SOS en el dominio del valor

Las averías SOS en el dominio del valor son aquellas que se originan cuando las tensiones de entrada se encuentran fuera de los rangos establecidos para fijar un nivel lógico acorde con la tecnología utilizada. Un ejemplo de avería SOS se podría dar cuando la señal transmitida se encuentra próxima a una tensión umbral. Si la señal recibida se encuentra degradada no alcanzando dicha tensión umbral, cada nodo podría interpretar un valor lógico distinto. Si el valor coincide con el transmitido, el receptor no detecta ningún error, pero si no coincide, la comprobación del CRC debería marcar la trama como incorrecta. En el escenario de la avería observamos como algunos nodos han aceptado la transmisión mientras que otros han detectado un error, situación de inconsistencia no deseable.

Durante los experimentos no se observaron averías SOS en el dominio del valor debidas a transmisiones espurias en el canal de comunicaciones. Tampoco se observaron con otras herramientas como SWIFI o simulación, atribuyéndose un bajo porcentaje causadas por partículas α [Sivencrona *et al*. 2003 (b)].

Es posible poner a prueba al protocolo de comunicaciones forzando un escenario típico de una avería SOS provocando recepciones diferentes de una misma trama en distintos nodos. El experimento implica inyectar fallos múltiples de pegado en las líneas de recepción de dos o más nodos (figura 6.14.). Se hicieron varios experimentos tanto con inyección física como con el modelo en VHDL utilizando VFIT, pero al igual que con radiación de partículas α, el comportamiento del sistema siempre es el mismo: aplica una estrategia NGU por mayoría para alcanzar un estado consistente. Por tanto, observamos el mismo comportamiento del sistema con averías SOS en el dominio del valor que con las averías SOS en el dominio del tiempo planteadas en el cuarto escenario.

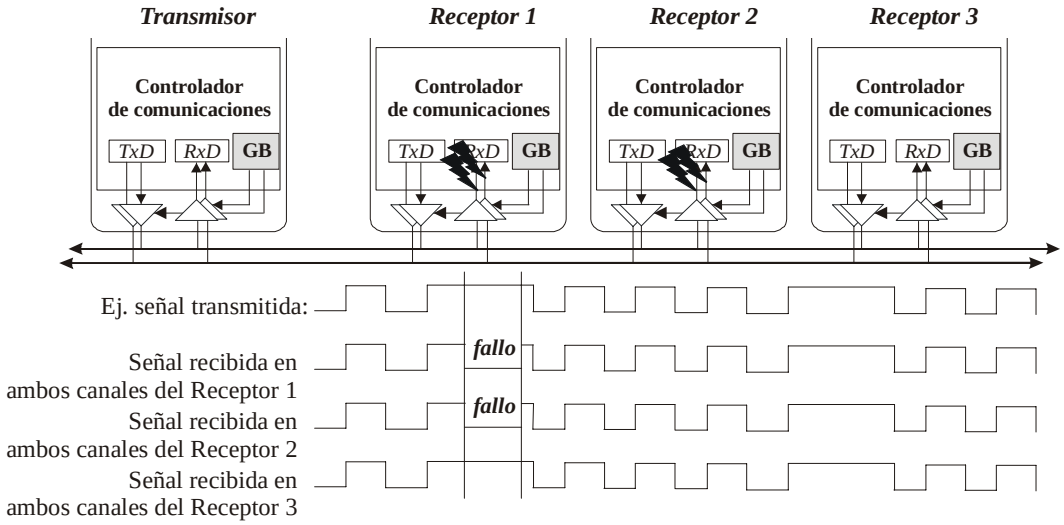


Figura 6.14. Escenario de una avería SOS en el dominio del valor provocado mediante la inyección de fallos múltiples de pegado

6.3.2.6. Reflexión sobre las estrategias NGU

El término *estrategia NGU* ("Never Give Up") implica la implementación de un recurso que evite, en cualquier caso, la caída completa del sistema. En el caso del TTP/C se opta por una estrategia NGU por mayoría. Si se pierde el consenso entre los nodos que comparten el canal de comunicaciones es muy probable que todo el sistema caiga, o lo que es lo mismo, que todos los nodos hayan pasado a estado pasivo al final del ciclo del sistema. Por tanto, la estrategia NGU está orientada a evitar la falta de consenso que, como hemos visto en los apartados anteriores, puede suceder ante una avería SOS.

Sin embargo, una estrategia NGU no asegura que el nodo origen del error asuma su condición de erróneo. Todo lo contrario, quizá sean nodos cuyo funcionamiento es totalmente correcto los que cesen su transmisión. Además, tampoco se asegura que habiendo dos nodos replicados sólo uno de ellos pase a estado pasivo permaneciendo su réplica activa.

Por tanto, en general, la ventaja de aplicar una estrategia NGU a no aplicar nada es la posibilidad de especificar el tiempo de restablecimiento del sistema. En un máximo de un TDMA, todos los nodos reconocen la nueva situación, incluso en los ejemplos de las figuras 6.11. y 6.12. Por ejemplo, ante una avería SOS transitoria, como las descritas en el tercer y cuarto escenario, que obligue una estrategia NGU, el tiempo máximo de restablecimiento del sistema podría estar limitado a dos TDMA, siempre que los nodos que permanezcan activos envíen tramas de tipo I de sincronización en el siguiente TDMA al concurrente con la avería que faciliten la reintegración de los nodos. Aún dependiente de la planificación del ciclo del sistema que el usuario realice, este tiempo de restablecimiento puede ser de varios ciclos inferior a una reintegración completa del sistema forzando un arranque en frío, aunque también podría ocurrir que el restablecimiento no fuese viable si en la planificación no se tiene en cuenta el envío de tramas de tipo I por parte de todos los nodos. Las especificaciones del TTP/C no imponen restricciones ni recomendaciones a la planificación que realiza el usuario en previsión de una posible averías SOS. De hecho, la hipótesis de fallos asume que el TTP/C tolera este tipo de averías si se utiliza un guardián de bus con funcionalidad adicional, refiriéndose al "signal reshaping" en una topología en estrella.

6.3.3. Pérdidas de Conexión

Uno de los problemas a resolver en una arquitectura de disparo por eventos es la dificultad de asegurar la consistencia del sistema ante las pérdidas de conexión. Las pérdidas de conexión se producen por defectos en el canal de comunicaciones generando divisiones aisladas del sistema en subgrupos. En la figura 6.15. se muestra una división por pérdida de conexión, donde los nodos *A*, *B* y *C* quedan aislados de los nodos *D* y *E*.

En un arquitectura de disparo por eventos, y especialmente aquellas con transmisiones "multicast", es difícil que todos los nodos detecten la avería al mismo tiempo. Por ejemplo, en un protocolo básico donde el reconocimiento se incluya en la trama de datos con un valor recesivo, si el nodo *A* envía una trama dirigida a *C* y *D*, *C* recibirá la trama correctamente y cambiará el valor del reconocimiento de recesivo a dominante. *A* detecta el reconocimiento de la trama, mientras que *D* ni siquiera la ha recibido.

Con una arquitectura de disparo por tiempo las pérdidas de conexión son fácilmente detectables. Cualquier transmisión es "broadcast". La trama se identifica por la ventana de tiempo en la que es enviada, e independientemente del tipo de datos transmitidos,

todos los nodos (a excepción del transmisor) comprueban que se recibe una trama y ésta es correcta. Por tanto, en el ejemplo anterior, *D* y *E* detectarán que el canal se mantiene en silencio durante las ventanas correspondientes a *A*, *B* y *C*. Del mismo modo, *A*, *B* y *C* detectarán que el canal se mantiene en silencio durante las ventanas correspondientes a *D* y *E* (figura 6.15.).

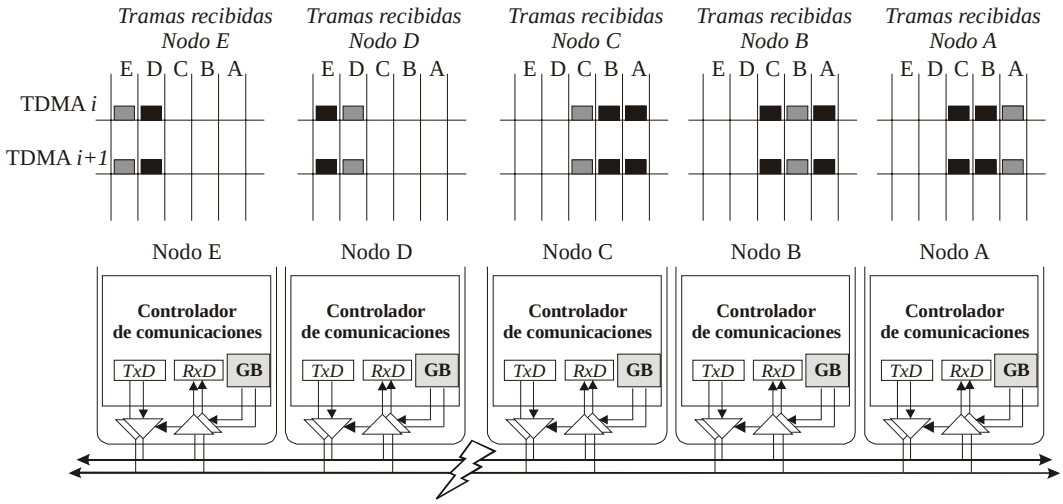


Figura 6.15. Ejemplo de una pérdida de conexión

En un TDMA, la situación será reconocida por todos los nodos, reaccionando de la siguiente forma:

Versión v0.1. Los silencios detectados ***sí*** se tienen en cuenta para el reconocimiento implícito:

- Si el grupo es de un solo nodo, se desconectará mediante un error de protocolo.
- Si el grupo incluye a más de la mitad del sistema pero menos de cuatro, mantendrá su funcionalidad en modo degradado.
- Si el grupo incluye un número inferior o igual de nodos que la mitad del sistema, se desconectarán al ejecutar el algoritmo de segregación en grupos[98]. Cuando la conexión vuelva a ser reactivada, si los nodos desconectados pueden alcanzar el estado de escucha, pasarán a estado pasivo desde donde deben reintegrarse en el sistema.

Versión v1.0. y v1.1 Los silencios detectados ***no*** se tienen en cuenta para el reconocimiento implícito:

- Si el grupo es de un solo nodo, se desconectará mediante un error de protocolo.
- Si el grupo incluye más de un nodo pero menos de cuatro, mantendrá su funcionalidad en modo degradado.
- Si el grupo incluye más de un nodo, cada nodo marcará como no activos a aquellos nodos cuya ventana de tiempo se mantenga en silencio. El grupo mantendrá su funcionalidad. Cuando la conexión vuelva a ser activada, dependiendo de la agrupación en la que hubiese quedado el sistema, es posible que algunos nodos sean excluidos temporalmente, pasando a un estado pasivo y reintegrándose en el siguiente TDMA.

Sin necesidad de cortar el canal de comunicaciones, es posible analizar el efecto de una avería por pérdida de conexión mediante inyección física de fallos a nivel de pin. Los experimentos implican la eliminación del pulso o pulsos correspondientes a la señal de

[98] Clique avoidance

control que habilita la transmisión de un nodo. En una muestra de 1.500 experimentos, se probaron dos configuraciones: topología de bus con canal replicado y con canal no replicado.

En una topología de bus con canal replicado, eliminar un pulso de habilitación impide la transmisión de la trama por uno de los canales del bus. La implementación de los algoritmos correspondientes al servicio de pertenencia y reconocimiento implícito tienen en cuenta esta posible circunstancia, considerando válida la trama si ésta se recibe correctamente por al menos un canal. Un silencio de bus o una trama inválida en un canal no se consideran siempre que se obtenga una trama válida por el otro canal. En este sentido, los experimentos realizados tienen como objetivo verificar la implementación de los algoritmos que resultó ser positiva. Pero por otro lado, también hay que tener en cuenta cual es el valor que se ha perdido junto con la trama. Para tramas no replicadas, si la trama perdida es de tipo I (de sincronización) podría tener un efecto negativo en acciones de reintegración. Si la trama perdida es de tipo N (de datos), el impacto se observará sólo a nivel de aplicación.

En una topología de bus con canal no replicado, los algoritmos correspondientes al servicio de pertenencia y reconocimiento implícito requieren la correcta recepción de la única trama enviada. En la muestra de experimentos, los fallos inyectados tienen una duración equivalente a la ventana mínima de la planificación utilizada. Sin embargo, el instante de inyección no se encuentra sincronizado con el inicio de la ventana de transmisión, ya que no existe ninguna señal de salida disponible que indique dicho inicio, obteniéndose una activación del 20,6% de los errores (con 4 ventanas planificadas y un disparo uniformemente distribuido). Para todos los errores activados, el nodo pierde su pertenencia en menos de un TDMA tal como se indica en las especificaciones.

6.4. Resumen y Conclusiones del Capítulo

Este capítulo se centra en la inyección de fallos a nivel de pin como técnica que facilita el análisis del comportamiento del protocolo de comunicaciones TTP ante diferentes tipos de averías mediante la introducción deliberada de un conjunto de fallos cuya ocurrencia afecta directamente a la transmisión o recepción de mensajes.

En sistemas distribuidos tolerantes a fallos, el protocolo de comunicaciones debería asegurar un comportamiento confiable del sistema, no sólo ante errores internos ocurridos en un nodo, sino también ante fallos en el canal de comunicaciones o ante acciones de desconexión y reintegración sin que conlleven una pérdida del servicio y sin que se incumplan los requisitos establecidos para albergar el tipo de aplicaciones a las que se orienta el sistema. En este sentido, los experimentos realizados han permitido, por un lado, verificar si la implementación del prototipo cumple con las especificaciones de diseño, y por otro lado, validar la confiabilidad del protocolo ante los tipos de avería más frecuentes que se pueden observar en un sistema distribuido, como son la parada de un nodo, las transmisiones espurias sintáctica o semánticamente incorrectas, las denominadas averías SOS y la pérdida de conexión.

Una parada en las transmisiones de un nodo activo se puede considerar como un evento crítico en un sistema distribuido, ya que, aunque un nodo se desactive temporalmente por alguna causa interna, se espera que el resto del sistema tolere esta circunstancia y mantenga un servicio correcto. La arquitectura TTA asegura el estado consistente del sistema ante eventos como la pérdida de mensajes, donde cualquier error debe ser detectado por todos los nodos activos debiendo consensuar la solución. El reconocimiento de la transmisión es más robusto que en otros protocolos, debido a que es

el sistema quien reconoce la transmisión y no sólo el nodo destino de los datos. De hecho, la posibilidad de que un nodo pare temporalmente sus transmisiones se tiene en cuenta en las especificaciones de diseño y además, se asume como el comportamiento esperado que debe tener el nodo ante la detección de un error (sistemas "*fail-silence*" o de comportamiento silencioso). Los experimentos realizados en esta parte de la validación del protocolo se basan en la variación controlada de los márgenes temporales de las señales de control que tienen como consecuencia la interrupción del volcado de la trama o la habilitación de la transmisión fuera del periodo de tiempo establecido, caso que favorece la aparición de fallos espurios coincidentes con la transmisión de otro nodo. La perturbación, tanto en un canal como en ambos simultáneamente con fallos conexos forzando pegados simples y dobles, ha permitido verificar la implementación del reconocimiento implícito de las tramas definido en el protocolo y analizar sus carencias cubiertas por el servicio de pertenencia.

La reintegración de un nodo como parte activa del sistema tras una parada debe asegurar que su restablecimiento no implique la desactivación de otro nodo. La inyección física de fallos a nivel de pin ha permitido emular paradas temporales de los nodos y su posterior reintegración, observando una premisa no especificada en el protocolo: existiendo un número mínimo de nodos que permiten mantener la base de tiempos del sistema sincronizada, sería posible asegurar la reintegración simultánea de un número determinado de nodos siempre que cada nodo del sistema tenga planificado el volcado de una trama de sincronización por ciclo en alguno de los dos canales de comunicaciones. Bajo esta premisa, los experimentos no han detectado que la reintegración de uno o más nodos supusiera la exclusión de algún otro nodo activo. Sin embargo, cabe destacar que el tiempo de establecimiento necesario es muy alto, normalmente superior al ciclo completo de transmisiones, siendo recomendable la replicación de aquellos nodos que soporten algoritmos de control críticos. En sistemas donde puedan existir diferencias en la precisión de las señales de reloj de los controladores, el TTP propone que sean sólo aquellos nodos con características físicas similares (denominados *relojes maestro*) los que realicen el ajuste de la base de tiempos. Sin embargo, en este capítulo se considera que ésta no es una solución que aporte ninguna mejora en la reintegración de nodos, por el contrario, podrían ocurrir situaciones en las que la reintegración fuera inviable. Por lo tanto, parece más interesante la replicación de los nodos críticos independientemente de la precisión de sus osciladores.

La validación del TTP ante transmisiones espurias se ha llevado a cabo mediante la inyección de fallos coincidentes con la transmisión de alguna trama, causando la alteración de la misma, y con fallos no coincidentes, atribuyéndose a fallos del propio canal o de los componentes pasivos del nivel físico. La división de los experimentos según el instante de la inyección ha facilitado el análisis de los resultados. Primero, respecto a los fallos coincidentes con una transmisión, se observa experimentalmente una cobertura de detección por el cálculo del CRC de la trama del 99,9%. Segundo, los fallos contenidos dentro del *intervalo entre tramas IFG* no tienen efecto directo sobre la comunicación. Sin embargo, el IFG delimita la apertura de la ventana de recepción. La desviación en la apertura de las ventanas de los nodos puede favorecer la aparición de averías SOS en el dominio del tiempo ante la ocurrencia de pulsos espurios en el límite del IFG. Por tanto, la sincronización de la base de tiempos común es crítica en el TTP, y extensible a cualquier protocolo de disparo por tiempo, ya que a mayor desviación de los relojes locales mayor probabilidad de que una transmisión espuria afecte al correcto funcionamiento del sistema al convertirse en una avería SOS, tipo de averías asimétricas con grave incidencia en la arquitectura TTA. Finalmente, con respecto a los fallos coincidentes con una ventana de transmisión diferente a la asignada, se ha observado la fuerte dependencia que existe entre el guardián del bus y los componentes del nivel físico utilizados en el diseño. Componentes de los que depende para contener eficazmente

transmisiones espurias y evitar así la propagación de errores.

En este capítulo se propone la división de los experimentos en escenarios según el instante de inyección del fallo, diferenciando, por un lado, el intervalo entre la apertura de la ventana de recepción y el instante a partir del cual se espera la recepción de una trama, y por otro lado, el intervalo entre dicho instante y el cierre de la ventana. El análisis de los experimentos realizados sobre el primer intervalo se ha coordinado entre inyección física a nivel de pin y simulación del modelo con VHDL. Ambas técnicas han resultado complementarias, ya que la percepción del problema en el prototipo ha facilitado su reproducción en el modelo mejorando la precisión en el instante de inyección. Los experimentos realizados sobre el segundo intervalo se han llevado a cabo sólo con inyección física, revelando su potencial para escenarios específicos como aquellos que implican diferencias en la precisión de los relojes locales.

El estudio de los resultados obtenidos a partir de fallos inyectados entre la apertura de la ventana de recepción y el instante establecido para la recepción de la trama revela como punto débil del protocolo que no existe limitación en la actualización de sus variables por ventana de tiempo, en concreto, el incremento de los contadores relativos al número de tramas válidas e inválidas no está limitado por ventana, donde se asume que sólo se puede recibir una trama. Así, observamos, por ejemplo, que en la versión v0.1. del protocolo, cuatro nodos no son suficientes para soportar una avería SOS produciéndose la desconexión completa del sistema. Las nuevas versiones v1.0. y v1.1. del protocolo han sido motivadas por errores o carencias detectadas durante la validación experimental. La versión v1.1. mejora el algoritmo haciendo más robusto el protocolo. Sin embargo no suficiente. Existen escenarios con número impar de nodos donde, a pesar de existir una mayoría que detecta la ocurrencia de un pulso espurio anterior a la recepción de la trama, son los nodos en minoría los que se mantienen como activos, produciéndose la desconexión en cadena del resto.

Otro escenario crítico es el que plantea una transmisión espuria detectada antes de la recepción real de la trama e interpretada como parte de la misma. Este escenario, especialmente si se tiene en cuenta la desviación de los relojes locales que retarda o adelanta la apertura de unas ventanas de recepción con respecto a otras, fuerza una situación de inconsistencia no deseable. Al igual que con averías SOS en el dominio del valor, sólo algunos nodos marcan la trama recibida como inválida mientras que el resto la toman como válida. En general, una avería SOS en un sistema distribuido provoca la pérdida de consenso entre los miembros activos del sistema. Esta pérdida de consenso es la que trata de evitar el TTP utilizando estrategias como la NGU donde el subgrupo mayoritario impone su visión del sistema con la intención de mantener la continuidad del servicio.

Además de comprobar experimentalmente que no siempre se cumple este objetivo, ya bien sea por carencias del protocolo o por situaciones de empate, en una reflexión sobre esta estrategia se apuntan dos hechos. Primero, que no se asegura que el nodo origen del error asuma su condición de erróneo. Segundo, que tampoco se asegura que habiendo dos nodos replicados sólo uno de ellos pase a estado pasivo permaneciendo su réplica activa. Consideramos que la diferencia entre aplicar una estrategia como la NGU o no aplicar nada es la posibilidad de especificar el tiempo de restablecimiento del sistema completo. Aún dependiente de la planificación que el usuario realice, este tiempo de restablecimiento puede ser muy inferior a la reintegración forzada desde un arranque en frío, situación probable tras la pérdida de consenso.

Finalmente, en el capítulo se analiza experimentalmente el comportamiento del protocolo ante la pérdida de conexión de uno o más nodos. Para que exista consenso entre

los miembros activos del sistema, el TTP, a diferencia de otros protocolos de disparo por evento, facilita que todos los nodos detecten una posible avería en las comunicaciones con una latencia máxima conocida. La inyección física a nivel de pin posibilita la validación del protocolo ante este tipo de averías emulando cortes transitorios de distinta duración sin causar daño alguno al sistema. Los experimentos realizados diferencian entre una topología de bus con canal sin replicar y con canal replicado verificando la correcta implementación del servicio de pertenencia y del reconocimiento implícito propuesto en TTP.

Transversalmente a los experimentos realizados con inyección física a nivel de pin, en este capítulo se procura mantener una comparación con los resultados obtenidos con otras técnicas de inyección. En general, las técnicas de inyección física de fallos son más apropiadas para inyectar fallos en el canal de comunicaciones sobre el prototipo que las técnicas basadas en SWIFI. Aunque podemos comparar algunos comportamientos similares, éstos suelen deberse a errores diferentes. Además, la inyección de fallos a nivel de pin es una técnica que dota al entorno de validación de un control no disponible con otras técnicas de inyección física. Por ejemplo, en los experimentos llevados a cabo con radiación de partículas α se observan comportamientos similares a los observados con SWIFI o con AFIT. Sin embargo, la imprecisión a la hora de analizar el origen de cada comportamiento observado hace que no sea posible afirmar que un experimento determinado haya sido reproducido con ambas técnicas. Los mejores logros en la reproducción de experimentos se han alcanzado entre inyección física a nivel de pin e inyección de fallos en el modelo VHDL. Ambas técnicas pueden llegar a ser complementarias en el análisis de escenarios críticos.

Capítulo 7. Mejora de la Cobertura de Detección

La confiabilidad de un sistema distribuido basado en la arquitectura TTA no sólo depende de su fiabilidad ante averías relacionadas con el protocolo de comunicaciones, sino también debe garantizar la seguridad en el dominio del valor. Este dominio hace referencia al contenido de los mensajes recibidos por algún miembro del sistema. En la arquitectura TTA ningún nodo conectado al canal de comunicaciones debe transmitir un mensaje a no ser que pueda asegurar una transmisión libre de errores, lo que incluye ambos dominios, el tiempo y el valor. Un mensaje cuyo contenido sea incorrecto no puede ser detectado por el receptor, a menos que se le proporcione de algún recurso como mecanismo de tolerancia a fallos. Estos mecanismos no se encuentran implementados a todos los niveles en el controlador TTP^{TM}/C, lo que lo hace vulnerable ante fallos físicos, como es el caso de fallos localizados en la interfaz de comunicaciones. Esta interfaz se considera en la arquitectura como una barrera de contención que debe impedir la propagación de errores entre el nivel de aplicación y el protocolo de comunicaciones.

En este capítulo se analizan los tipos de errores derivados de fallos físicos localizados en la interfaz de comunicaciones y se proponen diferentes alternativas para mejorar la cobertura de detección de estos errores, alternativas aplicables a cualquier arquitectura distribuida. Las observaciones y resultados contenidos en el capítulo son tratados también en [Blanc y Gil 2003 y Blanc *et al*. 2004].

7.1. Errores de Diseño e Implementación

En el capítulo 2 se describen las diferencias principales entre los dos posibles objetivos de una campaña de inyección: la eliminación de fallos y la predicción de fallos. Como parte de la eliminación de fallos se incluye la detección de errores de diseño e implementación. En muchas ocasiones estos errores se detectan porque el sistema no cumple con las especificaciones de diseño ante la ocurrencia de cierto evento. En este sentido, durante los experimentos realizados con inyección física de fallos a nivel de pin, se detectó que la implementación del algoritmo de vida descrito en las especificaciones del protocolo TTP era incorrecta.

Dentro de los servicios de interconexión con el nivel de aplicación se describe el algoritmo de vida[99]. Utiliza dos registros: el testigo de vida del controlador y el testigo de vida de la aplicación. Siempre que el nivel de aplicación se encuentre activo, éste debe actualizar su testigo de vida. En caso de que el controlador de comunicaciones TTP/C detecte que el testigo no ha sido actualizado correctamente, detiene las transmisiones y pasa a estado pasivo esperando el restablecimiento del nivel de aplicación. Además, según las especificaciones de diseño, el controlador debe generar un pulso asociado a una interrupción externa del microcontrolador que soporta la aplicación y actualizar el

[99] Apartado 6.2.3.6.

registro de estado de las interrupciones localizado en la CNI.

Durante los experimentos se detectó una situación no esperada. Ante fallos inyectados durante la actualización del testigo de vida de la aplicación, el controlador pasa a estado pasivo y deja de transmitir en el siguiente TDMA, sin embargo, no genera el pulso asociado a la interrupción externa ni actualiza *el registro de estado de las interrupciones*. Esta discrepancia entre el comportamiento observado y las especificaciones de diseño sólo se puede considerar como un error en la implementación del controlador.

A fin de validar ciertos resultados, sería interesante reproducir los mismos experimentos con dos técnicas de inyección diferentes. Este es el caso de la inyección física a nivel de pin y la inyección de fallos basada en simulación de modelos con VHDL. Utilizando una herramienta como VFIT [Gracia *et al*. 2002 (b)], podemos trabajar a nivel de sistema en VHDL, disponiendo de los pines de entrada y salida del controlador.

El algoritmo de vida establece dos accesos al **área de memoria de registros de control y estado** de la CNI. El primer acceso es una lectura sobre el testigo de vida del controlador. El segundo acceso es una escritura que actualiza el testigo de vida de la aplicación con el valor inverso al leido en el testigo de vida del controlador. La figura 7.1. muestra los seis tipos de fallos inyectados, su duración y el acceso con el que se sincroniza el disparo.

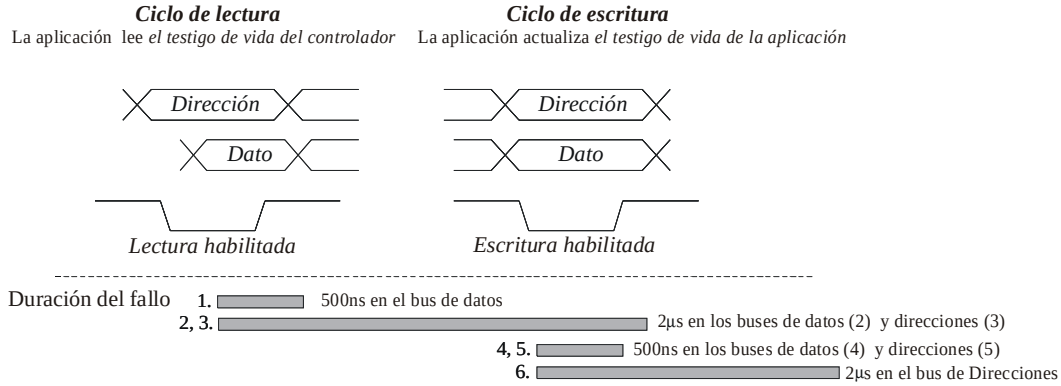


Figura 7.1. Algoritmo de vida: descripción de los fallos inyectados

En 1 y 4, el testigo se actualiza con un valor erróneo. En 3 y 5, el testigo no se actualiza. En 6 observamos una excepción del sistema operativo, mientras que en 2 el error no se activa si el fallo es efectivo durante la lectura, ya que el valor a escribir en la CNI es precisamente el que se fuerza con el pegado. Por ejemplo, si se fuerza un pegado a 0, el valor leído por la aplicación será 0 cuando debería ser 1; la aplicación invierte el valor de 0 a 1 y trata de escribirlo, sin embargo, si el pegado persiste, lo que realmente escribirá es un 0, valor invertido del 1 original. Consecuencia del fallo, en 1, 3, 4 y 5 el controlador suspende la transmisión durante un TDMA. Por el servicio de pertenencia[100], es excluido como miembro activo pasando a estado pasivo. Sin embargo, el controlador no actualiza los registros especificados de la CNI. Este defecto se marca como ***silencios sin actualización*** en los resultados que se exponen en las siguientes secciones.

Este error de implementación se detectó en ambos controladores del TTP/C-C1 y TTP/C-C2, tanto en el prototipo, mediante inyecciones físicas a nivel de pin, como en el modelo en VHDL. Además de detectar un error de implementación, estos experimentos

[100] Apartado 6.2.3.3.

resultan de gran importancia ya que nos permiten validar herramientas de inyección que mediante técnicas diferentes obtienen resultados equivalentes. Por otra parte, también remarcan la necesidad de validar el modelo, evitando que aparezcan errores de diseño o implementación al construir el prototipo.

7.2. Errores Simples y Múltiples Derivados de Fallos Físicos

La utilización de algoritmos de control sintéticos, con la misma carga de trabajo que un algoritmo de control específico pero con un intercambio de mensajes con valores predefinidos, resulta de gran utilidad para observar la forma de los errores en memoria causados por fallos físicos.

En los algoritmos de control distribuidos relacionados con sistemas "*x-by-wire*", el número de bytes de datos que un nodo envía por mensaje se aproxima más a diez bytes que a cien bytes [Lönn y Snedsböl 1998]. Según esta aproximación, en los experimentos realizados con inyección física a nivel de pin, se opta por diseñar una carga sintética con una planificación similar a la que tendría el algoritmo de control del ABS[101], utilizando mensajes con 6×16 bits de datos.

Las ***tareas software*** del monitor[102] implementadas en los nodos TTP son las siguientes:

- La primera tarea, implementada en un nodo libre de fallos, consiste en la lectura del vector de pertenencia cada TDMA para comprobar y notificar si algún nodo ha perdido su condición de miembro durante la última transmisión.

- Existen dos nodos replicados que ejecutan la misma aplicación. Uno es el nodo inyectado, mientras que el otro nodo ejecuta la aplicación libre de fallos[103]. La segunda tarea, implementada en un tercer nodo, compara los datos recibidos de ambas réplicas. En caso de que los datos difieran, se lleva a cabo una acción adicional. La tarea debe determinar si los datos recibidos desde el nodo inyectado son datos sin actualizar, pertenecientes a un mensaje anterior, o si son datos nuevos que contienen errores. El primer caso indica que la memoria de mensajes de la interfaz de comunicaciones CNI no ha sido actualizada en el último TDMA con los nuevos valores calculados por el algoritmo de control. Este evento debería detectarse evitando la retransmisión del mensaje. El segundo caso indica que un fallo ha modificado el contenido de los datos enviados.

- La tercera tarea, implementada en un nodo libre de fallos, está destinada a comprobar, cuando existan, si los mecanismos de tolerancia a fallos pueden detectar errores en los datos enviados.

7.2.1. Errores no detectados

En una arquitectura como TTA, la distribución de los algoritmos de control hace que desde que se calcula el valor de una variable del algoritmo de control en un nodo hasta

[101] Algoritmo "brake-by-wire" proporcionado por VOLVO Technological Development [FIT 2002]
[102] Apartado 5.2.2.
[103] Golden node

que este nuevo valor es utilizado en el nivel de aplicación de otro nodo, transcurra un tiempo durante el cual pueden aparecer en el sistema uno o varios fallos físicos que provoquen errores en los bits de datos. En la figura 7.2. se muestra una relación secuencial de las etapas por las que pasan los datos. Los extremos de la figura representan los cálculos realizados a nivel de aplicación en dos nodos diferentes. Las estrategias funcionales[104] pueden ser muy apropiadas para detectar errores asociados a fallos físicos que afectan a la ejecución del algoritmo o que corrompen el resultado. Al aplicar una estrategia como la doble ejecución, el número de errores no detectados se puede reducir a la mitad [Ademaj 2003]. La mayoría de los errores que siguen sin ser detectados son aquellos que se originan en memoria, causados por fallos que corrompen el valor de los datos antes de que se lleve a cabo la primera ejecución. Por este motivo, en la figura 7.2. se considera que durante el almacenamiento de los datos en la memoria de mensajes pueden aparecer fallos físicos que afecten a celdas de memoria, como por ejemplo, entre otros, SEU debidos a la exposición del semiconductor a niveles de radiación. Además, debemos considerar una etapa intermedia entre el cálculo y el almacenamiento en memoria: la escritura o lectura entre controladores[105]. La escritura (o lectura) de datos en memoria también puede verse afectada por fallos físicos, como por ejemplo los debidos a interferencias electromagnéticas, radiación de partículas, deterioro de las soldaduras, condiciones ambientales, etc. Finalmente, en la figura 7.2. se ha considerado el volcado de los datos en el bus o canal de comunicaciones. Durante la transmisión, la mayoría de los protocolos de comunicaciones protegen la trama con algún código de detección de errores, como por ejemplo, en el caso del TTP, con un código cíclico redúndate (CRC). Sin embargo, en la arquitectura TTA, los datos no se encuentran protegidos por ningún mecanismo de detección de errores ni durante la escritura o lectura entre controladores ni durante el tiempo que los datos permanecen almacenados en memoria. Además, debemos considerar que un fallo físico puede causar varios errores [DBench ETIE2 2002] y que es posible que ocurran varios fallos físicos, no necesariamente conexos sino distribuidos a lo largo del periodo contenido entre los extremos de la figura 7.2.

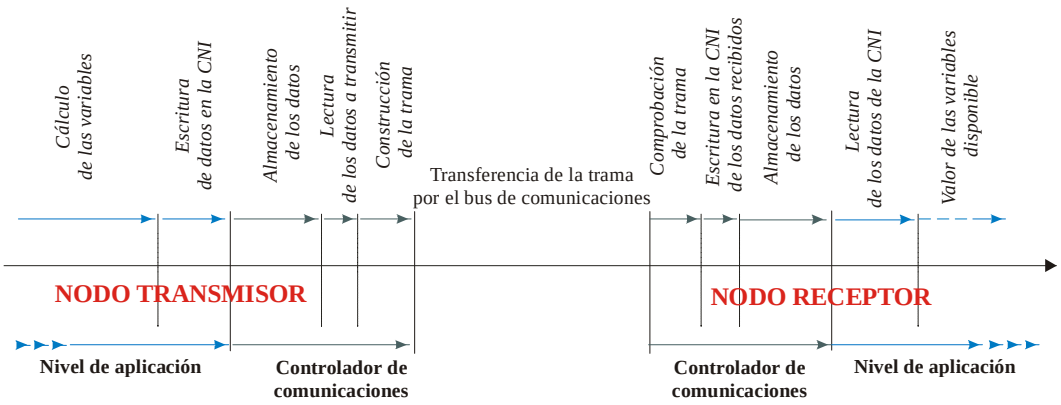


Figura 7.2. Relación secuencial relativa a la vulnerabilidad de los datos

7.2.1.1. Errores unidireccionales sobre datos derivados de fallos simples

Se considera como error unidireccional a la inversión del valor lógico de varios bits de datos causado por un mismo evento, siendo la dirección de cambio (a 0 o a 1) la misma para todos los bits [Pradhan 1980, Pradhan *ed*. 1996]. Estos errores se consideran,

[104] Apartado 2.2.1.4.
[105] Entre el controlador que soporta la aplicación y el controlador de comunicaciones, ambos conectados a través de la CNI.

típicamente, causados por una línea que se encuentre pegada a un valor[106]. Para transmisiones en serie, todos los bits afectados serán adyacentes en la secuencia. Sin embargo, para transmisiones en paralelo, los bits afectados son los transmitidos por la misma línea de bus, perteneciendo a palabras distintas pero adyacentes de la secuencia de transmisión.

La figura 7.3. muestra un ejemplo de error unidireccional debido a un pegado a 0 en la línea 6 del bus de datos. El **bloque de datos** se compone de los bits de datos que la aplicación lee o escribe en la memoria de mensajes de la CNI antes del envío de un mensaje o posterior a la recepción. Siendo el bus de datos de 16 bits (tamaño asignado en el TTP/C), el bloque de datos representa en horizontal las palabras escritas (o leídas) en cada acceso a memoria. Por ejemplo, en la figura se representan 6 accesos para escritura o lectura de seis palabras de 16 bits (bloque de datos de 96 bits). Un **combinación vertical** se compone de todos los bits de un *bloque de datos* transmitidos por la misma línea del bus de datos. Siendo los accesos a memoria consecutivos, la duración del fallo influye en el número de bits afectados en una misma combinación vertical. Así, en el ejemplo de la figura, el fallo afecta a 4 palabras; en todas, el bit 6 fuerza su valor a 0.

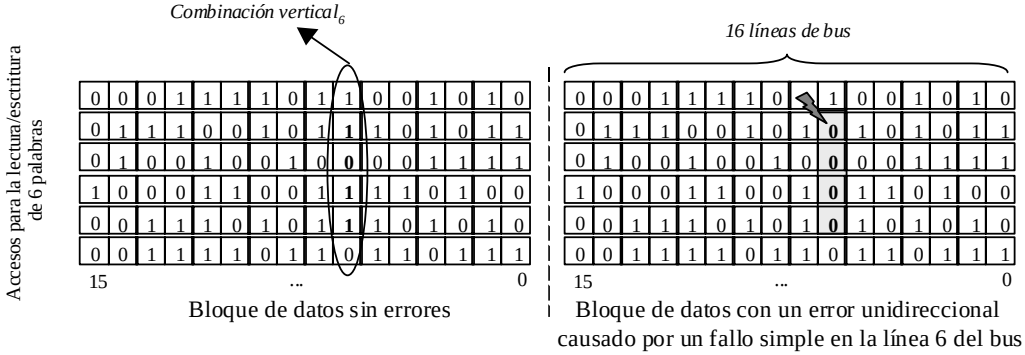


Figura 7.3. Ejemplo de error unidireccional en un bloque de datos

En la figura 7.4. se muestran algunos ejemplos de posibles patrones de error. Los puntos en negro muestran los bits afectados por el fallo pero cuyo valor lógico permanece inalterado, mientras los puntos en rojo muestra los bits que invierten su valor. No todos los bits afectados por un fallo invierten su valor; el motivo se debe a que el valor original del bit afectado puede coincidir con el valor forzado por el fallo, por tanto, el fallo no es efectivo en dicho bit.

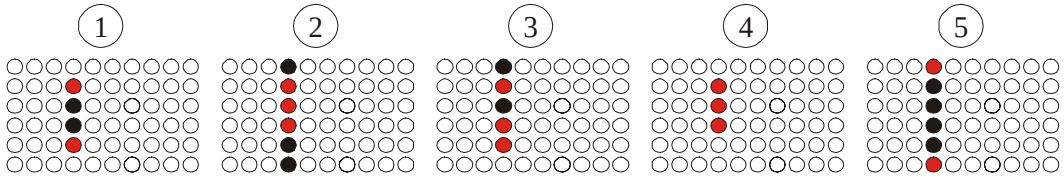


Figura 7.4. Ejemplos de patrones de error que representan errores unidireccionales

7.2.1.2. Errores aleatorios múltiples derivados de fallos simples

Se considera como error aleatorio múltiple a la inversión del valor lógico de varios bits de datos causado por un mismo evento, siendo la dirección de cambio aleatoria para cada bit afectado [Pradhan 1980, Pradhan *ed*. 1996]. Las causas que pueden provocar un error aleatorio múltiple en datos son diversas [DBench ETIE2 2002], aunque se pueden clasificar según su efecto. En TTA, atendiendo a las observaciones extraídas de los

[106] Stuck-at

experimentos sobre memoria de mensajes, se podrían clasificar en inversiones en memoria o registros, escrituras no efectivas y accesos a direcciones erróneas.

Inversiones en memoria o registros

Como se vio en el capítulo 3, una posible causa de error múltiple en memorias SRAM es la radiación de partículas, aunque los errores múltiples también pueden aparecer debidos al deterioro, el desgaste o a condiciones ambientales como las interferencias electromagnéticas o el aumento de la temperatura. En el caso de errores producidos por radiación, en los experimentos realizados hasta el momento el máximo número de bits afectados es aproximadamente de cuatro, aunque el incremento de la densidad de integración y la reducción de la tensión de alimentación también pueden incrementar el número de posibles celdas afectadas por una incidencia.

Los patrones de error en memoria son muy variados. Los errores pueden producirse en celdas de memoria no necesariamente alineadas en un mismo eje, sino que se pueden distribuir en horizontal, vertical, diagonal o combinaciones. Además, la distribución de los datos almacenados en memoria no tiene necesariamente que coincidir con el orden establecido por el bloque de datos, resultando en patrones de error muy diversos.

Escrituras no efectivas

Se observó que ante la ocurrencia de ciertos fallos transitorios, la escritura de datos en memoria no llega a ser efectiva en la dirección establecida. El valor contenido en la dirección no se actualiza con los nuevos valores, siendo éste un suceso no detectado. Las observaciones corresponden a fallos físicos transitorios a nivel de pin coincidentes con la escritura de la aplicación sobre la memoria CNI del controlador de comunicaciones. Además, también se observó el mismo efecto con fallos SWIFI cuando la escritura se realiza tras la recepción de un nuevo mensaje. Si el fallo altera el contenido de algunos registros del controlador de comunicaciones se impide que la PCU ejecute parte del código que escribe el contenido del mensaje en la CNI [Ademaj 2003]. En ambos casos, el fallo debe coincidir con la escritura de datos en memoria.

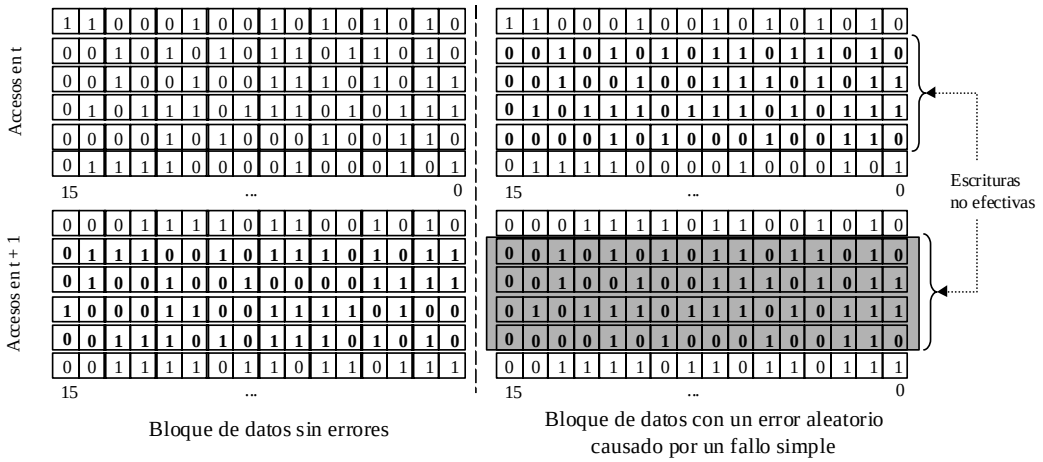


Figura 7.5. Ejemplo de error aleatorio múltiple causado por escrituras no efectivas

La figura 7.5. muestra un ejemplo de cómo quedaría el bloque de datos en caso de una escritura libre de fallos (ver izquierda de la figura 7.5.) y cómo queda el bloque de datos en caso de fallo (ver derecha de la figura 7.5.). En este caso, la duración del fallo también influye en el número de palabras no actualizadas (cuatro en la figura 7.5.).

Un bloque de datos formado erróneamente tanto por palabras correctas (actualizadas) como incorrectas (no actualizadas) se puede considerar como un bloque de datos que contiene un error aleatorio múltiple. En la figura 7.5. se muestran algunos ejemplos de posibles patrones de error. No todos los bits afectados invierten su valor ya que puede coincidir con el valor sin actualizar. El patrón de error depende de cómo se vayan actualizando las variables calculadas por el algoritmo de control. El ejemplo 1 de la figura 7.6. podría representar variables de 16 bits que sólo varían su parte baja. El ejemplo 2 representaría lo mismo con variables de 32 bits y el ejemplo 3 con variables de 8 bits. El ejemplo 4 representaría variables de diferente tamaño.

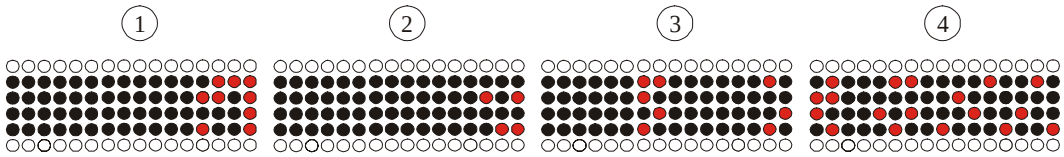


Figura 7.6. Ejemplos de patrones de error causados por escrituras no efectivas

Accesos a direcciones erróneas

En este caso, la escritura (o lectura) es efectiva pero se realiza sobre una dirección diferente a la establecida. En una escritura, el valor contenido en la dirección deseada no se actualiza. Igual que en el caso anterior, el bloque de datos quedaría formado erróneamente tanto por palabras correctas como incorrectas. Sin embargo, en este caso, las palabras del bloque de datos afectadas no serán necesariamente consecutivas. Por ejemplo, dada la siguiente secuencia de acceso:

Secuencia de acceso esperada	*Secuencia real con fallo*
0000 0000 0110 0010	0000 0000 0110 0010
0000 0000 0110 0011	0000 0000 0110 0011
…	…
0000 0000 0110 0111	0000 0000 0110 0011
0000 0000 0110 1000	0000 0000 0110 1000

Una inversión o pegado a 0 en la línea 3 que fuerce un error simple, afectando sólo a una dirección, provoca que el valor contenido en la dirección 67_{hex} no se actualice correctamente. Es más, el valor contenido en la dirección 63_{hex} será sobrescrito con un valor no deseado. El fallo debe coincidir con la escritura o lectura de datos. Este tipo de accesos erróneos se observaron tanto con fallos físicos a nivel de pin localizados en el bus de direcciones como fallos SWIFI en registros del controlador [Ademaj 2003].

7.2.1.3. Errores derivados de fallos múltiples

El único mecanismo de detección de errores definido en el TTP es el CRC de la trama transmitida por el canal de comunicaciones. Pero la transmisión serie no el la única susceptible de fallo. Pueden ocurrir múltiples fallos en cualquier instante de la relación secuencial de la figura 7.2. no siendo necesariamente conexos o coincidentes en localización o instante de aparición. Los patrones de error derivados de fallos múltiples son muy diversos y probablemente los más difíciles de detectar.

Hasta el momento, el efecto de los errores múltiples en memoria de datos no se ha considerado en profundidad en los trabajos prácticos de validación experimental relacionados con técnicas de inyección de fallos. De hecho, el modelo de inversión simple sigue siendo muy utilizado sobre todo en inyección SWIFI. Es cierto que la inversión de

un solo bit pueden tener un gran impacto en memoria de código durante la decodificación de instrucciones, ya que la inversión de un bit podría modificar una instrucción de salto, una llamada del sistema operativo o cambiar una instrucción válida por otra igualmente válida. Sin embargo, el impacto en memoria de datos es mínimo. La inversión de un bit se puede detectar fácilmente con un código de paridad. Son los errores múltiples los que precisan de una detección más compleja en sistemas que requieren alta confiabilidad, y es la detección de estos errores lo que motiva el estudio de la mejora de la cobertura en la arquitectura TTA en los siguientes apartados de este capítulo.

7.2.2. Errores detectados *vs.* errores no detectados

A cada TDMA un nodo transmite una trama, bien de sincronización o bien de datos. En el caso de una trama de datos, se transmiten los valores correspondientes a una o varias variables del algoritmo de control, las que forman el bloque de datos. En total, un máximo de 120 palabras.

El protocolo TTP utiliza un esquema de tiempos fijo, no sólo para la transmisión de las tramas, sino también para la ejecución de tareas a nivel de aplicación. Podemos diferenciar tres tipos de tareas, las del algoritmo de control implementado por el usuario, las tareas del sistema operativo y las tareas de lectura y escritura sobre la CNI. Tanto el instante de inicio de la tarea como sus márgenes temporales de ejecución en el caso típico y peor de los casos establecen el esquema de ejecución. Utilizar una carga sintética como algoritmo de control que ejercite este esquema de tiempos puede resultar de gran utilidad a la hora de clasificar los errores derivados de un tipo de fallo y el comportamiento del controlador. A nivel de pin, los fallos pueden localizarse en las líneas de control o en los buses de direcciones y datos de las interfaz de comunicaciones y la interfaz del descriptor de mensajes[107].

7.2.2.1. Eliminación de pulsos en las líneas de control

La degradación de las líneas de control puede analizarse mediante la inyección de fallos permanentes.

Interfaz de comunicaciones: Los resultados muestran la no ocurrencia de averías. Todos los errores activados provocan excepciones o bien causan otros errores internos que son detectados y aislados en la primera zona de contención.

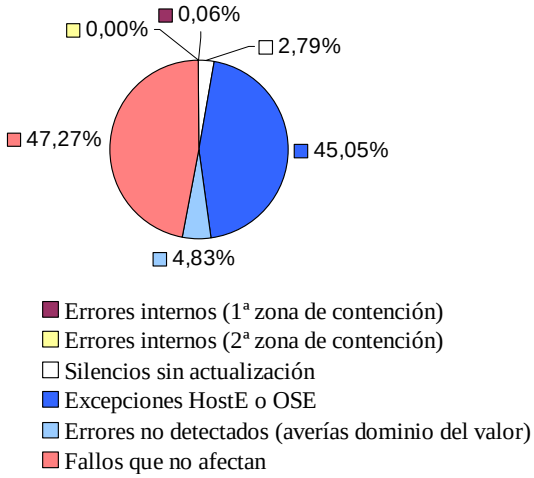
Interfaz del descriptor de mensajes: La mayor parte de los errores se detectan con el CRC asociado a la transferencia de datos entre la MEDL y la PCU. Existe un bajo porcentaje de errores que causan otros errores internos detectados y aislados en la primera o segunda zona de contención.

7.2.2.2. Variación de pulsos existentes y generación de pulsos inexistentes

Interfaz de comunicaciones en el TTP/C-C1: Los experimentos de inyección de fallos llevados a cabo con el TTP/C-C1 en el bus de datos y direcciones de la CNI se realizan utilizando un *disparo por tiempo* que selecciona la expiración de un tiempo aleatorio desde el inicio del experimento hasta que el fallo es forzado en la localización deseada. La gráfica 7.1. muestra un porcentaje muy elevado de excepciones. Se debe a la relativamente baja utilización de los buses para accesos a la CNI en comparación con los accesos a la propia memoria externa del microcontrolador. Por tanto, es deseable poder

[107] Esta última sólo para el controlador TTP/C-C1

sincronizar la inyección de los fallos con los accesos a la CNI, sincronización que se realiza en los experimentos llevados a cabo con el controlador TTP/C-C2.



Gráfica 7.1. Datos obtenidos con un disparo por tiempo aleatorio en el TTP/C-C1[108]

Interfaz de comunicaciones en el TTP/C-C2: En la figura 7.7. se clasifican los eventos observados inyectando fallos físicos transitorios a nivel de pin durante la escritura del bloque de datos en memoria. El objetivo de estos experimentos es comprobar si realmente es necesaria la implementación de mecanismos de detección de errores.

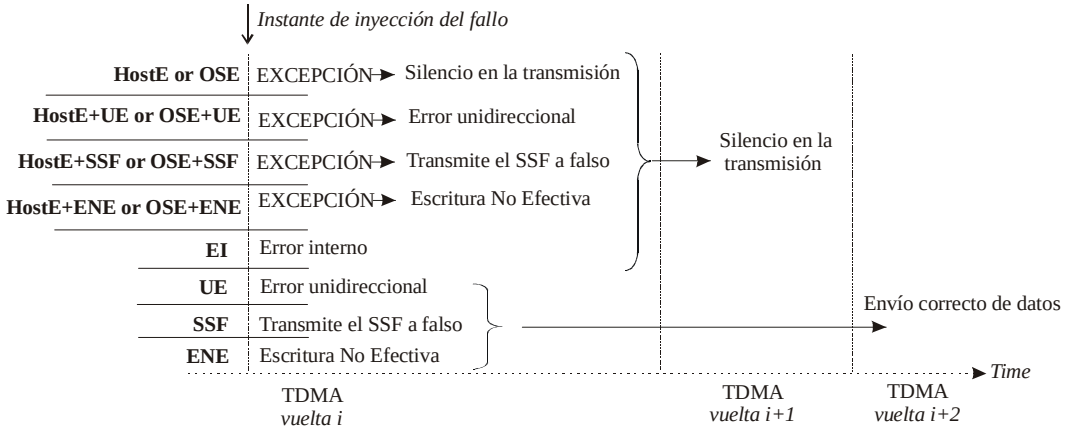


Figura 7.7. Eventos observados por la activación de un error

- ***HostE*** – Excepción del microcontrolador que aloja las tareas del algoritmo de control, del sistema operativo y de lectura-escritura en memoria. El microcontrolador PPC555 detecta un error de bus.
- ***OSE*** – Excepción del sistema operativo. Se detecta que el tiempo de ejecución de la tarea en curso desborda del margen establecido.
- ***EI*** – Errores internos. La gestión de errores del controlador TTP/C detecta un error relacionado con el protocolo de comunicaciones, de pérdida de pertenencia, un error de sincronización con el nivel de aplicación o un error interno detectado por el guardián del bus.
- ***SSF*** – "Sender Status Flag". El nodo receptor del mensaje detecta el SSF de

[108] Silencios sin actualización: Fallos de diseño e implementación observados con el algoritmo de vida.

alguna de las palabras recibidas a falso. El SSF es un mecanismo implementado a nivel de aplicación[109]. Consiste en el envío de una palabra adicional por cada 16 palabras transmitidas. Cada bit de la palabra adicional es asignado a una de las palabras transmitidas diferenciando, con el valor contenido en el bit, el estado de la palabra entre válida o no válida. El SSF de cada palabra se escribe y se lee sólo a nivel de aplicación.

- ***UE*** – Errores unidireccionales en el bloque de datos.
- ***ENE*** – Errores aleatorios múltiples en el bloque de datos. Evaluando sólo los fallos a nivel de pin en el bus de datos durante tareas de escritura, todos los errores corresponden a escrituras no efectivas en el bloque de datos transmitido.
- ***HostE + UE*** (o ***OSE + UE***) – A pesar de la excepción ocurrida, existe una transmisión con errores unidireccionales.
- ***HostE + ENE*** (o ***OSE + ENE***) – A pesar de la excepción ocurrida, existe una transmisión con errores aleatorios múltiples.
- ***HostE + SSF*** (o ***OSE + SSF***) – A pesar de la excepción ocurrida, existe una transmisión con errores múltiples debido a escrituras no efectivas, llevando marcados los correspondientes SSF a falso.

La tabla 7.1. muestra los resultados de 3.000 inyecciones efectivas en el bus de datos durante escrituras del microcontrolador PPC555 en la CNI del TTP/C-C2. 1.500 inyecciones corresponden a pegados a 0 y otras 1.500 corresponden a pegados a 1, fallos de duración uniformemente distribuida entre 500ns y 5μs.

		HostE		
Errores Activados	HostE	**HostE+UE**	HostE+SSF	**HostE+ENE**
Pegado a [0]: 76,39%	19,28%	**1,54%**	1,54%	**0%**
Pegado a [1]: 94,95%	49,25%	**0%**	15,54%	**0,53%**

		OSE		
Errores Activados	OSE	**OSE+UE**	OSE+SSF	**OSE+ENE**
Pegado a [0]: 76,39%	0%	**0%**	0%	**0%**
Pegado a [1]: 94,95%	16,37%	**0,53%**	1,73%	**0%**

Errores Activados	SSF	**UE**	**ENE**	EI
Pegado a [0]: 76,39%	0,37%	**37,10%**	**16,57%**	0%
Pegado a [1]: 94,95%	0%	**11,01%**	**0%**	0%

Tabla 7.1. Muestra de 3.000 inyecciones efectivas en el bus de datos

Se consideran **errores activados** al porcentaje de fallos que desvían la función normal del nodo, por ejemplo debido al tratamiento de errores o excepciones, o que provocan una violación de la hipótesis de fallos o una avería ya sea en el dominio del tiempo o del valor.

[109] Mecanismos conocidos como "end-to-end", ya que se aplican al nivel más alto ISO/OSI disponible del protocolo de comunicaciones, normalmente al nivel de aplicación. El TTP implementa tres niveles: físico, enlace y aplicación. El nivel de enlace está gestionado íntegramente por el controlador TTP/C, mientras que el nivel de aplicación se implementa sobre el microcontrolador PPC555.
A nivel de enlace, los datos se encuentran protegidos por un CRC que se incluye en la trama. Sin embargo, a nivel de aplicación no existe ningún código de protección para los datos transmitidos desde el PPC555 al controlador TTP/C (y viceversa), ni tampoco para protección de los datos durante el tiempo que permanezcan en la CNI antes de ser transmitidos. Se suponía que el SSF sería suficiente para detectar posibles errores.

En la tabla se aprecian varias averías en el domino del valor. Son los casos marcados como *UE* y *ENE*, donde se transmite un mensaje semánticamente incorrecto. En los casos marcados como *HostE + UE*, *HostE + ENE* y *OSE + UE* se detecta una excepción pero no se impide la propagación de un error *UE* o *ENE* a través del canal de comunicaciones. Este tipo de averías resultan especialmente importantes en algoritmos de control distribuidos. Por ejemplo, un nodo realiza la lectura de un sensor cuyo valor es necesario en el algoritmo de control de otro nodo. La propagación del error es cierta, ya que se trata de nodos físicamente separados donde el nodo receptor no puede saber si la información recibida es correcta o no, por eso se asume que sólo se transmite si se pueden asegurar datos libres de errores[110].

Además, el error no se produce por el cálculo o ejecución del algoritmo de control sino por la transferencia o almacenamiento de los datos calculados. Tanto en la transferencia como en el almacenamiento interviene una barrera de contención, la CNI. Por la tanto, la detección de estos errores debería ser parte de las acciones realizadas por el controlador de comunicaciones si se quiere asegurar un comportamiento silencioso en el dominio del valor.

En la tabla 7.1. se muestra que la barrera de contención no es capaz de detectar errores causados por fallos a nivel de pin. Existe un 55,21% ± 4,7% de averías (mensajes semánticamente incorrectos) con pegados a 0 y un 12,07% ± 2,8% con pegados a 1. Comparando estos resultados con otra muestra de 3.000 experimentos adicionales con fallos dobles en líneas adyacentes, obtenemos que el número de averías con pegados dobles a 0 disminuye al 50,60% ± 4,7% debido al incremento de excepciones. Con pegados dobles en diferente dirección, una línea a 0 y otra línea a 1, el porcentaje de averías observadas es del 31,91% ± 4,3%.

Los márgenes de error asignados a cada porcentaje representan la diferencia entre el porcentaje de averías observado y el porcentaje de averías mínimo estimado para una muestra de *Y* inyecciones efectivas y *X* errores no detectados según una distribución F con ν_1, ν_2 grados de libertad [Cukier *et al*. 1999 (A.4)]:

$$\text{Porcentaje de averías mínimas estimadas} = \frac{(X+1)\,F_{2(X+1),\,2(Y-X),\,95\%}}{(Y-X)+(X+1)\,F_{2(X+1),\,2(Y-X),\,95\%}}$$

En general, el número de averías observadas es alto y justifica la necesidad de implementar nuevos mecanismos para la prevención de averías en el dominio del valor.

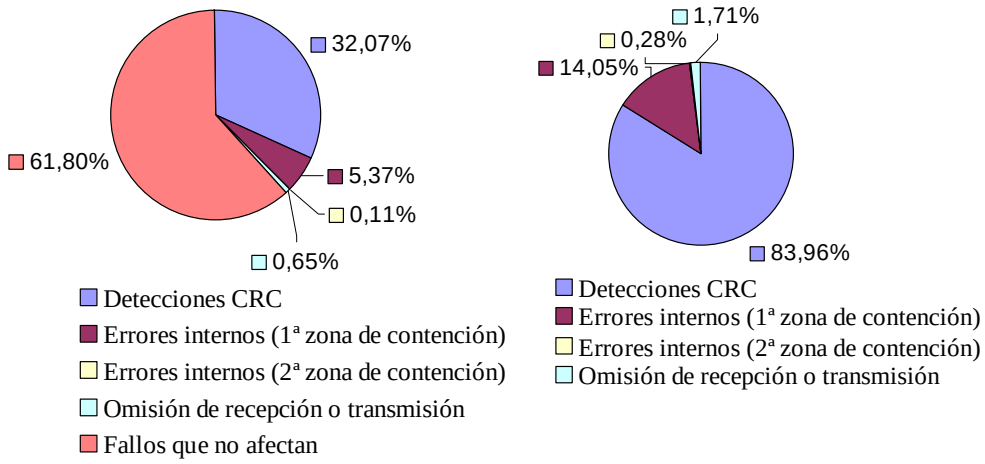
La interfaz del descriptor de mensajes: Para evaluar la cobertura de detección del CRC en el TTP/C-C1 asociado a la transferencia de datos entre la MEDL y la PCU se inyectan fallos transitorios en los buses de direcciones y datos de esta interfaz. Se considera un disparo aleatorio de la inyección no sincronizado con los periodos de acceso a la MEDL.

Se observa un 61,8% de inyecciones efectiva que no activan ningún error (gráfica izquierda 7.2.), lo que indica que más de la mitad de las inyecciones coincidieron con periodos de inactividad en los buses, donde el fallo no conlleva error o alteración de la función del sistema. Sobre los errores activados (gráfica derecha 7.2.), no todos los errores son detectados por el cálculo del CRC. Aquellos que sí son detectados por este mecanismo activan un error de protocolo[111]. Existe un porcentaje de errores que causan

[110] Fail-silent behaviour
[111] Protocol error

otros errores internos que son detectados y aislados en la primera zona de contención (14,5%) o en la segunda zona de contención (0,28%).

Como se aprecia en la gráfica, el CRC implementado no cubre todos los errores. El valor erróneamente leído en el descriptor de mensajes, ya sea por un error unidireccional o aleatorio múltiple, altera el esquema de tiempos en la recepción o transmisión de mensajes haciendo que el controlador desatienda la recepción de una trama o no habilite una transmisión. El peor de los casos es la omisión de una recepción, ya que el algoritmo de control acepta como actualizados unos valores en el área de mensajes de la CNI que realmente no han sido actualizados en la última recepción. Por tanto, se puede producir el cálculo de las variables utilizando datos erróneos.



Gráfica 7.2. Distribución de errores en la interfaz del descriptor de mensajes

La cobertura real de detección del CRC sobre los errores activados es del 83,96%, mientras que este mismo mecanismo aplicado a las transmisiones serie ha demostrado una cobertura cercana al 99,9%. La diferencia estriba en el tipo de transmisión, ya que los errores unidireccionales son típicos de la transmisión en paralelo. Además, para bloques de datos pequeños[112], el fallo puede provocar fácilmente escrituras no efectivas del bloque completo, incluyendo los 16 bits del CRC, lo que impide que se detecte el error.

7.3. Evaluación de los Resultados

Después de las observaciones realizadas con la carga sintética, resulta interesante utilizar un algoritmo de control real para un ajuste más preciso del esquema de ejecución. El algoritmo proporcionado por VOLVO Technological Development durante el proyecto FIT [FIT 2002] es el de frenado por cable "*brake-by-wire*" para el control activo del ABS.

El algoritmo utiliza cuatro nodos: el vehículo, el pedal, la rueda frontal izquierda y una réplica:

– El *nodo vehículo* simula las tres ruedas restantes y calcula la velocidad global del vehículo en *m/s*, la velocidad de las cuatro ruedas en *rad/s* y la distancia recorrida. Utiliza dos parámetros: la aceleración del vehículo y la fuerza de frenado, que son calculados y enviados desde el *nodo pedal* en función de los ángulos de ambos pedales. En función de estos parámetros, de la última

[112] El máximo tamaño de bloque especificado es de 12×16, aunque el más frecuente es de 4×16.

velocidad del vehículo calculada y de la velocidad registrada en cada rueda, calcula tanto la nueva velocidad que debe llevar el vehículo como la velocidad que debe llevar cada rueda en particular.

– La velocidad del vehículo y la velocidad de la rueda son parámetros de entrada en el *nodo rueda frontal izquierda* y su réplica. Este nodo implementa el algoritmo de frenado, calculando la fuerza de fricción en el disco de frenado de la rueda y envía la nueva velocidad adquirida al nodo vehículo (ver figura 7.8. y tabla 7.2.).

Las inyecciones se localizan en el *nodo rueda*. Los datos enviados por la réplica libre de fallos se utilizan en la segunda tarea software del monitor.

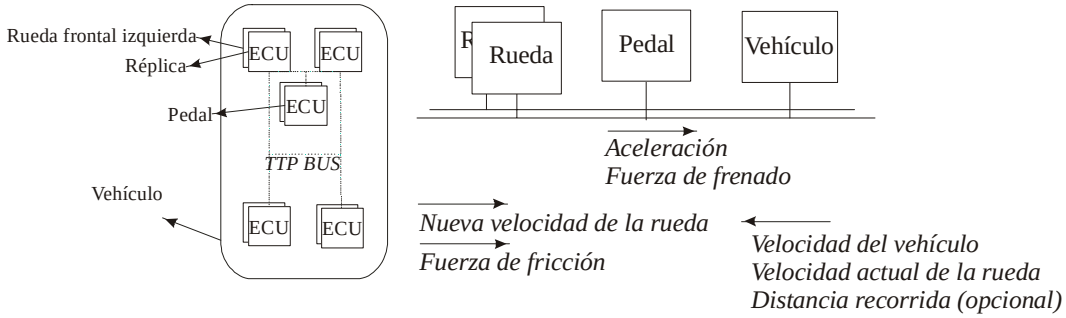


Figura 7.8. Transferencia de variables para el control del ABS

	Nodo Rueda (2×)	**Nodo Pedal**	**Nodo Vehículo**
Bucle de control	ABS, 200Hz	Fuerza de frenado, 50Hz	Simula el vehículo, 200Hz
Entradas sensores	Velocidad actual (rueda), 200Hz[113]	Ángulos pedales	
Entradas por el TTP Bus	Fuerza de frenado, 50Hz Velocidad actual (rueda), 200Hz		Fuerza de frenado, 50Hz Aceleración, 50Hz Velocidad actual (rueda), 200Hz
Salidas por el TTP Bus	Fuerza de fricción, 200Hz Velocidad real, 200Hz	Fuerza de frenado, 50Hz Aceleración, 50Hz	Velocidad del vehículo, 50Hz Velocidad actual (rueda), 200Hz Distancia recorrida (opcional)

Tabla 7.2. Frecuencias de transmisión de las variables para el control del ABS

Un código sencillo de detección de errores aplicable a bloques de transmisión de tamaño variable es un código de paridad bit por línea del bus de datos. Aplicamos un bit de paridad a cada combinación vertical del bloque (figura 7.3.) aumentando en sólo una palabra la transmisión. Se obtienen los siguientes errores no detectados:

Fallos en el bus de datos sincronizados con ciclos de escritura

Duración del fallo	500ns … < 750ns	750ns … <2μs
Fallos que no afectan	58,86%	33,71%
Detectados por el código	31,61%	33,6%
Errores no detectados	**1,59% ± 0,4%**	**9,14% ± 3%**

[113] Al no existir sensor que determine la velocidad actual de la rueda, el valor que toma esta variable es la velocidad actual calculada y enviada desde el nodo vehículo.

Fallos en el bus de datos sincronizados con ciclos de lectura

Duración del fallo	500ns … < 750ns	750ns … <2µs
Fallos que no afectan	71,87%	34,92%
Detectados por el código	0%	21,89%
Errores no detectados	**0% ± 1%**	**0% ± 0,4%**

Fallos en el bus de direcciones sincronizados con ciclos de escritura

Duración del fallo	500ns … < 750ns	750ns … <2µs
Fallos que no afectan	83,23%	55,8%
Detectados por el código	1,86%	11%
Errores no detectados	**2,48% ± 0,6%**	**2,36% ± 0,8%**

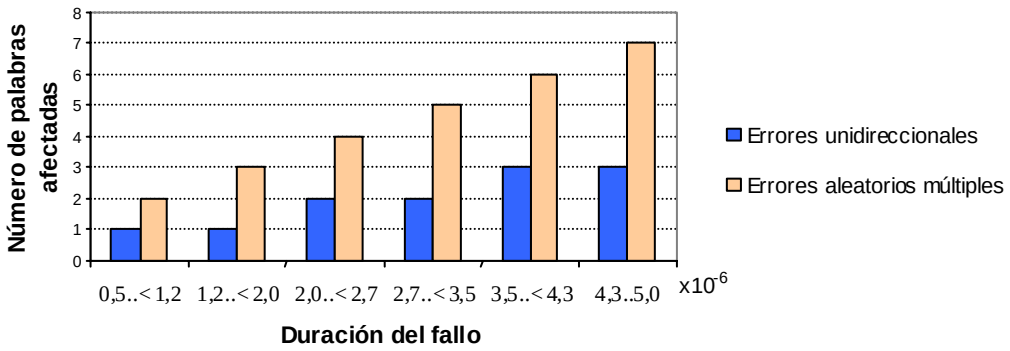
Fallos en el bus de direcciones sincronizados con ciclos de lectura

Duración del fallo	500ns … < 750ns	750ns … <2µs
Fallos que no afectan	58,69%	33,11%
Detectados por el código	0%	14,89%
Errores no detectados	**0% ± 0,7%**	**3,05% ± 1%**

Tablas 7.3. Errores no detectados por un código de paridad bit por columna

La tabla 7.3. desglosan los resultados obtenidos en una muestra con 1.600 fallos simples efectivos (pegados a 0 y 1) uniformemente distribuidos en localización y duración. El porcentaje de *errores no detectados* más elevado corresponde a fallos localizados en el bus de datos y coincidentes con ciclos de escritura. Este es el peor de los casos, y comparando los resultados con los obtenidos con la carga sintética, observamos que sigue siendo un porcentaje elevado que justifica la necesidad de implementar algún mecanismo de protección adicional.

El número de bits erróneos que podemos observar en una combinación vertical depende del número de palabras afectadas por el fallo. Hasta 2µs, el número de errores simples observados en las combinaciones verticales sigue siendo mayor que el número de errores múltiples, donde varias palabras se ven afectadas por el fallo. Pero para fallos de mayor duración un código de paridad no es suficiente. La relación existente entre la duración del fallo y el número máximo de palabras afectadas se muestra en la gráfica 7.3. que diferencia si el fallo causa un error unidireccional, o si causa escrituras no efectivas o incorrectas generando un error aleatorio múltiple.



Gráfica 7.3. Relación entre la duración del fallo y el número de palabras afectadas

Incluso con fallos de duración inferior a 1,4μs (duración de cada acceso a la CNI) se pueden provocar errores en más de una palabra. Con esta motivación, la siguiente sección presenta un análisis para la mejora de la cobertura de detección de errores tanto unidireccionales como errores aleatorios múltiples.

7.4. Propuesta para la Mejora de la Cobertura de Detección

Aplicar un código de detección de errores sobre datos puede mejorar la cobertura frente a fallos ocurridos durante su transmisión y almacenamiento. La primera vez que se transmiten los datos es entre el nivel de aplicación y la memoria de mensajes. Por tanto, el código debe ser incluido como parte de los datos a nivel de aplicación. Su comprobación también se realiza a nivel de aplicación en el nodo receptor.

En este apartado se estudia cómo mejoraría la cobertura de detección ante errores múltiples con diferentes alternativas, tales como el código de detección de errores cíclico redundante, códigos separables verticales o posibles alternativas que impliquen más de un código.

7.4.1. Código de detección cíclico redundante

Al igual que en la interfaz del descriptor de mensajes del TTP/C-C1, también se puede añadir un CRC de 16 bits a los bloques de datos que se escriben o se leen a través de la interfaz de comunicaciones. Como se muestra en la figura 7.9. el código CRC_{16} se calcula alineando las palabras según el orden de transmisión, y se añade al bloque de datos forzando un acceso más a memoria (acceso *i* de la figura).

Para un CRC con un polinomio generador de grado *k* $G(x)_k$, representamos con $\overline{P}(x)_g$ los patrones de error de grado *g* no detectados por el código:

$$\overline{P}(x)_{16} = G(x)_k \cdot 1 \qquad \overline{P}(x)_{18} = \begin{array}{l} G(x)_k \cdot x^2 \\ G(x)_k \cdot (x^2 + 1) \\ G(x)_k \cdot (x^2 + x) \\ G(x)_k \cdot (x^2 + x + 1) \end{array}$$

$$\overline{P}(x)_{17} = \begin{array}{l} G(x)_k \cdot x \\ G(x)_k \cdot (x + 1) \end{array} \qquad \overline{P}(x)_{19} = \begin{array}{l} G(x)_k \cdot x^3 \\ G(x)_k \cdot (x^3 + 1) \\ G(x)_k \cdot (x^3 + x) \\ G(x)_k \cdot (x^3 + x + 1) \\ G(x)_k \cdot (x^3 + x^2) \\ G(x)_k \cdot (x^3 + x^2 + 1) \\ G(x)_k \cdot (x^3 + x^2 + x) \\ G(x)_k \cdot (x^3 + x^2 + x + 1) \end{array}$$

…

En general, en un bloque de datos de *n* bits (incluyendo los bits del código), no se detectan los patrones representados por $\overline{P}(x)_{(n-1)}$, $\overline{P}(x)_{(n-2)}$, $\overline{P}(x)_{(n-3)}$, …, $\overline{P}(x)_k$ siendo *k* el grado del polinomio generador y $n > k$. El número de errores no detectables por el CRC sería: $(2^0 + 2^1 + 2^2 + \ldots + 2^m) = (2^{m+1} - 1)$, siendo $m = (n - k - 1)$. Lo que implica $(2^{n-k} - 1)$ errores no detectados, entre 2^n posibles combinaciones.

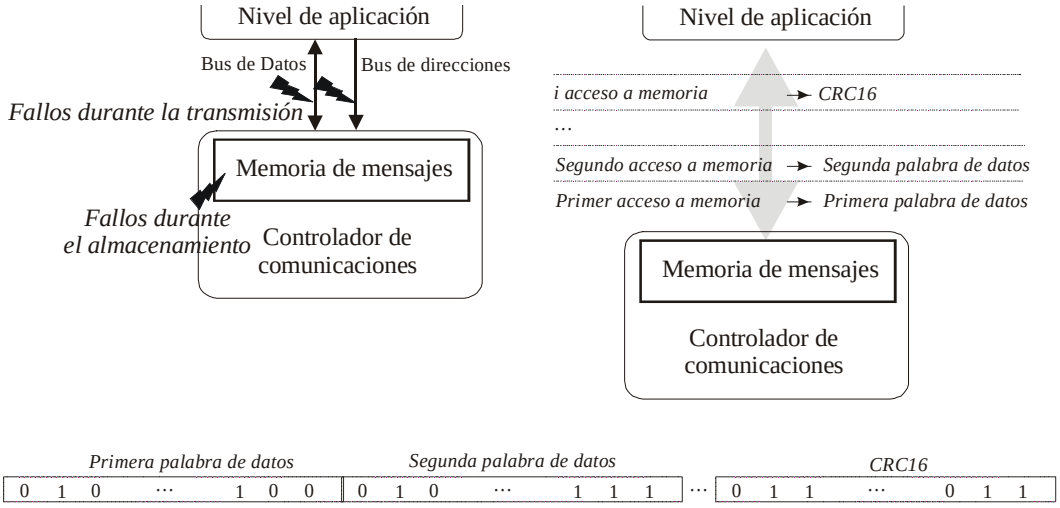


Figura 7.9. Consideración del bloque de datos con un CRC_{16}

La cobertura teórica no pondera las probabilidades de ocurrencia de un tipo de error determinado sobre las combinaciones con mayor frecuencia de aparición. El polinomio generador utilizado para la transferencia de bloques de datos desde la MEDL (descriptor de mensajes) es $x^{16}+x^{12}+x^{11}+x^{8}+x^{5}+x^{4}+1$, lo que supone una distancia mínima de Hamming de 5 para bloques de datos de 257 bits o inferiores. Teniendo en cuenta que los algoritmos de control "*x-by-wire*" utilizan bloques de datos más cercanos a diez bytes que a cien bytes, la cobertura que ofrece este CRC_{16} sería muy alta ante errores múltiples derivados de fallos simples, salvo los derivados de escrituras no efectivas que afecten a más de un ciclo de acceso a memoria o los accesos a direcciones erróneas. La distribución de bits erróneos en el bloque de datos como consecuencia de una escritura no efectiva es muy variable. La figura 7.10. muestra posibles distribuciones de seis errores en dos palabras consecutivas en un bloque de datos (fallos de 1,2µs que afecten a dos ciclos de escritura) no cubiertas por el CRC_{16}.

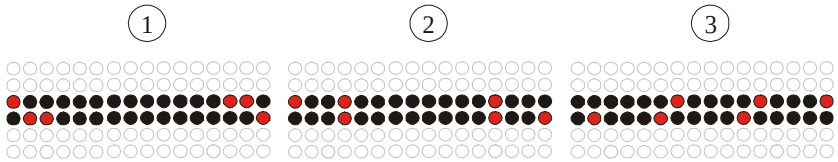


Figura 7.10. Ejemplos de patrones no detectables con el CRC_{16}

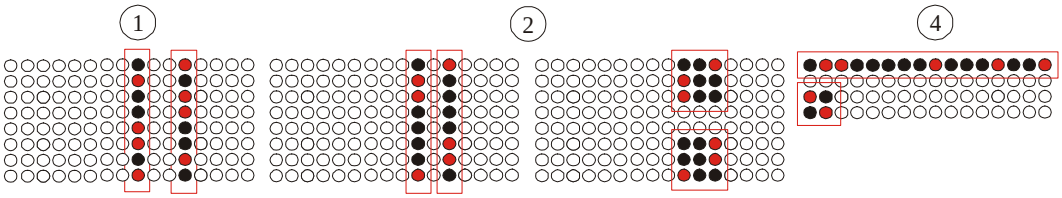


Figura 7.11. Ejemplos de errores causados por fallos múltiples

Algunos de los patrones de error no detectados podrían ser causados también por fallos múltiples. El ejemplo 2 de la figura 7.10. podrían ser consecuencia de dos SEU durante el almacenamiento de datos en memoria. El ejemplo 1 de la figura 7.11. puede ser causa de dos pegados en el bus de datos, mientras que el ejemplo 2, aunque afecta a dos líneas del bus de datos, por la proximidad de las inversiones también representaría dos

SEU. Además, existen patrones alcanzables con fallos de diferente origen, como el ejemplo 3 de una escritura no efectiva y un SEU.

Para un bloque de datos de *n* bits, obtenemos $2^{(n\ -\ 16)}$ – 1 patrones de error no detectados. Esta cifra, frente al número total de combinaciones es muy baja, sin embargo, no podemos asegurar que experimentalmente la cobertura no se vea reducida debido a la alta probabilidad que pudieran ofrecer algunos de estos patrones de error, consecuencia de escrituras no efectivas o fallos múltiples, en las combinaciones más frecuentes.

Al decir que un fallo causa varias escrituras no efectivas, dichas escrituras corresponden a accesos a memoria consecutivos. Por tanto, dos escrituras no efectivas, analizando cada combinación vertical (*o columna*) del bloque, pueden provocar la inversión de bits adyacentes en la combinación vertical, error que podría ser fácilmente detectado por un código de detección de errores vertical *CDEV*. El número máximo de inversiones adyacentes detectables dependerá del tipo de CDEV utilizado. Además, el hecho de disponer de un código de detección por cada columna y de que sean independientes entre sí, podría significar una mejora frente a fallos múltiples, asumiendo que a mayor número de columnas con error mayor probabilidad de que se detecte al menos uno, lo que ya implicaría el rechazo del bloque.

7.4.2. Códigos de detección verticales

Otra alternativa para mejorar la cobertura de detección es la utilización de un CDEV. En la figura 7.12. se muestra la formación del bloque de datos con los bits de información y con los bits correspondientes a un CDVE separable.

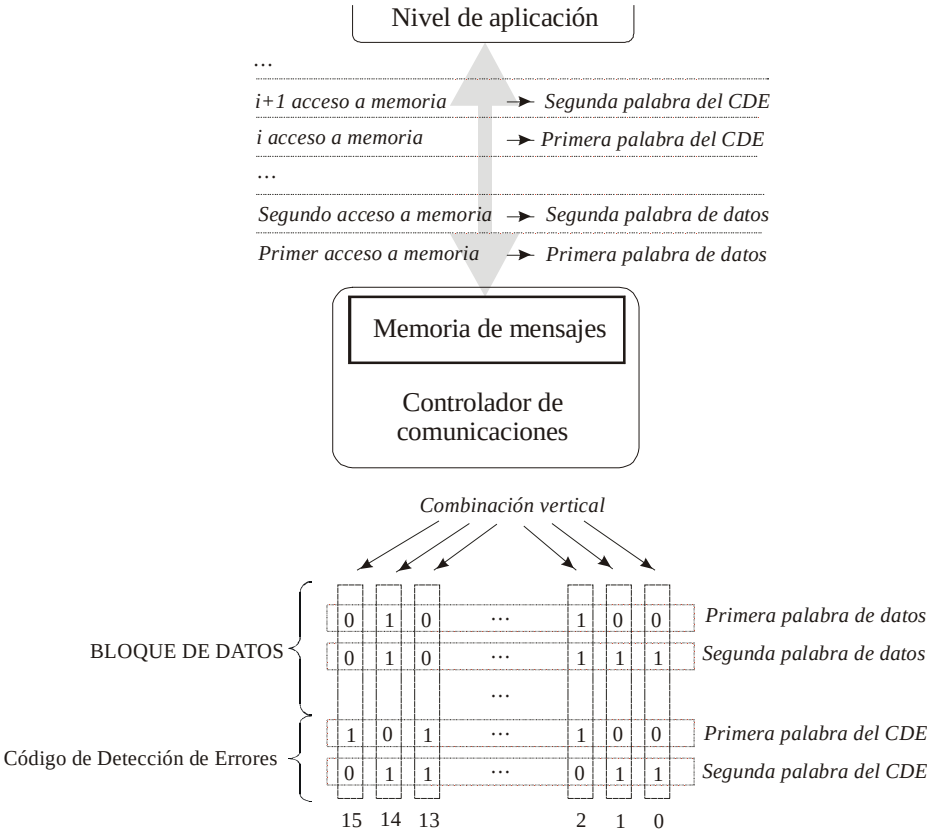


Figura 7.12. Consideración del bloque de datos con un CDE separable vertical

Cada combinación vertical incluye los bits transmitidos por la misma línea del bus de datos. Por tanto, cada bloque de datos incluirá 16 combinaciones verticales compuestas por **bits de información** y **bits de código**.

7.4.2.1. CRC con polinomio generador de grado 3

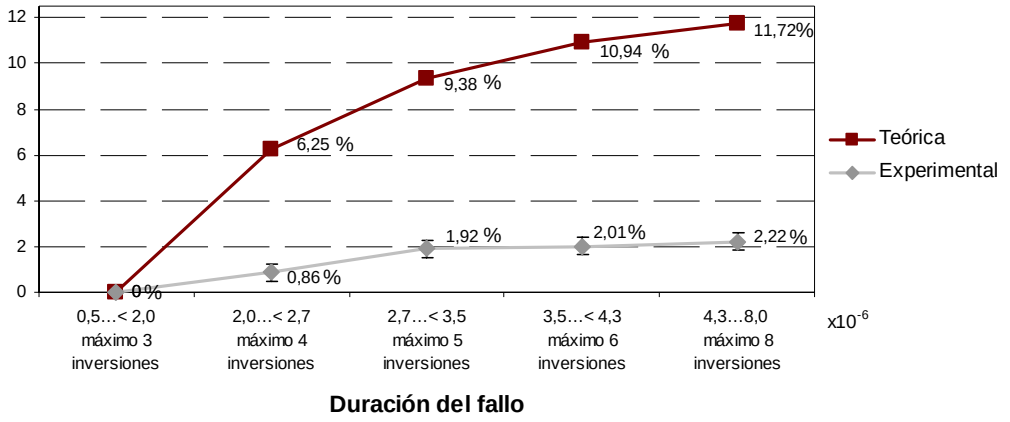
Es posible combinar las ventajas de la alta cobertura de detección teórica que presenta un CRC con las ventajas que aportaría un CDEV ante escrituras no efectivas o fallos múltiples, construyendo un CRC_3 vertical con un polinomio generador $G(x)_3 = x^3 + 1$.

Para comprobar experimentalmente la mejora que se obtiene en la cobertura de detección, se analiza una muestra 3.000 experimentos en la que se observan:

Tipo de fallo	*Inyecciones efectivas*	*Nº de bloques de datos erróneos transmitidos*	*Nº total de combinaciones verticales erróneas*
Pegado simple a [1]	1.376	588	1.694
Pegado doble a [0,1]	1.411	1.170	3.945

La diferencia numérica entre los pegados simples y dobles se debe a que los pegados dobles causan, sobre los valores transmitidos por el algoritmo de control activo del ABS, y al igual que los pegados simples a 0, más escrituras no efectivas (un 65% aproximadamente) que errores unidireccionales (un 35% aproximadamente), mientras que los pegados simples a 1 invierten estos porcentajes, motivo por el cual se han utilizado tanto pegados simples a 1 como pegados dobles, una línea a 0 y otra a 1, para obtener una muestra que incluya tanto errores unidireccionales como escrituras no efectivas.

La duración de los fallos varía desde 0,5µs a 8µs. La gráfica 7.4. presenta una comparación entre el porcentaje teórico de errores no detectados y el porcentaje obtenido experimentalmente. Según la duración del fallo, se ha estimado el número máximo de inversiones que pueden aparecer por combinación vertical. Además, la curva con las observaciones experimentales incluye las barras que identifican el margen de error para cada valor calculado. De los cinco rangos considerados relativos a la duración del fallo, el valor obtenido en [4,3µs… 8,0µs] no incluye fallos que causan la escritura no efectiva del bloque de datos completo[114], ya que éste es un error específico que necesitaría de algún mecanismo adicional para ser detectado.

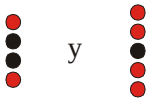


Gráfica 7.4. Relación entre la duración del fallo y el porcentaje de errores no detectados

[114] 6×16 bits de información, 16 bits de los SSF y 16 bits del CRC. En total, 8×16 bits.

En la gráfica 7.4. se observa una variación muy baja de los valores que forman el bloque de datos a transmitir. Existen combinaciones que se transmiten muy frecuentemente, siendo el impacto de un fallo sobre estas combinaciones lo que marca la desviación entre la cobertura real y la teórica. Si los patrones de error que aparecen con más frecuencia se detectan con el CDE utilizado, se puede considerar la cobertura observada como optimista, y viceversa. Además, la cobertura teórica contempla todos los posibles patrones de error, mientras que la observación experimental revela sólo aquellos que implican accesos consecutivos a memoria.

Un dato a considerar es la carencia de la cobertura a nivel de bloque, o lo que es lo mismo, cuando ninguno de los 16 CDEV del bloque de datos detecta alguna combinación vertical errónea habiéndose producido inversiones en los bits que componen el bloque. En la muestra se ha observado una carencia de siete bloques no detectados sobre un total de 1.758 bloques transmitidos que contenían algún error. Los patrones de error no detectados que se repiten con más frecuencia son:



Sería posible reducir el número de combinaciones que responden a estos patrones si tenemos en cuenta la paridad. El código Berger, por ejemplo, aplicado como un CDEV es capaz de detectar cualquier error unidireccional de cualquier tamaño de ráfaga[115], reduciendo a su vez los errores aleatorios no detectables a aquellos cuya inversión múltiple de bits de información mantenga la misma paridad que la original.

7.4.2.2. U_n: Detección de errores unidireccionales

Se propone en este capítulo el código U_n como una variación del código Berger, donde *n* es el número de bits que forman el código de detección que se añade a los bits de información, orientado a la detección de errores unidireccionales y la mejora en la cobertura de detección de errores aleatorios.

Concepto básico: *A*) siendo *m* el número de bits de información y *n* el número de bits del código añadido, el CDEV deberá detectar cualquier error unidireccional ocurrido sobre (*m* + *n*) y *B*) siendo c_i cualquier combinación de *m* bits de información con *i* representando el número de unos, el código asignado a c_i deberá incrementar la distancia de Hamming entre dos combinaciones c_i y c_j con i≠j en al menos tres bits no unidireccionales si $i \neq 0$ o (*m*-1) o en al menos dos si $i = 0$ o (*m*-1).

Reglas de composición: siendo *m* el número de bits de información y *n* bits de código, *n* debe ser el mínimo número tal que $n \geq log_2 (m + 1)$.

Definición 1: siendo *m* el número de bits de información, sea C el conjunto de las 2^m *m*-tuplas binarias. C se puede estructurar como $\bigcup_{i=0}^{m} C_i$, tal que $C_i = \{ c_i^0, c_i^1, ..., c_i^{\frac{m!}{(m-i)!i!}} \}$, con *i*

[115] Distancia entre el primer y último bit invertido

representando el número de unos contenidos en los bits de información, donde $C_i \cap C_j = \varnothing$, si $i \neq j$.

Definición 2: siendo *n* el número de bits que forman un código, sea U_n el conjunto de las 2^n *n*-tuplas binarias. Cada *n*-tupla es un posible código a asignar. u_k representa el conjunto de todos los códigos con *k* número de ceros, para todo *k* entre 0 y *n*. Todos los códigos pertenecientes a U_n se pueden ordenar en un vector $Y = [\, y_0, y_1, ..., y_{(2^n-1)} \,]$.

Para ordenar el vector consideramos que si y_i pertenece a u_k y y_j pertenece a u_p donde $k \neq p$, entonces se satisface que $y_i < y_j$ siempre que k<p. Pero si y_i y y_j pertenecen al mismo u_k, la posición relativa que ocupen es indiferente. Por tanto, es posible obtener más de un vector. Entre los vectores posibles, el vector seleccionado finalmente debe cumplir la condición de asignación.

Asignación: *Un código y_i que pertenezca a un vector Y se asigna a todas las combinaciones incluidas en $C_k \in$ C, siendo k el número de unos en los bits de información.*

Condición de asignación: *Siendo y_i, y_j dos códigos que pertenecen a un vector Y, $0<i<j<2^n-1$. y_i podrá ser asignado a C_k, con k número de unos, y y_j podrá ser asignado a C_{k+1} si la distancia mínima de Hamming (y_i, y_j) es ≥ 2.*

Algoritmo de asignación (*m* bits de información, *n* bits de código):

1. Si $2^n - 1 = m$, saltar al paso 4.
2. Si $2^n - 1 > m$, y *m* es impar. Entonces, se eliminan $2^n-(m+1)$ códigos consecutivos del vector Y teniendo en cuenta el cumplimiento de la *condición de asignación*. Saltar al paso 4.
3. Si $2^n - 1 > m$, y *m* es par, saltar al paso 7.
4. Siendo Y es vector seleccionado, sólo se necesitan *m* códigos; el código y_0 se asigna a $c_0^0 \in C_0$
5. y_1 se asigna a todas las combinaciones incluidas en C_1, y_2 se asigna a todas las combinaciones incluidas en C_2, y así sucesivamente tal que y_i se asigna a todas las combinaciones incluidas en C_i, $i = 1,...,m-1$.
6. El código y_m se asigna a $c_m^0 \in C_m$. Terminar.
7. Siendo Y el vector seleccionado, sólo se necesitan $m+1$ códigos; el código y_0 se asigna a $c_0^0 \in C_0$
8. Existe un C_k para $k=0,...,m$ donde el número de ceros en cualquier combinación c_k perteneciente a C_k es igual a número de unos. C_k es el mayor grupo de C y puede ser

dividido en dos subgrupos C_k' and C_k''. Las asignaciones se realizan cumpliendo el paso 5 y teniendo en cuenta la condición de asignación.

Al aumentar el número de grupos, necesitamos dos códigos para C_k' y C_k''. Siendo y_i, y_j dos códigos pertenecientes al vector Y, $0<i<j<2^n$-1, y_i se asigna a C_k' y y_j se asigna a C_k'', siempre que la distancia entre (y_i, y_j) sea igual a 1, tal que, siendo y_u el código asignado a C_{k-1} y y_v el código asignado a C_{k+1}, la distancia mínima entre cualquier par (y_u, y_i), (y_u, y_j), (y_v, y_i), (y_v, y_j), es ≥ 2. Existen 2^n-(m+2) códigos del vector Y que no se utilizan.

9. El código y_{m+1} se asigna a $c_m^0 \in C_m$. Terminar.

U_n cumple con el primer concepto básico deseado (A). Para alcanzar (B) definimos R_α

7.4.2.3. U_n y R_α: Detección de errores unidireccionales y aleatorios múltiples

Un código U_n es capaz de detectar cualquier error unidireccional que se produzca en una combinación vertical de bloque de datos. La carencia del código se observa con errores aleatorios. Si dichos errores son debidos a escrituras no efectivas o SEU, los bits invertidos serán adyacentes en el bloque de datos. Bajo este supuesto, se podría mejorar la cobertura de detección de U_n. Analizando las carencias de U_n según la localización de los bits invertidos, se observa:

Errores ocurridos sobre bits de información

Un código U_n no puede detectar la ocurrencia de inversiones aleatorias (no unidireccionales) adyacentes si:

A1. Se invierten dos bits de información o un número par.

A2. Dependiendo del vector utilizado, existen dos inversiones, una en los bits de información y otra en el código, siempre que dicho código ocupe la primera o última posición en la ordenación del vector.

Las inversiones aleatorias de número par de bits deben de mantener la paridad de m inalterada, sino sería un error unidireccional. Siguiendo las reglas de composición de U_n sabemos que si c_i y c_j pertenecen al mismo C_i, tienen asignado el mismo código. Entonces, la inversión aleatoria de dos bits que provoque el cambio de c_i a c_j no será detectada por U_n. Definimos un segundo código separable R_α para la detección de inversiones dobles aleatorias ocurridas en una ráfaga máxima de 3 bits: $R_\alpha = \{r_0, r_1, ..., r_{(2^\alpha-1)}\}$.

Regla básica: Cualquier combinación c_i de bits tiene una representación decimal d_i. Siendo d_i y d_j las representaciones decimales de c_i y c_j, ambas pertenecientes a C_i, si $|d_i - d_j| = 2^k$ o $(3\times2^{(k-2)})$, k = 0,...,m-1, los códigos r_α y r_β asignados a c_i y c_j deben ser diferentes. Así cubrimos los casos *A1* y *A2*. Por ejemplo:

	R_2	U_3	
110111	10	001	*Dec* = 55
011111	00	001	*Dec* = 31
101111	01	001	*Dec* = 47

Ejemplo: $|d_1 - d_2| = 3 \times 2^3$

Errores ocurridos sobre bits de información y sobre bits del código

Para seleccionar un código de detección apropiado, primero es necesario definir el orden en el que se van a añadir U_n y R_α a los bits de información.

En caso de establecer el orden: bits de informacion, R_α , U_n, se deberá tener en cuenta que:

A3. Ante la ocurrencia de tres inversiones aleatorias consecutivas, distribuidas de la siguiente manera:

$$b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\ r_1\ r_0\ y_2\ y_1\ y_0$$

Donde la combinación b_m mantiene su paridad inalterada, R_α y U_n podrían no detectar el error.

Para cubrir este caso, es necesario añadir una regla adicional a R_α:

E1. Siendo d_i y d_j la representación decimal de c_i y c_j ambas pertenecientes a C_i, si $|d_i - d_j| = 1$ los códigos r_α y r_β asignados a c_i y c_j deben tener una distancia mínima $(r_\alpha , r_\beta) \geq 2$. Por ejemplo:

	R_2	U_3	
111110	10	001	◄
111101	01	001	◄

Ejemplo: $|d_1 - d_2| = 1$

En caso de establecer el orden: bits de information, U_n, R_α, se deberá tener en cuenta que:

A4. Ante la ocurrencia de tres inversiones aleatorias consecutivas, distribuidas de la siguiente manera:

$$b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\ y_2\ y_1\ y_0\ r_1\ r_0$$

R_α y U_n podrían no detectar el error si y_n mantiene su paridad inalterada. Tampoco detectará el error si existe un error unidireccional sobre dos bits de información y un simple de valor contrario sobre U_n.

Para cubrir estos casos, la regla adicional cambia:

E2. Siendo d_i y d_j la representación decimal de c_i y c_j , c_i perteneciente a C_i y c_j perteneciente a C_j con $i \neq j$, y $|d_i - d_j| = 1$ o $|d_i - d_j| = 3$ los códigos r_α y r_β asignados a c_i y c_j deben ser diferentes. Por ejemplo:

	U_3	R_2		
10010	0 01	1	11	*Dec* = 36 ◄
10010	1 10	1	10	*Dec* = 37 ◄

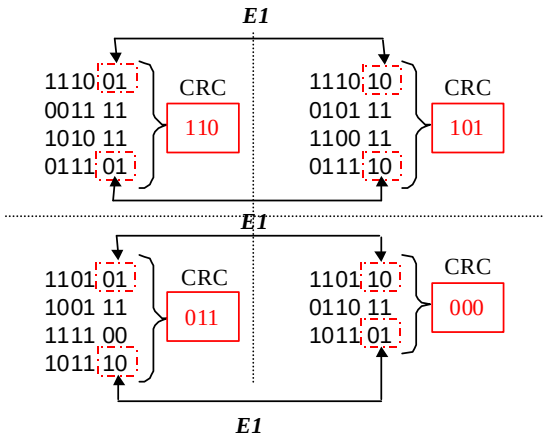
Ejemplo A4: $|d_1 - d_2| = 1$

0111	00 1	10	01	*Dec* = 28 ◄
0111	11 0	10	00	*Dec* = 31 ◄

Ejemplo A5: $|d_1 - d_2| = 3$

La mejora de U_n sobre el código Berger es que limita la falta de cobertura ante inversiones aleatorias dobles a combinaciones pertenecientes al mismo C_i. Para que una combinación $c_i \in Ci$ alcance otra combinación válida $c_j \in C_j$, siendo $i \neq j$, es necesaria la inversión de al menos tres bits de forma no unidireccional, lo que permite, por ejemplo, generar R_α utilizando sólo dos bits adicionales siguiendo la agrupación de un CRC con polimonio generador de grado 3 para cubrir cualquier ráfaga de errores de tres bits.

Por ejemplo, sea m = 6 y un polinomio generador $G(x)_3 = x^3 + 1$. Podemos dividir $C_4 \in C$ en cuatro grupos:

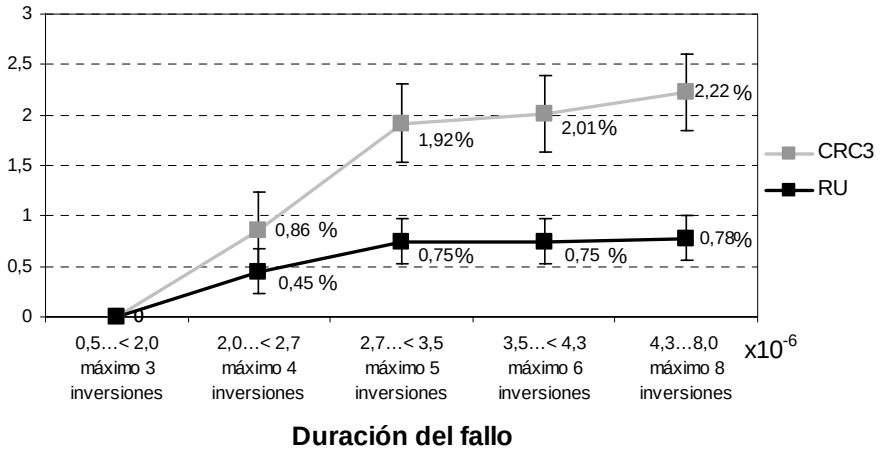


Donde todas las c_4 de cada grupo obtienen el mismo valor de CRC. Sin embargo, no es necesario utilizar los tres bits de este código. Suponiendo un orden: bits de informacion, R_α, U_n, será necesario seguir la regla básica de composición de R_α y *E1* que se cumple con un código r_α de dos bits, por ejemplo, los menos significativos del CRC calculado.

Los valores de la gráfica 7.5. representan una muestra de 3.000 experimentos en la que se observan:

Tipo de fallo	***Inyecciones efectivas***	***Nº de bloques de datos erróneos transmitidos***	***Nº total de combinaciones verticales erróneas***
Pegado simple a [1]	1.497	617	1.712
Pegado doble a [0,1]	1.575	1.352	4.707

Con la suma de R_α y U_n obtenemos una mejora sobre la cobertura reduciendo el número de errores no detectados aproximadamente a la mitad (gráfica 7.5.), no habiendose observado carencia de la cobertura a nivel de bloque. Los 1.969 bloques de datos que contenían algún error fueron detectados.



Gráfica 7.5. Comparación experimental entre un CRC con polinomio generador de grado 3 y los códigos R_2, U_3

Otra muestra es la recogida en la tabla 7.4. donde se desglosan los resultados según los porcentajes de errores detectados por cada código (R_2 o U_3) para fallos de diferente

duración. La muestra representa un total de 4.500 fallos efectivos, con 6.445 combinaciones verticales erróneas, de la cuales 55 no fueron detectadas por R_2 U_3. No se observó carencia de la cobertura a nivel de bloque.

	Detectado por:	0,5..<1,25µs	1,25..<2µs	2..<2,7µs	2,7..<3,5µs	3,5..<4,3µs	≥4,3µs
Pegado simple a [0]	U_3	100%	93,1%	86,2%	89,35%	81,85%	83,66%
	R_2	0%	6,9%	13,8%	10,65%	18,15%	16,34%
Pegado simple a [1]	U_3	100%	99,32%	94,85%	92,36%	91%	82,46%
	R_2	0%	0,68%	5,15%	7,64%	9%	17,54%
Pegado doble a [0, 1]	U_3	98,04%	82,31%	77,30%	82,56%	77,07%	82,16%
	R_2	1,96%	17,69%	22,7%	17,44%	22,93%	17,82%

Tabla 7.4. Distribución de las detecciones con R_2 o U_3 en el algoritmo de control del ABS

La mayoría de los errores pueden ser detectados por U_3, aunque no es suficiente. U_3 cubre todos los errores unidireccionales sobre los (m + n) bits, así como errores con cualquier número de inversiones aleatorias sobre los n bits de código. También cubre parte de los errores aleatorios sobre m bits de información, como por ejemplo, aquellos con un número impar de inversiones. La importancia de R_2 se aprecia mejor con los pegados a 0 y fallos dobles, donde el porcentaje de errores aleatorios provocados es mayor al de unidireccionales, sobretodo en la detección de errores de número par, como es el caso de los intervalos [2µs..< 2,7µs] y [3,5µs…< 4,3µs].

7.4.2.4. Alternancia entre diferentes códigos de detección

La gran desventaja de los códigos verticales es la sobrecarga que suponen. Cada columna de la tabla 7.5. se encabeza con el número de palabras de información (sin código) a transmitir e indica el tanto por ciento de aumento que supondría la inclusión de un código determinado en el bloque de datos[116].

	4 palabras	***8 palabras***	***16 palabras***	***32 palabras***	***64 palabras***	***128 palabras***
U_n	75%	50%	31,25%	18,75%	10,94%	6,25%
R_2 U_n	125%	75%	43,75%	25%	14,06%	7,81%
$CRC_{3\ vertical}$	75%	37,5%	18,75%	9,38%	4,69%	2,34%
CRC_{16}	25%	12,5%	6,25%	3,13%	1,56%	0,78%

Tabla 7.5. Comparación de la sobrecarga entre 4 códigos de detección de errores

El CRC_{16} siempre va a obtener una sobrecarga inferior que cualquier código vertical. Sin embargo, un *CDEV* como el R_2U_n siempre será más efectivo ante fallos múltiples o casos como las escrituras no efectivas.

Se podría pensar en combinar la alta cobertura teórica que presenta un código calculado sobre el bloque completo de datos con la efectividad de un CDEV ante fallos múltiples o escrituras no efectivas. Por ejemplo, en bloques de datos pequeños, donde la sobrecarga de un CDEV es muy elevada se puede recurrir a un CRC_{16} junto con una

[116] Se han tomado los porcentajes más pesimistas en el caso de U_n y R_2 U_n

paridad bit por columna (32 bits de código). En este caso, los errores no detectados se reducirían a aquellos patrones de error no detectados por el CRC_{16} y donde además, el número de inversiones por combinación vertical sea par. Experimentalmente, mediante simulación, se observa que existe un patrón de error no detectado para inversiones localizadas en *m* palabras adyacentes por cada combinación de una de las *m* palabras. En este sentido es posible comparar un CRC_3 vertical y un CRC_{16} con paridad por columna. El CRC_3 vertical cubre cualquier error múltiple con inversiones localizadas en tres palabras adyacentes, mientras que un CRC_{16} con paridad por columna presenta algunas carencias. Si cada palabra es de 16 bits, existe un total de $(2^{16}-1)$ patrones no detectados. En general, el número de patrones no detectados para bloques de *n* bits es de $2^{n-32}-1$ (un código cíclico redundante de 32 bits). En caso de errores unidireccionales, cuando las inversiones se localicen en dos combinaciones verticales, se cubren todos los errores hasta bloques de seis palabras (inclusive). La tabla 7.6. muestra algunos ejemplos que indican la reducción en el número de patrones de error no detectados con bloques de mayor tamaño e inversiones en dos combinaciones verticales.

Tamaño del bloque

	7 palabras	***8 palabras***	***9 palabras***	***10 palabras***	***11 palabras***	***12 palabras***
No detectados CRC_{16} + paridad	3	24	115	456	1.840	7.528
No detectados CRC_{16}	24	115	456	1.840	7.528	30.477

Tabla 7.6. Reducción del número de patrones no detectados con el CRC_{16}

Sin embargo, a medida que aumenta el tamaño del bloque, la localización de bits invertidos a causa de fallos múltiples puede presentar formas muy variadas, siendo más eficaz proteger los datos con códigos independientes para cada columna del bloque. Aunque también se podría utilizar códigos independientes por fila, no resolverían el problema de las escrituras (o lecturas) no efectivas ni el del acceso a direcciones erróneas, donde la ráfaga del error aleatorio sería del mismo tamaño que el ancho del bus de datos. Las escrituras no efectivas se originan con fallos transitorios de pequeña duración sobre las líneas del bus de datos. Entonces, resulta más interesante un código vertical, ya que aunque el número de palabras afectadas depende de la duración del fallo, en el caso por ejemplo del TTP/C, sería necesario un fallo de 12μs para causar la escritura no efectiva de dieciséis palabras. Para ser un fallo transitorio sobre la lógica combinacional debido a la radiación es demasiado alto, por tanto sólo podría tratarse de un fallo de pegado en una línea del bus. Si el fallo se prolonga en el tiempo, o bien se trata de un error unidireccional o el controlador de memoria parará la escritura generando una excepción.

El equilibrio entre sobrecarga y efectividad puede mejorarse con la alternancia entre diferentes códigos de detección. En función del impacto que supone el tratamiento de datos incorrectos en el algoritmo de control será necesaria una protección más o menos estricta. Cuanto más críticos sean los datos transmitidos, menor es la importancia que adquiere la sobrecarga y mayor la del código asignado. Además, teniendo en cuenta que un protocolo como TTP es capaz de transmitir hasta 240 bytes por canal en cada ventana de tiempo, el tamaño máximo de la trama podría incluir un código vertical si fuera necesario. En un protocolo de disparo por tiempo, la restricción en el tamaño de la trama no depende de un solo nodo, sino del periodo de tiempo máximo asignable al TDMA admitido por el algoritmo de control, en función del cual existe un número máximo de bytes que se pueden enviar por TDMA.

7.5. Resumen y Conclusiones del Capítulo

Este capítulo se centra en la inyección de fallos a nivel de pin llevada a cabo sobre la barrera de contención de errores que existe entre el nivel de aplicación y el controlador de comunicaciones, la interfaz de comunicaciones o CNI. Los experimentos realizados abordan dos medios para mejorar la confiabilidad del sistema: la eliminación de fallos y la predicción de fallos. Como parte de la eliminación de fallos se encuentra la detección de errores de diseño e implementación, errores que no suelen detectarse hasta que ante un evento determinado el servicio ofrecido no coincide con el especificado. Este es el caso del error de implementación en el algoritmo de vida observado durante los experimentos. El algoritmo de vida es un servicio de interconexión entre el controlador de comunicaciones y el nivel de aplicación. Este algoritmo mantiene el diálogo entre el nivel de aplicación y el nivel de enlace y su interrupción indica que alguno de los dos niveles ha detectado un error que afecta a su función. Las especificaciones del protocolo definen las acciones a seguir en caso de que este algoritmo se interrumpa. Sin embargo, estas acciones no han sido correctamente implementadas en el TTP^{TM}/C. Los experimentos que han permitido observar el error en el prototipo han podido ser reproducidos en el modelo en VHDL con la herramienta de inyección VFIT. Además de detectar un error de implementación, la reproducción de experimentos con técnicas diferentes permite validar el resultado y a su vez validar las propias herramientas de inyección.

En una arquitectura como TTA, la distribución de los algoritmos de control hace que desde que se calcula el valor de una variable del algoritmo de control en un nodo hasta que este nuevo valor es utilizado en el nivel de aplicación de otro nodo, transcurra un tiempo durante el cual pueden aparecer en el sistema uno o varios fallos físicos que provoquen errores en los bits de datos. La secuencia de etapas por las que pasan los datos comprenden desde el cálculo de las variables en un nodo origen hasta que dichas variables se encuentran disponibles en el nodo destino, pasando por la escritura de los datos en la CNI, el almacenamiento en memoria, la lectura de los datos por el controlador, la construcción de la trama y la transferencia de la trama por el canal de comunicaciones. En el camino inverso se realiza la comprobación de la trama recibida, la escritura de los datos en la CNI, su almacenamiento y la lectura de los datos desde la aplicación.

En esta secuencia existen dos etapas donde los datos son especialmente sensibles a fallos ya que no se encuentran protegidos mediante ningún mecanismo de detección de errores, son la escritura (o lectura) de los datos en la CNI y su almacenamiento en memoria. En este capítulo se demuestra la vulnerabilidad de los datos ante fallos físicos a nivel de pin, motivando el análisis de varios códigos de detección de errores para mejorar la cobertura del sistema.

Los errores en datos causados por fallos físicos se pueden agrupar en errores unidireccionales o errores aleatorios múltiples, ambos causados por fallos simples, aunque también pueden ocurrir errores derivados de fallos múltiples no necesariamente conexos o coincidentes en su instante de aparición. Los fallos físicos a nivel de pin se inyectan para eliminar, variar o generar pulsos en las señales correspondientes a los accesos de escritura o lectura de datos en memoria. Estos fallos causan, entre otros, errores unidireccionales y errores aleatorios múltiples en los valores que forman parte del mensaje. Por tanto, los errores se propagan con los mensajes a otros nodos que no tienen capacidad para decidir si la información recibida es correcta. Estas transmisiones se consideran averías en un sistema distribuido donde se asume que sólo se transmite si se pueden asegurar datos libres de errores.

El protocolo TTP utiliza un esquema de tiempos fijo, no sólo para la transmisión de las tramas, sino también para la ejecución de tareas a nivel de aplicación. Utilizar una carga sintética como algoritmo de control que ejercite un esquema de tiempos básico ha resultado de gran utilidad a la hora de clasificar los errores derivados de un tipo de fallos y el comportamiento del controlador. Sin embargo, a la hora de obtener la cobertura experimental de un determinado código de detección de errores se puede llevar a cabo una estimación más ajustada utilizando una carga típica para el sistema a validar, como es el algoritmo de frenado por cable para el control activo del ABS.

El número de bits afectados en un error unidireccional o aleatorio depende de la duración del fallo. En un bus paralelo como es el bus de datos de la interfaz de la CNI, se podría aplicar un código de paridad bit por línea del bus. Este código ofrece una buena cobertura ante fallos transitorios cuya duración no exceda el tiempo que conlleva un acceso simple a memoria. Sin embargo, aún con fallos transitorios inferiores a este tiempo es posible provocar errores múltiples en memoria, justificando el análisis de diferentes alternativas, tales como el código de detección de errores cíclico redundante, códigos separables verticales o la alternancia entre varios códigos de detección.

El primer código analizado es un código cíclico redundante o CRC calculado a partir de un polinomio generador de grado 16 con una distancia mínima de Hamming de 5 para bloques de datos de 257 bits o inferiores. Teniendo en cuenta que los algoritmos de control de guiado por cable "*x-by-wire*" utilizan bloques de datos más cercanos a diez bytes que a cien bytes, la cobertura que ofrece este CRC_{16} sería muy alta ante errores múltiples derivados de fallos simples, salvo los derivados de escrituras no efectivas, donde la posición direccionada no se actualiza con el nuevo valor, o los derivados de accesos a direcciones erróneas.

Al decir que un fallo causa varias escrituras no efectivas, dichas escrituras corresponden a accesos a memoria consecutivos, provocando la inversión de bits adyacentes en las transmisiones de una línea del bus de datos, error que podría ser fácilmente detectado por un código de detección de errores aplicado por cada línea del bus paralelo. Por tanto, se analizan también posibles códigos separables tratados como códigos verticales haciendo referencia con cada columna a una línea del bus de datos. El primer código analizado es un CRC calculado a partir de un polinomio generador de grado 3. La cobertura experimental obtenida en el algoritmo control activo del ABS es superior a la cobertura teórica calculada. Para comparar ambas coberturas se analiza individualmente el resultado de cada código vertical transmitido. Sin embargo, analizando la carencia de la cobertura a nivel de bloque, o lo que es lo mismo, que ninguno de los 16 códigos de detección enviados en cada transmisión detecte alguno de los errores contenidos en los datos, existen transmisiones erróneas que se aceptan como correctas.

Es posible reducir el número de patrones de error no detectados si tenemos en cuenta la paridad de la secuencia de bits sobre la que se aplica el código. En este capítulo se propone una variación del código Berger al que se le ha llamado U_n. Se describe tanto sus reglas de composición como su algoritmo de asignación. Aunque el objetivo principal de este tipo de códigos sea la detección de errores unidireccionales de cualquier ráfaga, la mejora de U_n sobre el código Berger es que limita la falta de cobertura a inversiones aleatorias pares en combinaciones de m bits de información con el mismo número de unos. Por tanto, aunque existe cierta carencia de U_n ante errores aleatorios, si dichos errores son debidos a escrituras no efectivas o SEU, los bits invertidos serán adyacentes, siendo posible aplicar reglas de detección similares a las de un CRC. El código resultante, denominado U_n R_α, será capaz de detectar cualquier error unidireccional y errores aleatorios múltiples de ráfaga limitada. Por ejemplo, en este capítulo se utiliza R_2, con 2

bits de código, capaz de detectar errores aleatorios en ráfagas máximas de tres bits. Para el resto de los errores aleatorios, la cobertura del código se ha probado experimentalmente obteniendo un porcentaje superior al 99% de detecciones por número de códigos totales transmitidos, mientras que no se ha registrado ninguna carencia a nivel de bloque.

La gran desventaja de los códigos verticales es la sobrecarga que suponen. El equilibrio entre sobrecarga y efectividad puede mejorarse con la alternancia entre diferentes códigos de detección. En función del impacto que supone el tratamiento de datos incorrectos en el algoritmo de control será necesaria una protección más o menos estricta. Cuanto más críticos sean los datos transmitidos, menor es la importancia que adquiere la sobrecarga y mayor la del código asignado. Además, teniendo en cuenta que un protocolo como TTP es capaz de transmitir hasta 240 bytes por canal en cada ventana de tiempo, el tamaño máximo de la trama podría incluir un código vertical si fuera necesario. En un protocolo de disparo por tiempo, la restricción en el tamaño de la trama no depende de un solo nodo, sino del periodo de tiempo máximo asignable al TDMA admitido por el algoritmo de control, en función del cual existe un número máximo de bytes que se pueden enviar por TDMA.

En general, los códigos de detección de errores se pueden aplicar a cualquier arquitectura distribuida en cuya confiabilidad se tenga en cuenta tanto la recepción de mensajes relativa al tiempo como si el mensaje recibido contiene datos correctos y por tanto útiles.

Capítulo 8. Conclusiones y Trabajo Futuro

La mayor parte de este trabajo de tesis se basa en resultados obtenidos durante el proyecto de investigación FIT ("Fault Injection for TTA") con financiación europea IST-1999-10748 bajo el quinto programa marco. Este proyecto de dos años de duración, desde mayo de 2000 hasta mayo de 2002, está centrado la evaluación de la confiabilidad de la arquitectura TTA y en la validación experimental mediante Inyección de Fallos de una implementación del protocolo de comunicaciones TTP basado en TTA, el controlador de comunicaciones TTP^{TM}/C. Entre las técnicas de inyección existentes se eligen seis: dos basadas en simulación de modelos, dos técnicas implementadas por software y dos técnicas hardware de inyección física.

Las técnicas de inyección basadas en simulación de modelos corresponden a Inyección de fallos en modelos VHDL, llevada a cabo por el Grupo de Sistemas Tolerantes a Fallos de la Universidad Politécnica de Valencia, y a Inyección de fallos en modelos desarrollados a partir de las especificaciones mediante lenguajes en alto nivel, llevada a cabo por la Universidad Técnica de Praga en colaboración con la Universidad Técnica de Pilsen-Bohemia.

La validación experimental con técnicas de Inyección de fallos implementada por software se plantea con dos objetivos. Primero, verificar el comportamiento del sistema ante errores permanentes y segundo, verificar y evaluar su comportamiento ante errores transitorios localizados en zonas accesibles por software. Los errores permanentes se inyectan con una herramienta "pre-runtime" (o de introducción de errores previa a la ejecución de la carga) diseñada por el Instituto Técnico de Carinthia, Austria, mientras que la inyección de errores transitorios se lleva a cabo por el Grupo de Sistemas de Tiempo Real de la Universidad Técnica de Viena.

Las dos técnicas que se consideraron más representativas dentro de la Inyección física de fallos son la radiación de partículas, llevada a cabo por la Universidad de Tecnología de Chalmers, Suecia, y la Inyección física de fallos a nivel de pin, llevada a cabo por el Grupo de Sistemas Tolerantes a Fallos de la Universidad Politécnica de Valencia, y cuyos resultados son la base del presente trabajo de tesis.

Esta colaboración entre grupos de investigación pertenecientes a diferentes universidades europeas representa un esfuerzo de colaboración, apoyado por empresas como VOLVO, desde su sede en Göteborg, Motorola en Munich y TTTech en Viena, que tiene como intención principal unificar conocimientos, crear las bases de la colaboración internacional dentro del marco europeo y hacer extensible la investigación realizada dentro y fuera de Europa mediante la divulgación de los resultados obtenidos durante el proyecto.

En este capítulo se resumen las conclusiones más importantes extraídas del desarrollo del trabajo de tesis doctoral presentado, así como las publicaciones derivadas de él. Finalmente, se ha dedicado una parte del capítulo para exponer las líneas de trabajo abiertas en futuras investigaciones.

8.1. Conclusiones

Tras la introducción de los conceptos y términos generales relacionados con el campo de la confiabilidad de los sistemas informáticos, este trabajo se orienta hacia una clase concreta de sistemas, los sistemas distribuidos. La necesidad de evaluar la confiabilidad de estos sistemas y de verificar su correcto funcionamiento ante cualquier situación hacen necesarias técnicas de validación experimental eficaces para observar y evaluar su comportamiento tanto en condiciones normales como en situaciones especiales provocadas por la aparición de errores o averías.

El efecto de un fallo en un sistema distribuido puede ser un error o avería con una doble consecuencia. Primero, relativa al comportamiento de la parte afectada directamente por el fallo, y segundo, cómo afecta ese comportamiento al resto de los miembros que forman el sistema y que comparten el canal de comunicaciones. En definitiva, cómo afecta al servicio que el sistema ofrece al usuario. En este sentido, la Inyección de Fallos ha demostrado su capacidad a la hora de validar experimentalmente el comportamiento de un sistema en presencia de fallos, evaluando la confiabilidad del producto antes de que éste llegue al mercado.

8.1.2. Técnicas de inyección de fallos

Uno de los objetivos cumplidos en el trabajo de tesis ha sido avanzar en el análisis del alcance e impacto de los fallos físicos. Se ha estudiado la bibliografía más importante, especialmente aquella que analiza la influencia del avance tecnológico en la reducción de geometrías de diseño, aumento de la frecuencia de funcionamiento y densidad de integración en la extensión y multiplicidad de los errores derivados de fallos físicos. En [DBench ETIE2 2002] se expone la relación entre las causas que favorecen la aparición de fallos físicos y los modelos de fallos representativos de dichas causas a nivel lógico y RT en la validación experimental mediante Inyección de Fallos. En este trabajo se revisan los efectos de la radiación en los semiconductores y la lógica combinacional, los efectos debidos al deteriodo, desgaste o a las propias condiciones ambientales tanto a nivel de transistor como en metalizaciones internas y soldaduras. Se completan, por tanto, los modelos definidos en los niveles lógico y RT con modelos aplicables a metalizaciones y soldaduras.

También cabe destacar el aporte de una estructuración actualizada al concepto de Inyección de Fallos a través de la agrupación de los parámetros relacionados con el proceso de validación. Dichos parámetros permiten entender mejor cuáles son las diferencias entre las distintas técnicas de inyección de fallos. El trabajo propone una lista de propiedades descriptivas aplicables a cualquier técnica o herramienta de inyección. Además, tras la revisión del estado del arte, se ha tratado de valorar estas propiedades sobre las técnicas más importantes publicadas hasta el momento como una posible comparación orientativa.

8.1.3. La arquitectura TTA ("Time-Triggered Architecture")

La forma de garantizar la fiabilidad y seguridad de un sistema electrónico, como los sistemas guiados por cable "*x-by-wire*", es desarrollarlo bajo una arquitectura que demuestre una alta confiabilidad. La arquitectura que hasta el momento ha demostrado unos niveles de seguridad más altos es la arquitectura TTA ("Time-Triggered Architecture") o *arquitectura de disparo por tiempo*.

En la actualidad, el diseño de sistemas distribuidos tolerantes a fallos que den soporten a aplicaciones críticas es un reto para la investigación y el desarrollo de arquitecturas fiables y seguras que proporcionen el grado de confiabilidad requerido en productos cuyo correcto funcionamiento es esencial para evitar riesgos humanos o económicos. Entre estos productos, y de pronta aparición en el mercado, se encuentra el controlador de comunicaciones TTP^{TM}/C, basado en TTA. Hereda, por tanto, las condiciones óptimas que la arquitectura garantiza sobre comportamiento seguro. Sin embargo, la aserción teórica de un comportamiento libre de inconsistencias no es suficiente ante la importancia del uso destinado para el producto. De ahí parte la iniciativa para validar experimental una implementación del protocolo de comunicaciones TTP, el controlador de comunicaciones TTP^{TM}/C, con diferentes técnicas de Inyección de Fallos durante el proyecto FIT, entre ellas, la Inyección física a nivel de pin.

Los fallos físicos a nivel de pin permiten ocasionar, de forma controlada, varios tipos de alteraciones sobre las señales de entrada y salida del prototipo, y más concretamente, sobre las líneas vinculadas a las barreras de contención de errores del controlador de comunicaciones. Estas barreras de contención constituyen una parte fundamental en la arquitectura TTA para garantizar su fiabilidad. El interés principal en la inyección física de fallos se centra en la representatividad de los fallos que inyecta. Así, la Inyección física a nivel de pin permite observar el comportamiento del prototipo ante fallos críticos en un sistema distribuido, como son los fallos en el canal de comunicaciones, difícilmente provocados por otras técnicas, o fallos que alteran el contenido de los mensajes causando errores múltiples no detectables por el receptor del mensaje.

En general, las técnicas de inyección física de fallos son más apropiadas para inyectar fallos en el canal de comunicaciones sobre el prototipo que las técnicas basadas en SWIFI. Además, la inyección de fallos a nivel de pin es una técnica que dota al entorno de validación de un control no disponible con otras técnicas de inyección física. Por ejemplo, en los experimentos llevados a cabo con radiación de partículas α se observan comportamientos similares a los observados con SWIFI o con HWIFI. Sin embargo, la imprecisión a la hora de analizar el origen de cada comportamiento observado hace que no sea posible afirmar que un experimento determinado haya sido reproducido con ambas técnicas. Los mejores logros en la reproducción de experimentos se han alcanzado entre Inyección física a nivel de pin e Inyección de fallos en el modelo VHDL. Ambas técnicas pueden llegar a ser complementarias en el análisis de escenarios críticos.

8.1.4. Inyección de fallos en sistemas distribuidos

En el trabajo se han descrito las ampliaciones y modificaciones realizadas sobre la herramienta de inyección física de fallos a nivel de pin AFIT ("Advanced Fault Injection Tool"), para su utilización con sistemas distribuidos. Esta herramienta genérica es el medio que permitirá inyectar fallos físicos en prototipos de diferentes sistemas como parte de su validación experimental.

Existen dos objetivos previos a la validación del protocolo. Primero, se ha avanzado en la Inyección física de fallos en sistemas distribuidos. La aportación más importante es la adaptación realizada en AFIT para sistemas distribuidos. Se trata de una nueva herramienta, también genérica, diseñada para resolver el proceso de adquisición de datos con herramientas de inyección que trabajen sobre un prototipo del sistema o el sistema real. La herramienta, diseñada y construida durante el proyecto de investigación FIT, combina un monitor hardware con tareas software implementadas según la arquitectura del sistema a observar. La sincronización entre AFIT y el monitor diseñado es de gran importancia para la adquisición de tiempos de latencia. Esta sincronización se consigue mediante la utilización de una única interfaz que supervisa la secuencia de control ejecutada en ambas herramientas. Además, se han reducido los retardos de propagación de las señales internas de control de la inyección. También, se ha capacitado a la herramienta para inyectar fallos múltiples, disponiendo de sondas de conexión directa o a través de zócalos de gran utilidad para la inyección sobre buses, dotándolas de nuevos terminadores que evitan el riesgo de perturbaciones no deseadas en el dispositivo bajo prueba.

Segundo, respecto a las inyecciones sobre las barreras de contención del controlador de comunicaciones TTP/C, se analiza la relación entre el conjunto de fallos inyectables y el impacto que se desea obtener en el sistema. Define, por tanto, el dominio de entrada de la validación experimental en el controlador ante fallos a nivel de pin. El conjunto de fallos incluye el detalle de su localización, modelo y persistencia, mientras que el impacto alude a los modos de avería que observaremos y ante los cuales el protocolo de comunicaciones deberá mantener un estado consistente. En general, se presenta la base para configurar el entorno de validación.

8.1.5. Validación del protocolo de comunicaciones

En sistemas distribuidos tolerantes a fallos, el protocolo de comunicaciones debería asegurar un comportamiento confiable del sistema, no sólo ante errores internos ocurridos en un nodo, sino también ante fallos en el canal de comunicaciones o ante acciones de desconexión y reintegración sin que conlleven una pérdida del servicio y sin que se incumplan los requisitos establecidos para albergar el tipo de aplicaciones a las que se orienta el sistema. En este sentido, los experimentos realizados han permitido, por un lado, verificar si la implementación del prototipo cumple con las especificaciones de diseño, y por otro lado, validar la confiabilidad del protocolo ante los tipos de avería más frecuentes que se pueden observar en un sistema distribuido, como son la parada de un nodo, las transmisiones espurias sintáctica o semánticamente incorrectas, las denominadas averías SOS y la pérdida de conexión.

Los experimentos relacionados con la averías con parada y reintegración se basan en la variación controlada de los márgenes temporales de las señales de control que tienen como consecuencia la interrupción del volcado de la trama o la habilitación de la transmisión fuera del periodo de tiempo establecido, caso que favorece la aparición de fallos espurios coincidentes con la transmisión de otro nodo. La perturbación, tanto en un canal como en ambos simultáneamente con fallos conexos ha permitido verificar la implementación del reconocimiento implícito de las tramas definido en el protocolo y analizar sus carencias cubiertas por el servicio de pertenencia. También, se han propuesto algunas consideraciones no especificadas en el protocolo relativas a cómo debe definirse la planificación de los mensajes para permitir la reintegración simultánea de varios nodos en el sistema.

La validación del TTP ante transmisiones espurias se ha llevado a cabo mediante la inyección de fallos coincidentes con la transmisión de alguna trama, causando la alteración de la misma, y con fallos no coincidentes. La división de los experimentos según el instante de la inyección ha facilitado el análisis de los resultados, parte de los cuales se ha podido reproducir en el modelo en VHDL para un mejor análisis. Se ha observado que la sincronización de la base de tiempos común es crítica en el TTP, y extensible a cualquier protocolo de disparo por tiempo, ya que a mayor desviación de los relojes locales mayor probabilidad de que una transmisión espuria afecte al correcto funcionamiento del sistema al convertirse en una avería SOS, tipo de averías asimétricas con grave incidencia en la arquitectura TTA. También existe una fuerte dependencia entre el guardián del bus y los componentes del nivel físico utilizados en el diseño. Componentes de los que depende para contener eficazmente transmisiones espurias y evitar así la propagación de errores.

Se revela como punto débil del protocolo que no existe limitación en la actualización de sus variables por ventana de tiempo, lo que favorece situaciones de inconsistencia ante pulsos espurios acaecidos entre la apertura de la ventana de tiempo y la recepción de una trama. Otro escenario crítico es el que plantea una transmisión espuria detectada antes de la recepción real de la trama e interpretada como parte de la misma. Este escenario, especialmente si se tiene en cuenta la desviación de los relojes locales que retarda o adelanta la apertura de unas ventanas de recepción con respecto a otras, fuerza una situación de inconsistencia no deseable. En general, son averías SOS que provocan la pérdida de consenso entre los miembros activos del sistema. Esta pérdida de consenso es la que trata de evitar el TTP utilizando estrategias como la NGU donde el subgrupo mayoritario impone su visión del sistema con la intención de mantener la continuidad del servicio.

Además de comprobar experimentalmente que no siempre se cumple este objetivo, ya bien sea por carencias del protocolo o por situaciones de empate, en una reflexión sobre esta estrategia se apuntan dos hechos. Primero, que no se asegura que el nodo origen del error asuma su condición de erróneo. Segundo, que tampoco se asegura que habiendo dos nodos replicados sólo uno de ellos pase a estado pasivo permaneciendo su réplica activa. Por tanto, la diferencia entre aplicar una estrategia como la NGU o no aplicar nada es la posibilidad de especificar el tiempo máximo de restablecimiento del sistema completo.

Finalmente, se ha analizado experimentalmente el comportamiento del protocolo ante la pérdida de conexión de uno o más nodos. La inyección física a nivel de pin posibilita la validación del protocolo ante este tipo de averías emulando cortes transitorios de distinta duración sin causar daño alguno al sistema. Los experimentos realizados diferencian entre una topología de bus con canal sin replicar y con canal replicado verificando la correcta implementación del servicio de pertenencia y del reconocimiento implícito propuesto en TTP.

8.1.6. Mejoras en la cobertura de detección de errores

Entre los trabajos de investigación publicados sobre Inyección de Fallos, se encuentran algunos estudios comparativos entre herramientas basadas en la misma técnica o basadas en técnicas distintas. Hasta ahora, la conclusión de estos trabajos es similar, las herramientas son complementarias, ya que difícilmente reproducen los mismos experimentos. Sin embargo, es posible llevar a cabo un análisis del sistema más preciso si

se coordinan dos técnicas para reproducir los mismos experimentos. Éste es el caso de la *eliminación de fallos*, uno de los medios orientados a mejorar la confiabilidad. El desarrollo dentro del Grupo de Sistemas Tolerantes a Fallos de dos herramientas basadas en técnicas distintas, como son la Inyección en modelos VHDL y la Inyección física a nivel de pin, ha permitido coordinar los experimentos orientados a la detección y análisis de errores de diseño e implementación, errores que no suelen detectarse hasta que ante un evento determinado el servicio ofrecido no coincide con el especificado. Las causas que originan un comportamiento determinado en el prototipo ante un error pueden ser analizadas con gran nivel de detalle al reproducir los experimentos en el modelo en VHDL. Además de detectar errores de diseño e implementación, la reproducción de experimentos con técnicas diferentes permite validar los resultados y a su vez validar las propias herramientas de inyección.

En una arquitectura como TTA, la distribución de los algoritmos de control hace que desde que se calcula el valor de una variable del algoritmo de control en un nodo hasta que este nuevo valor es utilizado en el nivel de aplicación de otro nodo, transcurra un tiempo durante el cual pueden aparecer en el sistema uno o varios fallos físicos que provoquen errores en los bits de datos. Existen dos etapas donde los datos son especialmente sensibles a fallos ya que no se encuentran protegidos mediante ningún mecanismo de detección de errores, son la escritura (o lectura) de los datos en la CNI y su almacenamiento en memoria. En este trabajo se ha demostrado la vulnerabilidad de los datos ante fallos físicos a nivel de pin, motivando el análisis de varios códigos de detección de errores para mejorar la cobertura del sistema.

Se ha observado que aún con fallos transitorios cortos es posible provocar errores múltiples en memoria, justificando el análisis de diferentes alternativas, tales como el código de detección de errores cíclico redundante, códigos separables verticales o la alternancia entre varios códigos de detección. Los códigos verticales presentan experimentalmente una cobertura de detección muy alta, siendo especialmente eficaces ante fallos múltiples o errores aleatorios que afecten a más de una palabra del mensaje. Sin embargo, la gran desventaja de los códigos verticales es la sobrecarga que suponen. El equilibrio entre sobrecarga y efectividad puede mejorarse con la alternancia entre diferentes códigos de detección. En función del impacto que supone el tratamiento de datos incorrectos en el algoritmo de control será necesaria una protección más o menos estricta. Cuanto más críticos sean los datos transmitidos, menor es la importancia que adquiere la sobrecarga y mayor la del código asignado.

En general, los códigos de detección de errores analizados se pueden aplicar a cualquier arquitectura distribuida en cuya confiabilidad se tenga en cuenta tanto la recepción de mensajes relativa al tiempo como si el mensaje recibido contiene datos correctos y por tanto útiles.

8.2. Resultados de la Investigación

Los resultados más importantes de la investigación llevada a cabo han sido publicados en varios congresos internacionales de alto impacto. También se han publicado artículos relacionados con la herramienta AFIT y artículos de difusión sobre la técnica de inyección, incluyendo el capítulo de un libro sobre técnicas de inyección de fallos de la editorial Kluwer. Además, existe numerosa información publicada en informes internos durante el proyecto FIT (“Fault Injection for TTA”), disponibles en www.cti.ac.at/fit

8.2.1. Publicaciones relacionadas con la investigación

- S. Blanc, J.C. Campelo, P.J. Gil, J.J. Serrano, "Stratified Fault Injection using Hardware and Software-implemented Tools", Proceeding of 4th IEEE International Workshop on Desing and Diagnostics of Electronic Circuits and Systems (DDECS'01), pp. 259-266, Györ, Hungría, Abril 2001.
- S. Blanc, P.J. Gil, A. Ademaj, H. Sivencrona, J. Torin, "Three Different Fault Injection Techniques Combined to Improve the Detection Efficiency for Time-Triggered Systems", Proceeding of 5th IEEE International Workshop on Desing and Diagnostics of Electronic Circuits and Systems (DDECS'02), pp. 412-415, Bnro, República Checa, Abril 2002.
- S. Blanc, J. Gracia, P.J. Gil, "A Fault Hypothesis Study on the TTP/C using VHDL-based and Pin-Level Fault Injection Techniques", Proceeding of IEEE 17th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2002), pp. 254-262, Vancouver, Canada, Octubre 2002.
- S. Blanc, P.J. Gil, "Improving the Multiple Error Detection Coverage in Distributed Embedded Systems", Proceedings of 22nd International Symposium on Reliable Distributed Systems (SRDS2003), pp. 303-312, Florencia, Italia, Septiembre 2003.
- S. Blanc, J. Gracia, P.J. Gil, "Experiences during the Experimental Validation of the Time-Triggered Architecture", Proceeding of the Design, Automation and Test in Europe Conference (DATE2004), pp. 30. 256 – 262, París, Francia, Febrero 2004.

8.2.2. Otras publicaciones relacionadas con Inyección de Fallos

- P.J. Gil, S. Blanc, J.J. Serrano, "Pin-level Hardware Fault Injection Techniques", capítulo 2.1 del libro "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation", A. Benso y P. Prinetto *ed*. Book series: Frontiers in Electronic Testing, vol. 23, pp.63-80 , 2003 Kluwer Academic Press.
- J. Gracia, S. Blanc, J.C. Baraza, D. Gil, P.J. Gil, "VFIT: Una herramienta Automática para la Inyección de Fallos en VHDL", Seminario Anual de Automática, Electrónica Industrial e Instrumentación (SAAEI 2002), pp 289-292, 2002.
- R. Ors, S. Blanc, J.C. Campelo, J.J. Serrano, "Garantía de Funcionamiento en Sistemas Tolerantes a Fallos", 3ª Jornada Técnica Anual sobre Fiabilidad y Calidad, Valencia 14 Noviembre 2001, ed. SP-UPV, pp. 29-68.
- S. Blanc, "Diseño de la Etapas de Temporización y Potencia de un Inyector de Fallos de alta velocidad", Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, 1998.
- S. Blanc, "Comparación de la Inyección Física de Fallos con Inyección de Fallos implementada por Software para Sistemas Digitales Comerciales de Alta Velocidad", Trabajo de investigación tutelado, Universidad Politécnica de Valencia, 2000.

8.2.3. Publicaciones relacionadas con la divulgación del proyecto FIT

- J. Gracia, S. Blanc, P.J. Gil, J.C. Campelo, "Fault Injection for the Time-Triggered Architecture", Proceedings of Dependable Systems and Networks 2001, pp. 25-27, Julio 2001, Göteborg, Suecia.

- S. Blanc, J. Gracia, P.J. Gil, J.V. Busquets, J.J. Serrano, “DBench”, Proceedings of Dependable Systems and Networks 2001, pp. 25-27, Julio 2001, Göteborg, Suecia.
- S. Blanc *et al*. FIT Project IST 1999-10748, Deliverable 1.2, 2.4, 3.3, 4, 5.3 and Fault Injection Techniques. Presentados y aceptados por la Information Societies Technology (IST) y disponibles en www.cti.ac.at/fit.

8.3. Trabajo Futuro

Existen principalmente dos líneas de investigación que continúan el trabajo realizado hasta el momento. Por un lado, la validación de sistemas distribuidos basados en la arquitectura TTA y por otro lado la mejora de las técnicas de inyección física de fallos utilizando recursos aportados por otras técnicas de inyección.

Con respecto a la validación de sistemas basados en TTA, TTP no es el único protocolo de comunicaciones que se implantará en productos comerciales en un plazo relativamente corto. Existen otros protocolos como FlexRay que ven en la inyección física de fallos un recurso para garantizar al usuario la confiabilidad de sus productos. La representatividad de los fallos físicos inyectados, así como su alcance en localizaciones tan fundamentales como el propio canal de comunicaciones han hecho que empresas relacionadas con el sector se muestren interesadas por este tipo de validación experimental. La línea de investigación abierta plantea diferentes objetivos:

- Aumentar el número de protocolos de comunicaciones basados en TTA sobre los que se lleve a cabo Inyección de Fallos. Tras la divulgación de los resultados del proyecto FIT son varias las empresas que se han interesado por este posible método de validación experimental. La cooperación entre grupos de investigación de universidades europeas sigue abierta para la colaboración con estas empresas.
- Estudiar nuevos entornos de validación que permitan la integración de distintas técnicas de inyección física de fallos con el objetivo de reproducir o coordinar experimentos generando métodos de verificación y evaluación para la obtención de factores de cobertura globales.
- Mantener una dinámica en la actualización de los efectos de los fallos físicos sobre los circuitos actuales que sirva como referente para cualquier técnica de Inyección de Fallos.

Típicamente, las herramientas de inyección física pecan en su falta de observabilidad sobre el sistema a validar tras la inyección de los fallos o sobre la precisión a la hora de seleccionar el instante de inyección. En este sentido, es posible recurrir a herramientas híbridas, hasta ahora poco desarrolladas. Una línea de trabajo futura es realizar la interconexión entre dos herramientas desarrolladas por el Grupo de Sistemas Tolerantes a Fallos de la Universidad Politécnica de Valencia. Las herramientas son AFIT, inyección física de fallos a nivel de pin, e INERTE, inyección de fallos “real-time” implementada por software a través de NEXUS. La unión de ambas herramientas presenta un gran potencial en la validación de sistemas distribuidos ya que combina la Inyección física con las características de observabilidad no intrusiva de NEXUS.

Bibliografía

[Ademaj 2002 (a)] A. Ademaj, "Slightly-Off-Specification Failures in the Time-Triggered Architecture", Proceeding of 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02), pp. 7-12, 2002.

[Ademaj 2002 (b)] A. Ademaj, "A Methodology of Dependable Evaluation of the Time-Triggered Architecture Using Software Implemented Fault Injection", Proceeding of 4th European Dependable Computing Conference (EDCC-4), pp. 172-190, 2002.

[Ademaj 2003] A. Ademaj, "Achieving Fail Silence in Time-Triggered Architecture", Proceeding of 6th IEEE Workshop on Design and Diagnostic of Electronic Circuits and Systems, pp. 165-170, 2003.

[Aidemark *et al*. 2001] J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson, "GOOFI: Generic Object-Oriented Fault Injection Tool", Proceeding of International Conference on Dependable Systems and Networks (DSN2001), pp. 83-88, 2001.

[Aidemark *et al*. 2002] J. Aidemark and J. Vinter and P. Folkesson and J.Karlsson, "Experimental Evaluation of Time-redundant Execution for a Brake-by-wire Application", Proceeding of International Conference on Dependable Systems and Networks (DSN2002), pp. 210-215, 2002.

[Amarasekera y Najm 1997] E.A. Amerasekera, F.N. Najm, "Failure Mechanisms in Semiconductor Devices", Ed. Wiley, segunda edición, 1997.

[Arlat 1990] J. Arlat, "Validation de la Sûreté de Fonctionnement par Injection de Fautes. Méthode – Mise en Oeuvre – Application", Thése présentée à L'Institut National Polytechnique de Toulouse, LAAS Report no. 90-399, Diciembre 1990.

[Arlat *et al*. 1990] J. Arlat, M.Aguera, L. Amat, J.C. Fabre, J.C. Laprie, E. Martins, D. Powel, "Fault Injection for Dependability Validation: A Methodology and Some Applications", IEEE Transactions on Software Engineering, vol. 16, no. 2, pp. 166-182, Febrero 1990.

[Arlat *et al*. 1993] J. Arlat, A. Costes, Y. Crouzet, J.C. Laprie, D. Powel, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems", IEEE Transactions on Computers, vol. 42, no. 8, pp. 913-923, 1993.

[Arlat *et al*. 2002] J. Arlat, M. Rodríguez, A. Albinet, "MAFALDA-RT A Tool for Dependability Assessment of Real-Time Systems", Proceeding of International Conference on Dependable Systems and Networks (DSN2002), pp. 267-272, 2002.

[Austrian Micro Systemes] TTP/C-C1 Communications Controller Data Sheet. TTTech Computertechnik AG. Disponible en http://www.tttech.com

[Avizienis 1976] A. Avizienis, "Fault-Tolerant Systems", IEEE Transactions on Computers, vol. c-25, pp. 1304-1312, Diciembre 1976.

[Avizienis 1978] A. Avizienis, "Fault Tolerance, the Survival Attribute of Digital Systems", Proceeding of the IEEE, vol. 66, no. 10, pp. 1109-1125, 1978.

[Avizienis 1995 (a)] A.A. Avizienis, "Building Dependable Systems:How to Keep with Complexity", Proceeding of 25th International Symposium on Fault-Tolerant Computing (FTCS-25), Special Issue Silver Jubilee, pp. 4-14, Julio 1995.

[Avizienis 1995 (b)] A.A. Avizienis, "The Methodology of N-version Programming", Michel R. Lyu ed., Software Fault Tolerance, pp. 23-46, John Wiley & Sons Ltd., 1995.
[Avizienis 1997] A. Avizienis, "Towards Systematic Design of Fault-Tolerant Systems", IEEE Computer, vol. 30, no. 4, pp. 51-58, Abril 1997.
[Avizienis *et al*. 1971] A. Avizienis, G. Giller, F. Mathur, D. Rennels, J. Rohr, D. Rubin, "The START (Self-Testing-And-Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design", IEEE Transactions on Computers, c-20, no.11, pp. 1312-1321, 1971.
[Avizienis *et al*. 2000] A. Avizienis, J.C. Laprie, B. Randell, "Fundamental Concepts of Dependability", Proceeding of 3rd Information Survivability Workshop (ISW'2000), pp. 7-12, 2000.
[Avizienis y Laprie 1986] A. Avizienis, J.C. Laprie, "Dependable Computing: from Concepts to Design Diversity", Proceedings of IEEE, vol.74, no.5, pp. 629-638, Mayo 1986.
[Avresky *et al*. 1992] D. Avresky, J. Arlat, J.C. Laprie, Y. Crouzet, "Fault Injection for the Formal Testing of Fault Tolerance", Proceeding of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), pp. 345-354, Julio 1992.
[Balkan *et al*. 2002] H. Balkan, D. Patterson, G. Burgess, C. Carlson, P. Elenius, M. Johnson, B. Rooney, J. Sanchez, D. Stepniak, J. Wood, "Flip-Chip Reliability: Comparative Characterization of Lead Free (Sn/Ag/Cu) and 63Sn/Pb Eutectic Solder", Proceeding of IEEE Electronic Components and Technology Conference, pp. 1263-1269, 2002.
[Barak *et al*. 2000] J. Barak, J.L. Barth, C.M. Seidleck, C.J. Marshall, M.A. Carts, R.A. Reed, "Single Event Upset in the Dual-Port-Board SRAMs of the MPTB Experiement", IEEE Transactions on Nuclear Science, vol. 47, no. 3, pp. 712-717, Junio 2000.
[Baraza 2003] J.C. Baraza, "Contribución a la Validación de Sistemas Complejos Tolerantes a Fallos. Nuevos Modelos de Fallos y Técnicas de Inyección de Fallos", Tesis doctoral, Departamento de Informática de Sistemas y Computadoras, Universidad Politécnica de Valencia, Noviembre 2003.
[Baraza *et al*. 2002] 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, vol. 47, no. 10, pp. 847-867, 2002.
[Barton *et al*. 1990] J.H. Barton, E.W. Czeck, Z.Z. Segallo, D.P. Siewiorek, "Fault Injection Experiments using FIAT", IEEE Transactions on Computers, vol. 39, no. 4, pp. 575-581, Abril 1990.
[Baumann *et al*. 2001] R. Baumann, J. Borel, E. Daly, J. Gasiot, J. Gautier, J.L. Leray, J. F. Ziegler, "Earth and Space Single-Events: in present and future electronics", RADECS Short Course, 2001.
[Baumann y Smith 2000] R.C. Baumann, E.B. Smith, "Neutron-Induced Boron Fission as a Major Source of Soft Errors in Deep Submicron SRAM Devices", Proceeding of 38th Annual International Reliability Physics Symposium, pp. 152-157, 2000.
[Benso *et al*. 1998] A. Benso, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "EXFI: A Low Cost Fault Injection System for Embedded Microprocessor-based Boards", ACM Transactions on Desing Automation of Electronic Systems, vol. 3, no. 4, pp. 626-634, Octubre 1998.
[Benso *et al*. 1999] A. Benso, M. Rebaudengo, M. Sonza Reorda, "FlexFi: A Flexible Fault Injection Environment for Microprocessor-based Systems", Proceeding of 18th International Conference on Computer Safety, Reliability and Security (SEFECOM'99), pp. 323-335, 1999.
[Benso *et al*. 2003] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, "A Watchdog Processor to Detect Data and Control Flow Errors", Proceeding of 9th on-line Testing Symposium (OILTS 2003), pp. 144-148, 2003.
[Berger 1961] J.M. Berger, "A Note on Error Detection Techniques for Asymmetric Channels", Information and Control, vol. 4, pp.68-73, 1961.
[Bianchini *et al*. 1994] R. Bianchini, M.E. Crovella, L. Kontothanassis, T.J. LeBlanc, "Software Interleaving", Proceeding of 6th IEEE Symposium on Parallele and Distributed Processing, pp. 56-65, 1994.

[Blanc 1998] S. Blanc, "Diseño de la Etapas de Temporización y Potencia de un Inyector de Fallos de alta velocidad", Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, 1998.
[Blanc 2000] S. Blanc, "Comparación de la Inyección Física de Fallos con Inyección de Fallos implementada por Software para Sistemas Digitales Comerciales de Alta Velocidad", Trabajo de investigación para la tesis doctoral de 12 créditos, Universidad Politécnica de Valencia, 2000.
[Blanc *et al*. 2001] S. Blanc, J.C. Campelo, P.J. Gil, J.J. Serrano, "Stratified Fault Injection using Hardware and Software-implemented Tools", Proceeding of 4th IEEE International Workshop on Desing and Diagnostics of Electronic Circuits and Systems (DDECS'01), pp. 259-266, 2001.
[Blanc *et al*. 2002 (a)] S. Blanc, P.J. Gil, A. Ademaj, H. Sivencrona, J. Torin, "Three Different Fault Injection Techniques Combined to Improve the Detection Efficiency for Time-Triggered Systems", Proceeding of 5th IEEE International Workshop on Desing and Diagnostics of Electronic Circuits and Systems (DDECS'02), pp. 412-415, 2002.
[Blanc *et al*. 2002 (b)] S. Blanc, J. Gracia, P.J. Gil, "A Fault Hypothesis Study on the TTP/C using VHDL-based and Pin-Level Fault Injection Techniques", Proceeding of IEEE 17th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2002), pp. 254-262, 2002.
[Blanc y Gil 2003] S. Blanc, P.J. Gil, " Improving the Multiple Error Detection Coverage in Distributed Embedded Systems", Proceedings of 22nd International Symposium on Reliable Distributed Systems (SRDS2003), pp. 303-312, 2003.
[Blanc *et al*. 2004] S. Blanc, J. Gracia, P.J. Gil, "Experiences during the Experimental Validation of the Time-Triggered Architecture", Proceeding of the Design, Automation and Test in Europe Conference (DATE2004), pp. 30.256 - 262, París, Febrero 2004.
[Blough y Liu 2000] D.M. Blough, P. Liu, "FIMD-MPI: A Tool for Injecting Faults into MPI Applications", Proceeding of 4th International Parallel and Distributed Processing Symposium (IPDPS2000), pp. 241-248, 2000.
[Blough y Torii 1997] D. Blough, T. Torii, "Fault Injection Based Testing of Fault Tolerant Algorithms in Message Passing Parallel Computers", Proceeding of 27th International Symposium on Fault-Tolerant Computing (FTCS-27), pp. 258-267, 1997.
[Boue *et al*. 1998] J. Boue, P. Pétillon, Y. Crouzet, "MEFISTO-L: A VHDL-based Fault Injection Tool for the Experimental Assessment of Fault Tolerance", Proceeding of 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 168-173, 1998.
[Burgun *et al*. 1996] L. Burgun, F. Reblewski, G. Fenelon, J. Bariber, O. Lepape, "Serial Fault Emulation", Proceeding of Design and Automation Conference (DAC), pp. 801-806, 1996.
[Burnsy Wellings 2001] A. Burns, A. Wellings, "Real-Time Systems and Programming Languages", Ed. Addison-Wesley, 2001.
[Byteflight-spec 2001] Especificaciones E100.38 y 100.39, 2001, disponibles en www.byteflight.de.
[Calha y Fonseca 2002] M.J. Calha, J. Fonseca, "Adapting FTT-CAN for the Joint Dispatching of Tasks and Messages", Proceeding of 4th International Workshop on Factory Commuunication Systems, pp. 117-124, 2002.
[Campelo 1999] J.C. Campelo, "Diseño y Validación de Nodos de Proceso Tolerantes a Fallos de Sistemas Industriales Distribuidos", Tesis doctoral, Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, Junio 1999.
[Campelo *et al*. 1999] J.C Campelo, F. Rodríguez, J.J. Serrano, P. Gil, "Design and Validation of a Distributed Industrial Control System's Nodes", 18th. Symposium on Reliable Distributed Systems Lausanne, Octubre 1999.
[Carreira *et al*. 1998] J. Carreira, H. Madeira, J.G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", IEEE Transactions on Software Engineering, vol. 24, no. 2, pp. 125-136, Febrero 1998.
[Castellani-Coulié *et al*. 2001] K. Castellani-Coulié, J.M. Palau, G. Hubert, M.C. Calvet, P.E.

Dodd, F. Sexton, “Various SEU Conditions in SRAM Studied by 3-D Device Simulation”, IEEE Transactions on Nuclear Science, vol. 48, no. 6, pp. 1931-1936, Diciembre 2001.
[Chandra *et al*. 2000] R. Chandra, R.M. Lefever, M. Cukier, W.H. Sanders, “Loki; A State-Driven Fault Injector for Distributed Systems”, Proceeding of International Conference on Dependable Systems and Networks (DSN2000), pp. 237-242, 2000.
[Chen *et al*. 1997] W. Chen, S.K. Gupta, M.A. Breuer, “Analytic Models for Crosstalk Delay and Pulse Analysis under non-linear Inputs”, Proceeding of International Test Conference, pp. 809-818, 1997.
[Chen *et al*. 2002] W. Chen, S.K. Gupta, M.A. Breuer, “Analytical Models for Crosstalk Excitation and Propagation in VLSI Circuits”, IEEE Transactions on Computer-Aided Design and Integrated Circuits and Systems, vol. 21, no. 10, pp. 1117-1131, Octubre 2002.
[Chérèque *et al*. 1992] M. Chérèque, D. Powell, P. Reynier, J.L. Richier, J. Voiron, “Active Replication in Delta-4”, Proceeding of 22nd International Symposium on Fault-Tolerant Computing Systems (FTCS-22), pp. 28-37, 1992.
[Chillarege *et al*. 1992] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D. Moebus, B. Ray, M. Wong, “Orthogonal Defect Classification – A Concept for In-Process Measurement”, IEEE Transactions on Software Engineering, vol 18, no.11, pp. 943-956, Noviembre 1992.
[Choi e Iyer 1992] G. Choi, R. Iyer, “FOCUS: An Experimental Environment for Fault Sensitivity Análisis”, IEEE Transactions on Computers, vol. 41, no. 12, pp. 1515-1526, Diciembre 1992.
[Civera *et al*. 2001] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, “FPGA-Based Fault Injectio for Microporcessor Systems”, Proceeding of 10th Asian Test Conference, pp. 304-312, 2001.
[Constantinescu 1994] C. Constantinescu, “Estimation of Coverage probabilities for Dependability Validation of Fault-Tolerant Computing Systems”, Proceeding of 9th Conference on Reliability, Fault Tolerance, Concurrency and Real Time, Security, pp. 101-106, 1994.
[Constantinescu 1998] C. Constantinescu, “Validation of the Fault/Error Handling Mechanisms of the TeraflopsSupercomputer”, Proceeding of 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 382-389, 1998.
[Constantinescu 2002] C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits”, Proceeding of International Conference on Dependable Systems and Networks (DSN2002), pp. 205-209, 2002.
[Cristian 1988] F. Cristian, “Agreeing on Who is Present and Who is Absent in a Synchronous Distributed System”, Proceeding 18th International Symposium on Fault-Tolerant Computing (FTCS-18), pp. 206-211, 1988.
[CSim 2002] C-Sim, disponible en http://www.c-sim.zcu.cz
[Cukier et al. 1999] M. Cukier, D. Powell, J. Arlat, “Coverage Estimation Methods for Stratified Fault-Injection], IEEE Transactions on Computers, vol 48, no. 7, pp. 707-723, Julio 1999.
[Czeck y Siewiorek 1990] E.W. Czeck, D.P. Siewiorek, “Effects of Transient Gate-Level Faults on Program Behaviour”, Proceeding of 20th International Symposium on Fault-Tolerant Computing (FTCS-20), pp. 236-243, 1990.
[Damm 1986] A. Damm, “The Effectiveness of Software Error-Detection Mechanisms in Real-Time Operating Systems”, in FTCS Digest of Papers. 16th International Symposium on Fault-Tolerant Computing Systems (FTCS-16), pp. 171-176, 1986.
[Daran y Thévenod-Fosse 1996] M. Daran, P. Thévenod-Fosse, “Software Error Analysis: A Real Case Study Involving Real Faults and Mutations”, Proceeding of 3rd Symposium on Software Testing and Analysis (ISSTA-3), pp. 158-171, 1996.
[Darracq *et al*. 2002] F. Darracq, T. Beauchêne, V. Pouget, H. Lapuyade, D. Lewis, P. Fouillat, A. Touboul, “Single-Event Sensitivity of a Single SRAM Cell”, IEEE Transactions on Nuclear Science, vol. 49, no. 3, pp. 1486-1490, Junio 2002.
[David y Thévenod-Fosse 1995] R. David, P. Thévenod-Fosse, “Random Testing of the Data Processing Section of a Microprocessor”, Proceeding of 25th International Symposium on Fault-

Tolerant Computing (FTCS-25), Highlights from Twenty-Five Years, pp. 344, 1995.
[Dawson *et al*. 1996] S. Dawson, F. Jahanian, T. Mitton, T.L. Tung, "Testing of Fault-Tolerant and Real-Time Distributed System via Protocol Fault Injection", Proceeding of 26th International Symposium on Fault-Tolerant Computing (FTCS-26), pp. 404-414, 1996.
[DBench ETIE2 2002] "Fault Representativeness", Deliverable ETIE2 of Dependability Benchmarking Project, July 2002.
[DeMillo 1988] R. DeMillo, "An Extended Overview of the Mothra Software Testing Environment", Proceeding of ACM Sigsoft/ IEEE Workshop on Software Testing, Verification and Analysis, pp. 142-151, 1988.
[Dupuy *et al*. 1990] A. Dupuy, J. Schwartz, Y. Yemini, D. Bacon, "NEST: A Network Simulation and Prototyping Testbed", Communications of the ACM, vol. 33, no. 10, pp. 64-74, Octubre 1990.
[Echtle y Chen 1991] K. Echtle, Y. Chen, "Evaluation of Deterministic Fault Injection for Fault-Tolerant Protocol Testing", Proceeding of 21st International Symposium on Fault-Tolerant Computing (FTCS-21), pp. 418-425, 1991.
[Echtle y Leu 1992] K. Echtle, M. Leu, "The EFA Fault Injector for Fault Tolerant Distributed System Testing", Proceeding of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp. 28-35, 1992.
[Elnozahy *et al*. 2002] E. Elnozahy, R. Melhem, D. Mosse, "Energy-efficient duplex and TMR real-time Systems", Proceeding of 23rd Real-Time Systems Symposium (RTSS 2002), pp. 256-266, 2002.
[Ferreira *et al*. 2002 (a)] J. Ferreira, P. Pedreiras, L. Almeida, J. Fonseca, "Achieving Fault Tolerance in FTT-CAN", Proceeding of 4th International Workshop on Factory Commuunication Systems, pp. 125- 132, 2002.
[Ferreira *et al*. 2002 (b)] J. Ferreira, P. Pedreiras, L. Almeida, J.A. Fonseca, "The FFT-CAN Protocol for Flexibility in Safety-Critical Systems", IEEE Micro, Special Issue on Critical Embedded Automotive Networks, pp. 47-55, Julio-Agosto 2002.
[Fischer *et al*. 1985] J.M. Fischer, N.A. Lynch, M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", Journal of the ACM, vol. 32, no. 2, pp. 374-382, 1985.
[FIT 2002] "Fault Injection for TTA": IST-1999-10748 FIT. Documentos en http://www.cti.ac.at/fit
[FlexRay 2000] BMW AG, DaimlerChrysler AG, Robert Bosch GmbH, General Motors/Opel AG, "FlexRay Requirements Specification", versión 2.0.2., 2000, Información disponible en www.flexray.com.
[Folkesson *et al*. 1998] P. Folkesson, S. Svensson, J. Karlsson, "A Comparison of Simulation-based and Scan Chain Implementation Fault Injection", Proceeding of 28th International Symposium of Fault-Tolerant Computing (FTCS-28), pp. 284-293, 1998.
[Freeman 1996] L.B. Freeman, "Critical Charge Calculations for a Bipolar SRAM Array", IBM Journal of Research and Development, vol. 40, no. 1, pp. 119-129, Enero 1996.
[Fuchs 1996] E. Fuchs, "An Evaluation of the Error Detection Mechanisms in MARS using Software-implemented Fault Injection", Spring-Verlang, Lecture Notes in Computer Science, vol. 1150, pp. 78-90, Octubre 1996.
[Führer *et al*. 2002] T. Führer, B. Müller, W. Dierterle, F. Hartwich, R. Hugel, M. Walther, Robert Bosch GmbH, "Time-Triggered Communication on CAN (Time Triggered CAN – TTCAN)", CAN in Automation (CiA), www.can-cia.de/can/ttcan/, 2002.
[Gaisler 1997] J. Gaisler, "Evaluation of a 32-bits Microprocessor with Built-in Concurrent Error-Detection", Proceeding of 27th International Symposium on Fault-Tolerant Computing (FTCS-27), pp. 42-47, 1997.
[Gaisler 2002] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor based on the SPARC V8 Architecture", Proceeding of International Conference on Dependable Systems and Networks (DSN2002), pp. 409-415, 2002.
[Gérardin 1986] J.P. Gérardin, "Aide à la conception dáppareils fiables et sûrs: le DEFI", Electronique industrielle, no. 116/15-11-1986.

[Gil 1992] P. Gil, "Sistema Tolerante a Fallos con Procesador de Guardia: Validación mediante Inyección Física de Fallos", Tesis doctoral, Departamento de Ingeniería de Sistemas, Computadores y Automática, Universidad Politécnica de Valencia, Septiembre 1992.
[Gil 1996] P.J. Gil, "Garantía de Funcionamiento: Conceptos Básicos y Terminología", DISCA, Universidad Politécnica de Valencia, 1996. Informe interno del Grupo de Sistemas Tolerantes a Fallos (GSTF).
[Gil 1999] D. Gil, "Validación de Sistemas Tolerantes a Fallos mediante Inyección de Fallos en Modelos VHDL", Tesis doctoral, Departamento de Ingeniería de Sistemas, Computadores y Automática, Universidad Politécnica de Valencia, Octubre 1999.
[Gil *et al.* 1997] P.J. Gil, J.C. Baraza, D. Gil, J.J. Serrano, "High Speed Fault Injection for Safety Validation of Industrial Machinery", 8th European Workshop on Dependable Computing, Experimental Validation of Dependable Systems, 1997.
[Gil, Blanc y Serrano 2003] P.J. Gil, S. Blanc, J.J. Serrano, "Pin-level Hardware Fault Injection Techniques", capítulo 2.1 del libro "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation", A. Benso y P. Prinetto *ed*. Book series: Frontiers in Electronic Testing, vol. 23, pp.63-80 , Kluwer Academic Press.
[Goswami e Iyer 1991] K. Goswami, R. Iyer, "A Simulation-Based Study of a Tripple Modular Redundant System using DEPEND", Proceeding of 5th International Test, Diagnosis Fault Tolerant Treatment Conference, pp. 300-311, 1991.
[Goswami e Iyer 1993] K. Goswami, R. Iyer, "Simulation of Software Behaviour under Hardware Faults", Proceeding of 23rd International Symposium on Fault-Tolerant Computing (FTCS-23), pp. 340-347, 1993.
[Goswami *et al.* 1997] K. Goswami, R. Iyer, L. Young, "DEPEND: A Simulation-based Environment for System Level Dependability Analysis", IEEE Transactions on Computers, vol. 46, no. 1, pp. 60-74, Enero 1997.
[Gracia *et al.* 2001] J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Comparison and Application of different VHDL-Based Fault Injection Techniques", Proceeding of 16th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2001), pp. 233-241, 2001.
[Gracia *et al.* 2002 (a)] J. Gracia, D. Gil, J. C. Baraza, and P. J. Gil, "Using VHDL-Based Fault Injection to exercise Error Detection Mechanisms in the Time-Triggered Architecture", Proceeding of Pacific Rim International Symposium on Dependable Computing (PRDC2002), pp. 316-320, 2002.
[Gracia *et al.* 2002 (b)] J. Gracia, S. Blanc, J.C. Baraza, D. Gil, P.J. Gil, "VFIT: Una herramienta Automática para la Inyección de Fallos en VHDL", Seminario Anual de Automática, Electrónica Industrial e Instrumentación (SAAEI 2002), pp 289-292, 2002.
[Gracia *et al.* 2003] J. Gracia, J.C. Baraza, D. Gil, P.J. Gil, "Early Diagnosis of Hard Real-Time Fault-Tolerant Embedded Systems", Proceeding of 6th IEEE Workshop on Design and Diagnostic of Electronic Circuits and Systems (DDECS'03), pp. 157-164, 2003
[Gray 1978] J. Gray, "Notes on Database Operating Systems", in Operating Systems: An Advance Course (R. Bayer, R.M. Graham, G. Seegmuller, Eds.), Lecture Notes in Computer Science, vol. 60, pp. 393-481, Springer-Verlag, Berlín, 1978.
[Grillinger *et al.* 2002] P.Grillinger, A. Ademaj, P. Herout, J. Hlavicka, "Fault Tolerance Evaluation using two based Fault Injection Methods", Proceeding of 8th IEEE International On-Line Testing Workshop, pp. 21-25, 2002.
[Gunneflo 1990] U. Gunneflo, "The Effects of Power Supply Disturbances on the MC6809E Microprocessor", Technical Report 89, Department of Computer Engineering, Chalmers University of Technology, Göteborg (Suecia), 1990.
[Gunneflo *et al.* 1989] U. Gunneflo, J. Karlsson, J. Torin, "Evaluation of Error Detection Schemes using Fault Injection by Heavy-ion Radiation", Proceeding of 19th International Symposium of Fault-Tolerant Computing (FTCS-19), pp. 340-347, Junio 1989.
[Hadzilacos y Toueg 1993] V. Hadzilacos, S. Toueg, "Fault-Tolerant Broadcast and Related Problems", in Distributed Systems (S. Mullender ed.), pp. 97-145, ACM Press, 1993.

[Hammett 1999] R.C. Hammett, "Ultra-Reliable Real-Time Control Systems- Future Trends", IEEE Aerospace and Eletronic Systems Magazine, vol. 14, no. 8, pp. 31-36, Agosto 1999.
[Hamming 1950] R.W. Hamming, "Error Detecting and Error Correcting Codes", Bell Systems Technology Journal, vol. 29, no. 2, pp. 147-160, Abril 1950.
[Han *et al*. 1995] S. Han, H.A. Rosemberg, K.G. Shin, "DOCTOR: An Integrated Software Fault Injection Environment for Distributed Real-Time Systems", Proceeding of IEEE International Computer Performance and Dependability Symposium, pp. 194-203, 1995.
[Hartwich *et al*. 2002] F. Hartwich, B. Müller, T. Führer, R. Hugel, Robert Bosch GmbH, "CAN Network with Time Triggered Communication", CAN in Automation (CiA), www.can-cia.de/can/ttcan/, 2002.
[Hazucha y Svensson 2000] P. Hazucha, C. Svensson, "Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate", IEEE Transactions on Nuclear Science, vol. 47, no. 6, pp. 2586-2594, Diciembre 2000.
[Hedenetz y Belschner 1998] B. Hedenetz, R. Belschener, "Brake-by-wire without Mechanical Backup by usign a TTP-Communication Network", publicado en SAE – Society of Automotive Engineers, 1998.
[Hexel 1999] R. Hexel, "Validation of Fault Tolerance Mechanisms in a Time Triggered Communication Protocol using Fault Injection", Tesis doctoral, Universidad Técnica de Viena, Departamento de Sistemas de Tiempo-Real, Viena (Aurtria), 1999.
[Hopkins *et al*. 1978] A. Hopkins, T. Smith, J. Lala, "FTMP – A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", Proceeding of the IEEE, vol. 66, no. 10, pp.1221-1239, 1978.
[Hoyme y Driscoll 1993] K Hoyme, K. Driscoll, "SAFEbus", IEEE Aerospace and Electronic System Magazine, vol. 8, no. 3, pp. 34-39, 1993.
[Huang *et al*. 1999] S.Y. Huang, K.T. Cheng, W.J. Dai, "Fault Emulation: A New Methodology for Fault Grading", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 10, pp. 1487-1495, Octubre 1999.
[Impagliazzo *et al*. 1998] L. Impagliazzo, P. Marmo, A. Benso, P. Prinetto, "Fault-List Collapsing for Fault Injection Experiments", Proceeding of Annual Reliability and Maintainability Symposium, pp. 383-388, 1998.
[Iyer y Kalbarczyk 2003] R.K. Iyer, Z. Kalbarczyk, "Hardware and Software Error Detection", Course of Fault-Tolerant Digital Systems, Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 2003.
[Jagannath y Rai 1995] A. Jagannath, S. Rai, "Impact of Hardware and Sofware Faults on ARQ Schemes – An Experimental Study", Proceeding of Annual Reliability and Maintainability Symposium, pp. 479-485, 1995.
[Jenn *et al*. 1994] E. Jenn, J. Arlat, M. Rimén, J. Olhsson, J. Karlsson, "Fault Injection into VHDL Models: the MEFISTO Tool", Proceeding of 24th International Symposium on Fault-Tolerant Computing (FTCS-24), pp. 336-344, 1994.
[Johnson *ed*. 1989] B.W. Johnson, "Design and Analysis of Fault Tolerant Digital Systems", *ed*. Addison-Wesley, 1989.
[Kaakani 2001] H. Kaakani, "Radiation Hardened Memory Development at Honeywell", Proceeding of IEEE Aerospace Conference, pp. 5-2249 5-2263, 2001.
[Kanawati *et al*. 1992] G.A. Kanawati, N.A. Kanawati, J.A. Abraham, "FERRARI: A Tool for the Validation of System Dependability Properties", Proceeding of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), pp. 336-344, 1992.
[Kanawati *et al*. 1995] G.A. Kanawati, N.A. Kanawati, J.A. Abraham, "FERRARI: A Flexible Software based Fault and Error Injection System", IEEE Transactions on Computers, vol. 44, no. 2, pp. 248-260, Febrero 1995.
[Kanoun *et al*. 1990] K. Kanoun, J. Arlat, L. Burrill, Y. Crouzet, S. Graf, E. Martins, A. McInnes, D. Powell, J.L. Richier, C. Rodriguez, J. Voiron, "Delta-4 Architecture Guide – Validation", LAAS Report nº 90.434, 1990**.**

[Kao e Iyer 1994] W. Kao, R.K. Iyer, "DEFINE: A Distributed Fault Injection and Monitoring Environment", IEEE CS Press Editor, Fault-Tolerant Parallel and Distributed Systems (D. Pradhan, D.R. Avresky, Eds.), pp. 252-259, California, USA, 1995.
[Kao *et al*. 1993] W. Kao, R.K. Iyer, D. Tang, "FINE: a Fault Injection and moNitorng Environment for tracing UNIX System Behaviour under Faults", IEEE Transactions on Software Engineering, vol. 19, no. 11, pp. 1105-1118, Noviembre 1993.
[Karlsson 1990] J. Karlsson, "Transient Fault Effects in the MC6809E 8-bit microprocessor: A Comparison of Results of Physical and Simulated Fault Injection Experiments", Technical Report 96, Department of Computer Engineering, U. Chalmers, Suecia, 1990.
[Karlsson *et al*. 1994] J. Karlsson, P. Lidn, P. Dahlgren, R. Johansson, U. Gunneflo, "Using Heavy-ion Radiation to Validate Fault-Handling Mechanisms", IEEE Micro, vol. 14, pp. 8-23, Febrero 1994.
[Karlsson *et al*. 1995] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, J. Reisinger, "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", Proceeding of 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5), pp. 150-161, 1995; y "Integration and Comparison of Three Physical Fault Injection Techniques", Predictably Dependable Computing Systems, Capítulo 5, pp. 309-329, Springer-Verlag, 1995.
[Kaufman *et al*. 2000] L.M. Kaufman, S. Shide, B.W. Johnson, "Modeling of Common-Mode Failures in Digital Embedded Systems", Proceeding of Reliability and Maintainability Symposium, pp. 350-357, 2000.
[Kieckhafer *et al*. 1988] R. Kieckhafer, C. Walter, A. Finn, P. Thambidurai, "The MAFT Architecture for Distributed Fault Tolerance", IEEE Transactions on Computers, vol. 37, no. 4, pp. 398-405, Abril 1988.
[Koga *et al*. 1990] R. Koga, N. Natz, S.D. Pinkerton, W.A. Kolasinski, D.L. Oberg, "Ion Beam Characterization for Single Event Phenomena", IEEE Transactions on Nuclear Science, vol 37, no. 6. pp. 1923-1928, Diciembre 1990.
[Koga *et al*. 2001] R. Koga, S. Crain, K. Crawford, P. Yu, "Heavy Ion Induce Hard Errors in Memory Devices with sub-micron Feature Sizes", Proceeding of 6th European Conference on Radiation and Its Effects on Components and Systems, pp. 423-430, 2001.
[Koopman 1999] P. Koopman, "The Ballista – COTS Software Robutness Testing", http://www.ece.cmu.edu/Kopman/ballista/./index.html, 1999.
[Koopman y DeVale 2000] P. Koopman, J. DeVale, "The Exception Handling Effectiveness of POSIX Operating Systems", IEEE Transactions on Sofware Engineering, vol 26, no. 9, pp. 837-848, Septiembre 2000.
[Kopetz 1997] H. Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications", Kluwer Academic Publishers, 1997.
[Kopetz 2000] H. Kopetz, "Composability in the Time-Triggered Architecture", SAE International Congress and Exhibition, March 2000.
[Kopetz 2002] H. Kopetz, "Fault Containment and Error Detection in TTP/C and Flexray", Technical Report 23/2002, Technical University of Vienna, 2002.
[Kopetz y Bauer 2002] H. Kopetz, G. Bauer, "The Time-Triggered Architecture", Proceeding of IEEE Special Issue on Modeling and Design of Embedded Software, vol. 91, no. 1, pp. 112-126, 2002.
[Kopetz y Grünsteidl 1993] H. Kopetz, W. Grünsteidl, " TTP – A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems", Proceeding of 23rd International Symposium on Fault-Tolerant Systems (FTCS-23), pp. 524-533, 1993.
[Kopetz y Merker 1985] H. Kopetz, W. Merker, "The Architecture of MARS", Proceeding of 15th International Symposium on Fault-Tolerant Systems (FTCS-15), pp. 274-279, 1985.
[Kopetz y Ochsenreiter 1987] H. Kopetz, W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems", IEEE Transactions on Computers, vol. C-36, no.8, pp. 933-940, 1987.
[LaBel 2000] K.A. Label, "Natural Space Radiation Effects on Technology",

http://radhome.gsfc.nasa.gov/radhome/Nat_Space_Rad_Tech.htm.
[Lala 1983] J. Lala, "Fault Detection, Isolation and Reconfiguration in FTMP: Methods and Experimental Results", Proceedings of 5th AIAA/IEEE Digital Avionics System Conference (DACS), pp. 21.3.1-21.3.9, 1983.
[Lala 1994] J.H. Lala, R.E. Harper, "Architectural Principles for Safety-Critical Real-Time Applications", Proceeding of the IEEE, vol. 82, no. 1, pp. 25-40, Enero 1994.
[Lala y Alger 1988] J. Lala, L. Alger, "Hardware and Software Fault Tolerance: a Unified Architectural Approach", Proceeding of 18th International Symposium on Fault-Tolerant Computing (FTCS-18), pp. 240-245, 1988.
[Lala y Harper 1994] J. Lala, R. Harper, "Architectural Principles for Safety-Critical Real-Time Applications", Proceeding of the IEEE, vol. 82, no. 1, pp. 25-40, 1994.
[Lamport et al. 1982] L. Lamport, R. Shostak, M. Pease, "The Bizantine Generals Problem", ACM Transactions on Programming Languages and Systems, vol. 4, no. 3, pp. 382-401, 1982.
[Landwher *et al*. 1994] E.C. Landwher, A.R. Bull, J.P. McDermontt, W.S. Choi, "A Taxomomy of Computer Program Security Flaws", ACM Computing Surv., vol. 26, no. 3, pp. 211-254, 1994.
[Laprie 1992] J.C. Laprie, "Dependability: Basic Concepts and Terminology: Dependable Computing and Fault-Tolerant Systems", Springer-Verlag 1992, Traducción al castellano por P.J. Gil, referencia [Gil 1996]
[Laprie 1998] J.C. Laprie, "Dependability of Computer Systems: from Concepts to Limits", Invited paper, IFIP International Workshop on Dependable Computing and its Applications (DCIA'98), pp. 108-126, 1998.
[Leber 1993] G. Leber, "First Results onf the Fault Injection Experiments with EMI in MARS", Technical Report 13/93, Vienna University of Technology, Real Time System Group, 1993.
[Lee *et al*. 2000] H. Lee, Y. Song, H. Shin, "SFIDA: A Software Implemented Fault Injection Tool for Distributed Dependable Applications", Proceeding of 4th International Conference on High Performace Computing in Asia-Pacific Region, pp. 410-415, 2000.
[Lettner *et al*. 1998] R. Lettner, M. Prammer, C. Scherrer, A. Steiniger, "Automated Measurement of Computer Fault Tolerance by Reproducible Fault-Injection Experiments", Proceeding of 3rd Workshop on ADC Modelling and Testing, pp. 557-562, 1998.
[Li y Wu 1995] Y.L. Li, C.W. Wu, "Cellular Automata for Efficient Parallel Logic and Fault Simulation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 14, no. 6, pp. 740-749, Junio 1995.
[Linden y Dahlgren 1995 (a)] P. Linden, P. Dahlgren, "Coverage of Transistor-Level and Gate-Level Stuck-at Faults in CMOS Checkers", IEEE International Symposium on Circuits and Systems (ISCAS-95), 1995.
[Linden y Dahlgren 1995 (b)] P. Linde, P. Dahlgren, "Switch-Level Modelling of Transistor-Level Stuck-at Faults", 13th International Conference on VLSI Design (VLSI-13), 1995.
[LIS 1996] "Guide de la Sûreté de Fonctionnement", Laboratoire d'Ingenierie de la Sûreté de Fontionnement (LIS), Cépaduès – Éditions, 1996.
[Lomelino e Iyer 1986] D. Lomelino, R. Iyer, "Error Propagation in Digital Avionic Processor – A Simulation-Based Study", Proceeding of Real-Time Systems Symposium (RTSS), pp. 218-225, 1986.
[Lönn y Snedsböl 1998] H. Lönn, R. Snedsböl, "Communication Bandwidth Requirements for Sensor and Actuator Data in Distributed Vehicle Control Systems", IEEE International Symposium on Intelligent Vehicles, 1998.
[Lovric y Echtle 1993] T. Lovric, K. Echtle, "ProFI: Processor Fault Injection for Dependability Validation", International Workshop on Fault and Error Detection for Dependability Validation of Computer Systems, 1993.
[Macedonia y Burtzman 1994] M.R. Macedonia, D.R. Burtzman, "MBone Provides Audio and Video Across the Internet", Computer, vol. 27, no. 4, pp. 30-36, Abril 1994.
[Madeira *et al*. 1994] H. Madeira, M. Rela, F. Moreira, J.G. Silva, "RIFLE: A General Purpose Pin-level Fault Injector", Proceedings of 1st European Dependable Computing Conference

(EDCC-1), pp. 199-216, 1994.
[Maderia *et al*. 2000] H. Madeira, D.G. Costa, M. Vieira, "On the Emulation of Software Faults by Software Fault Injection", Proceeding of International Conference on Dependable Systems and Networks (DSN2000), pp. 417-426, 2000.
[Mahmood y McCluskey 1982] M. Namjoo, E.J. McCluskey, "Watchdog Processors and Capability Checking", Proceeding of 12th International Symposium on Fault-Tolerant Computing (FTCS-12), pp. 245-248, 1982.
[Mahmood y McCluskey 1995] M. Namjoo, E.J. McCluskey, "Watchdog Processors and Capability Checking", Proceeding of 25th International Symposium on Fault-Tolerant Computing (FTCS-25), Highlights from Twenty-Five Years, pp. 94, 1995.
[Maia *et al*. 2002] R. Maia, L. Henriques, D. Costa, H. Madeira, "Xception TM – Enhanced Automated Fault-Injection Environment", Proceeding of International Conference on Dependable Systems and Networks (DSN2002), página 547, 2002.
[Martinet 1992] B. Martinet, "Contribution à l'Evaluaction de l'Efficacité du Test Fonctionnel de Microprocesseurs", Thèse de Doctorat de troisième cycle, Institut National Politechnique de Grenoble, Noviembre 1992.
[Martínez *et al*. 1999] J.R. Martínez, P.J. Gil, G. Martín, C. Pérez, J.J. Serrano, "Experimental Validation of High-Speed Fault-Tolerant Systems using Physical Fault Injection", Proceeding of 7th International Working Conference on Dependable Computing for Critical Applications (DCCA-7), pp. 233-250, 1999.
[Martins *et al*. 2002] E. Martins, C. Rubira, N. Leme, "Jaca: A Reflexive Fault Injection Tool based on Patters", Proceeding of International Conference on Dependable Systems and Networks (DSN2002), pp. 483-487, 2002.
[Martins y Rosa 2000] E. Martins, A.C.A. Rosa, "A Fault Injection Approach based on Reflexive Programming", Proceeding of International Conference on Dependable Systems and Networks (DSN2000), pp. 407-416, 2000.
[Maxion y Olszewski 1993] R.A. Maxion, R.T. Olszewski, "Detection and Discrimination of Injected Network Faults", Proceeding of 23rd International Symposium on Fault-Tolerant Computing (FTCS-23), pp. 198-207, 1993.
[McCluskey *et al*. 2001] P.P. Sirvan, N. Saxena, E.J. McCluskey, "Software-Implemented EDAC Protection Against SEUs", Technical Report, Standfort University, Mayo 2001.
[Metra *et al*. 1998] C. Metra, M. Favalli, B. Riccò, "On-line Detection of Logic Errors due to Crosstalk, Delay and Transient Faults", Proceeding of International Test Conference, pp. 524-533, 1998.
[Michel *et al*. 1991] T. Michel, R. Leveugle, G. Sucier, "A New Approach to Control Flow Checking without Program Modification", Proceeding of 21st International Symposium on Fault-Tolerant Computing (FTCS-21), pp. 334-341, 1991.
[Miner 2000] P. Miner, "Analysis of the SPIDER Fault Tolerance Protocols", C. Holloway eds., LFM 2000: 5th NASA Langley Formal Methods Workshop, 2000.
[Miremadi *et al*. 1992] G. Miremadi, J. Karlsson, U. Gunneflo, J. Torin, "Two Software Techniques for on-line Error Detection", Proceedings of 22nd International Symposium of Fault-Tolerant Computing (FTCS-22), pp. 328-335, 1992.
[Mitra *et al*. 2000] S. Mitra, N.R. Saxena, E.J. McCluskey, "Common-mode Failures in Redundant VLSI Systems: a Survey", IEEE Transactions on Reliability, vol. 49, no. 3, pp. 285-295, Septiembre 2000.
[Mitra y McCluskey 2001] S. Mitra, E.J. McCluskey, "Design of Redundant Systems Protected against Common–mode Failures", Proceeding of 19th VLSI Test Symposium (VTS'01), pp. 190-195, 2001.
[Moreno *et al*. 1999] W.A. Moreno, J.R. Sampson, F.J. Falquez, "Laser Injection of Soft Faults for the Validation of Dependability Design", Journal of Universal Computer Science, vol. 5, no. 10, pp. 712-729, Octubre 1999.
[Mostefaoui *et al*. 2000] A. Mostefaoui, M. Raynal, F. Tronel, "The Best of Both Worlds: A Hibrid Approach to Solve Consensus", Proceeding of International Conference on Dependable

Systems and Networks (DSN2000), pp. 513-522, 2000.
[Musseau *et al*. 1996] O. Musseau, F. Gardic, P. Roche, T. Corbière, R.A. Reed, S. Buchner, P. McDonald, J. Melinger, L. Tran, A.B. Campbell, "Analysis of Multiple –bit Upsets (MBU) in a CMOS SRAM", IEEE Transactions of Nuclear Science, vol. 43, no.6, Diciembre 1996.
[NEXUS] Industry Standards and Technology Organization (IEEE-ISTO), "The NEXUS 5001 ForumT Standard for a Global Embedded Processor Debug Interface, IEEE-ISTO 5001T-1999", 1999.
[Nguyen 2001] N.T. Nguyen, "Using Consensus for Solving Conflict Situations in Fault-Tolerant Distributed Systems, Proceeding of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 379-384, 2001.
[Oh *et al*. 2002 (a)] N. Oh, P.P. Shirvani, E.J. McCluskey, "Control-Flow Checking by Software Signatures", IEEE Transactions on Reliability, vol. 51, no.1, pp. 111-122, Marzo 2002.
[Oh *et al*. 2002 (b)] N. Oh, P.P. Shirvani, E.J. McCluskey, "Error Detection by Duplicated Instructions in Super-scalar Processors"; IEEE Transactions on Reliability, vol. 51, no. 1, pp. 63-75, Marzo 2002.
[Oh y McCluskey 2001] N. Oh, E.J. McCluskey, "Procedure Call Duplication: Minimization of Energy Consumption with Constrained Error Detection Latency", Proceeding International Symposium on Defect and Fault Tolerant in VLSI Systems (DFT2001), pp. 182-187, 2001.
[Ohlsson *et al*. 1992] J. Ohlsson, M. Rimén, U. Gunneflo, "A Study of the Effect of Transient Faults into a 32-bit RISC with Built-in Watchdog", Proceeding of 22nd International Symposium on Fault-Tolerant Computing (FTCS-22), pp. 316-325, 1992.
[Ors *et al*. 2001] R. Ors, S. Blanc, J.C. Campelo, J.J. Serrano, "Garantía de Funcionamiento en Sistemas Tolerantes a Fallos", 3ª Jornada Técnica Anual sobre Fiabilidad y Calidad, Valencia 14 Noviembre 2001. Edición Servicio de Publicaciones UPV, pp. 29-68.
[Palau *et al*. 2001] J.M. Palau, G. Hubert, K. Coulie, B. Sagnes, M.C. Calvet, S. Fourtine, "Device Simulation Study of the SEU Sensitivity of SRAMs to Internal Ion Tracks Generated by Nuclear Reactions", IEEE Transactions on Nuclear Science, vol. 48, no. 2, pp. 225-231, Abril 2001.
[Pallierer 2000] R. Pallierer, "Validation of Distributed Algorithms in Time-Triggered Systems by Simulation", PhD Thesis, Universidad Técnica de Viena, Grupo de Sistemas de Tiempo Real, Febrero 2000.
[Pallierer y Smith 1998] R. Pallierer, P. Smith, "BA Airbus Final Report", Technical Report D4.5c, British Aerospace Airbus, Filton, Bristol, IK, Noviembre 1998. ESPRIT Project – TTA.
[Patterson 2002] D.S. Patterson, "Understanding Reliability Criteria for Solder Bumped Devices", Meptec Report, Enero-Febrero 2002, pp. 25-28.
[PDCS2] Predictably Dependable Computing Systems, Web oficial del proyecto: http://www.newcastle.research.ec.org/pdcs.
[Pease *et al*. 1980] M. Pease, R. Shostak, L. Lamport, "Reaching Agreement in the Presence of Faults", Journal of ACM, vol. 27, no. 2, pp. 228-234, 1980.
[Pouget *et al*. 1998] V. Pouget, T. Clin, H. Lapuyade, D. Lewis, P. Fouillat, Y. Maidon, L. Sarger, "Elaboration of a New Pulse Laser System for SEE Testing", 4th International On-Line Testing Workshop, 1998.
[Powell 1994] D. Powell, "Distributed Fault-Tolerance – Lessons from Delta-4", IEEE Micro, vol. 14, no.1, pp. 36-47, 1994.
[Powell *et al*. 1988] D. Powell, G. Bonn, D. Seaton, P. Veríssimo, F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", Proceeding of 18th International Symposium on Fault-Tolerant Computing Systems (FTCS-18), pp. 246-251, IEEE Computer Society Press, 1988.
[Powell *et al*. 1995] D. Powell, E. Martins, J. Arlat, Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation", IEEE Transactions on Computers, vol. 44, no. 2, pp. 261-274, Febrero 1995.
[Powell *et al*. 1997] D. Powell, M. Cukier, J. Arlat, Y. Crouzet, "Estimation of Time-dependent

Coverage", Technical Report, LAAS no. 96466, 1997.
[Pradhan 1980] D. K. Pradhan, "A New Class of Error-Correcting/Detecting Codes for Fault-Tolerant Computer Applications", IEEE Transactions on Computers, vol. c-29, no. 6, pp. 471-481, June 1980.
[Pradhan 1986] J.A. Abraham, V.K. Agarwal, B. Bose, Y. Levendel, E.J. McCluskey, P.R. Menon, J. Metzner, Y. Tohma, "Fault-Tolerant Computing: Theory and Techniques", D.K. Pradhan ed., Prentice-Hall, 1986.
[Pradhan *ed*. 1986] D. K. Pradhan editor, "Fault-Tolerant Computer System Design", Prentice-Hall, 1996.
[Pradhan *ed*. 1996] D. K. Pradhan editor, J.A. Abraham, V.K. Agarwal, B. Bose, Y. Levendel, E.J. McCluskey, P.R. Menon, J. Metzner, Y. Tohma, "Fault-Tolerant Computing: Theory and Techniques", Prentice-Hall, 1986.
[Pradhan y Clark 1993] D.K. Pradhan, J.A. Clark, "REACT: A Synthesis and Evaluation Tool for Fault-Tolerant Multiprocessor Architectures", Proceeding of Reliability and Maintainability Symposium, pp. 428-435, 1993.
[Proteus 1996] Proteus Corporation, "Probestar DVT-100 Application Note", Marzo 1996.
[Rahbaran *et al*. 2002] B. Rahbaran, A. Steininger, M. Dalvai, W. Hubr, "An FPGA-based Development Platform for the Virtual Real-Time Processor Component SPEAR", Proceeding of 5th IEEE International Workshop on Design and Diagnosis of Electronic Circuits and Systems (DDECS'02), pp. 98-105, 2002.
[Rebaudengo *et al*. 1998] M. Rebaudengo, M. Sonza Reorda, A. Benso, P. Prinetto, "A Fault Injection Environment for Microprocessor-based Boards", Proceeding of IEEE International Test Conference (ITC'98), pp. 768-773, 1998.
[Rebaudengo *et al*. 2002] M. Rebaudengo, M. Sonza Reorda, M. Violante, P. Civera, L. Macchiarulo, "An FPGA-based Approach for Speeding-up Fault Injection Campaigns on Safety-Critical Circuits", Journal of Electronic Testing, Theory and Applications (JETTA), vol. 18, no. 3, pp. 261-271, Junio 2002.
[Reed *et al*. 1997] R.A. Reed, M.A. Carts, P.W. Marshall, C.J. Marshall, O. Musseau, P.J. McNulty, D.R. Roth, S. Buchner, J. Melinger, T. Corbière, "Heavy Ion and Proton-Induced Single Event Multiple Upset", IEEE Transactions on Nuclear Science, vol. 44, no. 6, Diciembre 1997.
[Rimén y Ohlsson 1993] M. Rimén, J. Ohlsson, "A Study of the Error Behaviour of a 32 bit RISC Subjected to Simulated Transient Fault Injection", PFCS2 report, pp. 445-460, Septiembre 1993.
[Rodríguez *et al*. 1999] M. Rodríguez, F. Salles, J.C. Fabre, J. Arlat, "MAFALDA: Microkernel Assessment by Fault Injection and Desing Aid", Proceeding of 3rd European Dependable Computing Conference (EDCC-3), pp. 143-160, 1999.
[Rosemberg y Shin 1993] H.A. Rosemberg, K.G. Shin, "Software Fault Injection and its Application in Distributed Systems", Proceeding of 23th International Symposium on Fault-Tolerant Computing (FTCS-23), pp. 208-217, 1993.
[Rushby 2001 (a)] J. Rushby, "A Comparison of Bus Architectures for Safety-Critical Embedded Systems", SRI International, CSL Technical Report, Septiembre 2001.
[Rushby 2001 (b)] J. Rushby, "Bus Architectures for Safety-Critical Embedded Systems", 1er Workshop on Embedded Software, Springer-Verlag Lecture Notes in Computer Science 2001.
[Saleh *et al*. 1990] A.M. Saleh, J.J. Serrano, J.H. Patel, "Reliability of Scrunning Recovery-Techniques for Memory Systems", IEEE Transactions on Reliability, vol. 39, no.1, pp. 114-122, Abril 1990.
[Sampson *et al*. 1998] J.R. Sampson, W.A Moreno, F.J Falquez, "A Technique for Automatic Validation of Fault Tolerant Designs using Laser Fault Injection", Proceeding of 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 162-167, 1998.
[Schmid et al. 2002] U. Schmid, B. Weiss, J. Rushby, "Formally Verified Byzantine Agreement in Presence of Link Faults", Proceeding of 22nd International Conference on Distributed Computing Systems, pp. 608-616, 2002.

[Schuette *et al*. 1986] M.A. Schuette, J.P. Shen, D.P. Siewiorek, Y.X. Zhu, "Experimental Evaluation of Two Concurrent Error Detection Schemes", in FTCS Digest of Papers. 16th International Symposium on Fault-Tolerant Computing Systems (FTCS-16), pp. 138-143, 1986.
[Segall *et al*. 1988] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, T. Lin, "FIAT – Fault Injection based Automated Testing Environment", Proceeding of 18th International Symposium on Fault-Tolerant Computing (FTCS-18), pp. 102-107, 1988.
[Serrano 1987] J.J. Serrano, "Sistema Tolerante a Fallos con Microprocesadores para Control Industrial", Tesis doctoral, Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Valencia, 1987.
[Shivakumar *et al*. 2002] P. Sivakumar, M. Kistler, S.W. Keckler, D. Burger, L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic", Proceeding of International Conference on Dependable Systems and Networks, pp. 389-398, 2002
[Sieh *et al*. 1997] V. Sieh, O. Tschäche, F. Balbach, "VERIFY: Evaluation of Reliability using VHDL-models with Embedded Fault Description", Proceeding of 27th International Symposium of Fault-Tolerant Computing (FTCS-27), pp. 24-27, 1997.
[Siewiorek y Swartz 1982] D. Siewiorek, R. Swartz, "Reliable Computer Systems-Design and Evaluation", 2nd ed. Digital Press, Butterworth, 1992.
[Sivencrona *et al*. 2003 (a)] H. Sivencrona, M. Persson, J. Torin, "Using Heavy-ion Fault Injection to Evaluate Fault Tolerance with Respect to Cluster Size in a Time-Triggered Communication System", Proceeding of 6th IEEE Workshop on Design and Diagnostic of Electronic Circuits and Systems (DDECS'03), pp. 171-176, 2003.
[Sivencrona *et al*. 2003 (b)] H. Sivencrona, P. Johannessen, M. Persson, J. Torin, "Heavy-ion Fault Injections in the Time-Triggered Communication Protocol", Lecture Notes of the 1rst Latin American Symposium on Dependable Computing, pp. 69-80, Octubre 2003.
[Stallings 2001] W. Stalling, "Comunicaciones y Redes de Computadores", 6ª edición, Prentice-Hall 2001.
[Steiner 2001] W. Steiner, "Validation of TTP/C Startup Algorithm", Master Thesis, Universidad Técnica de Viena, Grupo de Sistemas de Tiempo Real, 2001.
[Steiniger y Scherrer 1997] A. Steiniger, C. Scherrer, "On Finding an Optimal Combination of Error Detection Mechanisms based on Results of Fault Injection Experiments", Proceedings of 27th International Symposium on Fault-Tolerant Computing (FTCS-27), pp. 238-247, 1997.
[Stewart 1997] B.A. Stewart, "Board Level Automated Fault Injection for Fault Coverage and Diagnosis Efficiency", Proceedings of International Test Conference (ITC'97), pp. 649-654, 1997.
[Stott *et al*. 2000] D.T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, R.K. Iyer, "NFTAPE: A Framework for Assessing Dependability in Distributed Sytems with Lightweight Fault Injectors", Proceeding of Computer Performace and Dependability Symposium (IPDS2000), pp. 91-100, 2000.
[STSARCES 1999] P. Gil y J. Badiola, con la colaboración de J. Gracia y S. Blanc, "Annex 9: Safety Validation of Complex Components", Final Report of WP3.2, STSARCES: Standards for Safety Related Complex Electronic Systems http://www.safetynet.de/EC-Projects/stsarces/WP32_Annex9_bb-wb_tests.PDF, 1999.
[Temple 1998] C. Temple, "Avoiding the Babbling-Idiot Failure in a Time-Triggered Communication System", Proceeding of 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp. 218-227, 1998.
[Texas Instruments 1997] Texas Instruments, "1997 Digital Design Seminar", 1997.
[Thévenod-Fosse 1991 (a)] P. Thévenod-Fosse, "Software Validation by Means of Statistical Testing: Retrospect and Future Direction", Proceeding of 1st International Working Conference on Dependable Computing for Critical Applications (DCCA-1), in Dependable Computing for Critical Applications, A. Avizienis, J.C. Laprie ed. pp. 23-50, Springer-Verlag 1991.
[Thévenod-Fosse 1991 (b)] P. Thévenod-Fosse, "From Random Testing of Hardware to

Statistical Testing of Software", Proceeding of 5th European Computer Conference (CompEuro'91), Advanced Computer Technology, Reliable Systems and Applications, pp. 200-207, 1991.
[Travese 1988] P. Travese, "AIRBUS and ATR System Architecture and Specification", in U. Voges eds. Software Diversity in Computerized Control Systems, Springer Verlag, 1988.
[Tsai 1998] T. Tsai, "Fault Tolerance via N-modular Software Redundancy", Proceeding of 28th International Symposium on Fault-Tolerant Computing (FTCS-28), pp.201-206, 1998.
[Tsai e Iyer 1995] T.K. Tsai, R.K. Iyer, "FTAPE: A Fault Injection Tool to Measure Fault Tolerance", Proceeding of 10th AIAA Computing in Aerospace, pp. 339-346, 1995.
[Tsai *et al.* 1996] T. Tsai, R. Iyer, D. Jewitt, "An Approach towards Benchmarking of Fault Tolerant Commercial Systems", Proceedings of 26th International Symposium on Fault-Tolerant Computing (FTCS-26), pp. 314-323, 1996.
[Tsai *et al.* 1999] T.K. Tsai, M. Hsueh, H. Zhao, Z. Kalbarczyk, R.K. Iyer, "Stree-based and Path-based Fault Injection", IEEE Transactions on Computers, vol. 48, no. 11, pp. 1183-1201, Noviembre 1999.
[TTP] Communications Controller Data Sheet 0.1, 1.0 y 1.1. TTTech Computertechnik AG. Disponible en http://www.tttech.com
[Velazco *et al.* 1990] R. Velazco, C. Bellon, B. Martinet, "Failure Coverage of Functional Test Methods: a Comparative Experimental Evaluation", Proceeding of International Test Conference (ITC'90), pp. 1012-1017, 1990.
[Velazco *et al.* 2000] R. Velazco, S. Rezhui, R. Ecoffet, "Predinting Error Rate for Microprocessor based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection", IEEE Transactions on Nuclear Science, vol. 47, no. 6, pp. 2405-2411, Diciembre 2000.
[Velazco y Rezgui 2000] R. Velazco, S. Rezgui, "Transient Bitflip Injectionin Microprocessor Embedded Applications", Proceeding of 6th International On-Line Testing Workshop (IOLTW'00), pp. 80-84, 2000.
[Virtanen 2002] A. Virtanen, "Radiation Effects Facility RADEF", Proceeding of International On-Line Testing Workshop (IOLTW'02), página 188, 2002.
[Vmars-TUWien] www.vmars.tuwien.ac.at
[Wakerly 2001] J.F. Wakerly, "Diseño Digital: Principios y Prácticas", 3ª edición Prentice-Hall 2001.
[Wensley *et al.* 1978] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak, C.B. Weinstock, " SIFT: Desing and Analysis of a Fault-Tolerant Computer for Aircraft Control", Proceeding of the IEEE, vol. 66, no. 10, pp. 1240-1255, 1978.
[Wood 1994] A. Wood, "NonStop Availability in a Client/Sever Environment", Tandem Technical Report, no. 94.1, 1994.
[Wrobel *et al.* 2001] F. Wrobel, J.M. Palau, C.M. Calvet, O. Bersillon, H. Duarte, "Simulation of Nucleon-Induced Nuclear Reactions in a Simplified SRAM Structure: Scaling Effects on SEU and MBU Cross Sections", IEEE Transactions on Nuclear Science, vol. 48, no. 6, pp. 1946-1952, Diciembre 2001.
[Wu *et al.* 1998] C.W. Wu, S.A. Hwang, J.H. Hong, "Sequential Circuit Fault Simulation using Logic Emulation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 8. pp. 724-736, Agosto 1998.
[XByWire-DB-6/6-24] X-By-Wire Consortium, Safety Related Fault Tolerant Systems in Vehicle, Project no. BE 95/1329, Contract no. BRPR-CT95-0032, Final Report.
[Yount y Siewiorek 1996] C. Yount, D.P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors", IEEE Transactions on Computers, vol. 45, no. 8, pp. 881-891, Agosto 1996.
[Yuste *et al.* 2003 (a)] 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", International Conference on Dependable Systems and Networks (DSN2003), 2003.
[Yuste *et al.* 2003 (b)] P. Yuste, J.C. Ruiz, L. Lemus, P.J. Gil, "Non-intrusive Software-

implemented Fault Injector in Embedded Systems", 1rd Latin American Dependability Conference (LADC2003).