Validación Experimental por Inyección Física de Fallos de la Garantía de Funcionamiento de un Sistema Multiprocesador Tolerante a Fallos

por

Rafael Javier Martínez Durá

Tesis doctoral

Dirigida por

Prof. D. Pedro Joaquín Gil Vicente

Prof. D. Gregorio Martín Quetglás

Departamento de Informática y Electrónica

Facultad de Ciencias Físicas

Universitat de València

UMI Number: U607739

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U607739
Published by ProQuest LLC 2014. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

UNIVERSITAT DE VALÈNCIA
BIBLIOTECA CIÈNCIES
Nº R. 11497
DATA 12-1-98
SIGNATURA
303.T.D
Nº LIBIS:
1 1987 1570

→ Físicas

Validación Experimental por Inyección Física de Fallos de la Garantía de Funcionamiento de un Sistema Multiprocesador Tolerante a Fallos

por

Rafael Javier Martínez Durá

Dirigida por

Prof. Pedro Joaquín Gil Vicente
Prof. Gregorio Martín Quetglás

TRIBUNAL CALIFICADOR

Presidente		
Juan José Serrano Martín	Cat. de Universidad	U. Politécnica. de Valencia
Vocales		
Antonio Pérez Ambite	Cat. de Universidad	U. Politécnica de Madrid
José Miró Juliá	Prof. Tit. de Universidad	U. de las Islas Baleares
Rafael Ors Carot	Prof. Tit. de Universidad	U. Politécnica de Valencia
Vicente Arnau Llombart	Prof. Tit. de Universidad	U. de Valencia
Suplentes		
Germán Fabregat Llueca	Prof. Tit. de Esc. Universitaria	U. Jaime I de Castellón
Joan Pelechano Fabregat	Prof. Tit. de Universidad	U. de Valencia

UNIVERSITAT DE VALÈNCIA
FACULTAD DE FISICA
DEPARTAMENT D'INFORMATICA
I ELECTRONICA
C/. Doctor Moliner, 50
46100 - BURJASSOT (València)

D. **Pedro Joaquín Gil Vicente**, Profesor Titular del Departamento de Ingeniería de Sistemas, Computadores y Automática de la Universidad Politécnica de Valencia y D. **Gregorio Martín Quetglás**, Catedrático de Ciencias de la Computación de la Universitat de València,

HACEN CONSTAR: Que el licenciado en Informática D. Rafael Javier Martínez Durá ha realizado bajo nuestra dirección, en los laboratorios del Departamento de Informática y Electrónica, el trabajo titulado "**Validación Experimental por Inyección Física de Fallos de la Garantía de Funcionamiento de un Sistema Multiprocesador Tolerante a Fallos**", que se presenta para optar al grado de Doctor en Ingeniería Informática.

Y para que conste, firman el presente certificado en Valencia a 26 de junio de mil novecientos noventa y siete

Fdo.: D. Pedro J. Gil

Fdo.: D. Gregorio Martín

AGRADECIMIENTOS

A los directores de mi tesis quisiera agradecerles el apoyo que me han dado para llevar a buen término el trabajo de investigación.

A Carlos Pérez, por su labor en el desarrollo del software del Proyecto, su ayuda en la depuración de los módulos y por su constante colaboración.

A los compañeros del sótano, José Antonio y Juanjo, mi agradecimiento puesto que siempre han estado a mi lado en los momentos importantes y se han brindado a realizar cualquier tarea. A Javi le digo lo mismo.

Al Profesor Guillermo Ayala, del Dpto. de Estadística, mi consideración, por sus comentarios sobre cómo digerir y realizar el análisis de los datos.

Quisiera mencionar también al Grupo de Sistemas Tolerantes a Fallos de la Universidad Politécnica de Valencia, por la ayuda científica y medios materiales que han puesto a mi disposición.

A todos los miembros del Proyecto FASST, puesto que la colaboración en el Proyecto supuso el comienzo de mi tarea de investigación.

A la familia, por supuesto, y a todos los míos, quiero darles las gracias por el apoyo moral y por la constante preocupación desinteresada que han tenido hacia mi trabajo.

Rafa.

Índice

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

1 Antecedentes y fundamentos de la presente memoria

A medida que la tecnología avanza, los costes de producción decrecen, haciendo posible la incorporación en la sociedad de nuevos sistemas informáticos, cada vez más complejos, que se encargan de realizar o de monitorizar un sin fin de tareas, que pueden ir desde las sencillas aplicaciones domésticas, de tipo lúdico, control de procesos industriales, o sistemas de contabilidad, hasta el complejo manejo de centrales nucleares o el control de aviones. Afinando un poco más en la apreciación, el mismo efecto ha tenido lugar en los sistemas tolerantes a fallos, que son los sistemas que se distinguen porque continúan suministrando el servicio para el cual fueron especificados a pesar de estar afectados por algún tipo de fallo [7]. El incremento en la popularidad de estos sistemas hace que cada vez más se utilicen en aplicaciones críticas que afectan a la seguridad de las personas y por tanto aparecen un gran número de arquitecturas diferentes, con diversas topologías y mecanismos de tolerancia a fallos distintos, encaminadas todas ellas a preservar el correcto funcionamiento de la aplicación o, en el caso de que no sea posible, a evitar averías catastróficas.

La Comunidad Europea, siguiendo esta corriente y tratando ser competitiva tecnológicamente con otros mercados, puso como línea prioritaria en el Segundo Programa Marco la línea de fiabilidad y tolerancia a fallos, subvencionando el Proyecto ESPRIT P5212, llamado FASST (Fault-Tolerant Architecture with Stable Storage Technology), que comenzó en 1991 con una duración de cuatro años. El objetivo fue el diseño, construcción y evaluación de un sistema tolerante a fallos basado en el almacenamiento estable de la información. Los socios implicados en el Proyecto fueron: August Systems LTD (Reino Unido), Bull UK (Reino Unido), ETRA (España), Stollmann (Alemania), la Universidad de Newcastle Upon Tyne (Reino Unido), la Universidad Politécnica de Madrid (España), la Universidad de Valencia (España), el Trinity College de Dublín (Irlanda) y el instituto de investigación IRISA/INRIA de Rennes (Francia). Se propuso la implementación un sistema multiprocesador tolerante a fallos y con memoria compartida, basado en el bus Futurebus+. Las principales líneas de desarrollo fueron las del diseño de los módulos de proceso tolerantes a fallos, el diseño del sistema de entrada/salida, la implementación del sistema operativo, el diseño de una memoria que garantiza el almacenamiento correcto de la información y el diseño de un disco con las

mismas características basado en tecnología RAID. Todos estos elementos debían de ser implementados y comprobados en un prototipo de la arquitectura FASST [112][32].

Una vez finalizado el proyecto, hace ya casi 3 años, si realizamos un análisis de los objetivos que se han cumplido, podemos darnos cuenta de la influencia que han tenido los intereses particulares de los propios socios en los resultados obtenidos. Aunque existen buenos resultados individuales, tanto de carácter industrial como académicos, vagamente se han seguido fielmente los propósitos del proyecto, en el sentido de que no se ha construido una maquina real que recogiera todas las innovaciones tecnológicas que quedaban descritas en las especificaciones del diseño inicial. Si profundizamos un poco en el análisis, el trabajo desarrollado por la Universitat de València, en estrecha colaboración con August Systems, encargada de la parte de los módulos de proceso tolerantes a fallos, aunque lleno de buenos resultados académicos, (que se resumen en la publicación de dos tesis doctorales relacionadas con algoritmos de comunicaciones tolerantes a fallos [2], y con el diseño de un módulo de proceso tolerante a fallos basado en el procesador MIPS R4000 [1], la existencia de una tesis en preparación sobre el diseño de un microkernel que proporciona tolerancia a fallos de forma transparente al usuario y la colaboración en la publicación de un libro, junto con el consorcio, que describe los resultados del proyecto), no concluye con la implementación de un sistema real, listo para la comercialización.

La idea de no tener un prototipo del sistema multiprocesador, donde se verifiquen los supuestos de tolerancia a fallos incorporados y que sirva, dentro del ámbito académico como mínimo, como plataforma de investigación donde desarrollar software, tanto de bajo nivel como de alto nivel o incluso nuevos módulos con propósitos diferentes, desenvocó en una necesidad moral imperiosa de retomar algunas ideas iniciales de FASST, estudiarlas con más detalle, modificarlas y sobre todo implementarlas, para que de esta forma sí que se pueda poner de manifiesto una nueva metodología de tolerancia a fallos, cuyas versiones preliminares fueron discutidas dentro del ámbito del proyecto.

Además, a parte de todo el trabajo que quedó por desarrollar, existe una deficiencia grave en los resultados que FASST debía proporcionar que también debe de ser reconsiderada. Teniendo en cuenta que el principal objetivo del diseño de un sistema de estas características es la obtención de una alta garantía de funcionamiento, uno de los principales problemas que se deben resolver es la evaluación precisamente esta garantía de funcionamiento que supuestamente el sistema proporciona, o si se quiere con otras palabras, se debe analizar la bondad de los mecanismos de tolerancia a fallos que la máquina implementa, cosa que FASST no ha tenido en cuenta para nada. Si realmente se piensa en disponer de un equipo serio, capaz de competir en el amplio mercado existente (como se comentaba al comienzo de la introducción), *¿ cómo puedo tener una confianza justificada en que el sistema va a funcionar correctamente ante la ocurrencia de un fallo ?*. Para dar respuesta a esta pregunta, el trabajo no debe terminar simplemente en la implementación del prototipo y su verificación, sino que es obligado realizar la evaluación de la garantía de funcionamiento que éste ofrece (en este sentido lo mismo ocurre con los ordenadores convencionales, sólo que el objetivo en este caso es medir únicamente sus prestaciones), para cuantificar de alguna forma las respuestas que tiene el sistema ante la presencia de fallos. Este análisis servirá también para poder comparar al prototipo del sistema FASST, con las diversas alternativas existentes en el mercado. Teniendo en cuenta este análisis y sabiendo el objetivo de la aplicación, se podrá por fin elegir el sistema más apropiado que mejor realizará la tarea encomendada.

Para obtener de forma exacta los coeficientes de cobertura en la detección y recuperación de errores (que dan idea de lo buenos que son los mecanismos de tolerancia a fallos introducidos) y los tiempos de latencia en la detección y en la recuperación (que miden las prestaciones de dichos mecanismos), se debe hacer uso de métodos experimentales. El análisis no se puede hacer observando directamente al sistema tolerante a fallos debido a la baja probabilidad de ocurrencia de fallo que tienen los sistemas informáticos, por lo que se debe recurrir a técnicas que introducen (inyectan) fallos de forma voluntaria en el sistema. La técnica que mejor resuelve este problema es la inyección de fallos sobre el

propio prototipo, puesto que es la más exacta, por lo tanto se deberá disponer de un instrumento, llamado inyector de fallos que realice esta labor.

2 Objetivos perseguidos

El autor de la presente memoria ha continuado el trabajo, desde el comienzo del proyecto y prolongando la investigación durante más de 3 años, primero en el desarrollo e implementación de los mecanismos de tolerancia a fallos de los módulos de proceso, que a piro ya eran interesantes, pero que sin embargo no se les había prestado la atención suficiente debido sencillamente a la carencia de un verdadero interés industrial, y segundo a la evaluación de la garantía de funcionamiento del sistema mediante la técnica de la inyección física de fallos.

Considerando ideas complementarias a las que aporta la bibliografía y finalizado ya el diseño de los módulos de proceso nos propusimos la trayectoria de investigación que se resume a continuación:

1. Completar los requisitos y especificaciones de los mecanismos de tolerancia a fallos en los módulos de proceso, siempre siguiendo los cánones que implantaba la arquitectura FASST para realizar un modulo compatible con el sistema multiprocesador inicial.

2. La implementación de dos módulos de proceso, para ser incluidos en el sistema multiprocesador basado en Futurebus+. Dichos módulos deberán cooperar con otros módulos comerciales de memoria y de entrada/salida para formar un sistema completamente operativo.

3. Verificación del correcto funcionamiento del sistema. Este punto incluye la implementación de un monitor en ROM para desarrollar software de aplicación y de diagnosis y la inclusión de los mecanismos software de detección de fallos.

4. Selección y en su caso adaptación de las herramientas necesarias para la evaluación de la garantía de funcionamiento del sistema. En esta fase se debe sintonizar el inyector de fallos (que es una versión mejorada de un inyector desarrollado en la Universidad Politécnica de Valencia [3]) para que funcione a altas frecuencias con una margen reducido de inducción de ruido y para evitar tanto los errores en las medidas como los producidos en el mismo proceso de inyección de los fallos y se debe implantar un entorno de inyección específico para medir los efectos de la inyección en el prototipo.

5. Evaluación experimental de la garantía de funcionamiento por medio de la inyección de fallos. Se va a seguir una metodología que incluye la realización de los modelos matemáticos de fiabilidad , seguridad y disponibilidad y el análisis de los resultados mediante técnicas estadísticas para evitar errores experimentales.

Todo este trabajo nos permitirá conocer si las prestaciones de las ideas desarrolladas para esta arquitectura en concreto son comparables con otras arquitecturas tolerantes a fallos que obedecen a necesidades parecidas. Por tanto, los objetivos principales de este trabajo de investigación serán primero el de implementar el prototipo que demuestre los nuevos mecanismos de tolerancia a fallos propuestos en el módulo de proceso y segundo el de realizar un análisis de las características que hacen a la arquitectura FASST ser un ordenador tolerante a fallos. Es decir se va a realizar una medida objetiva de los mecanismos que garantizan el correcto funcionamiento del sistema a pesar de la ocurrencia de fallos y como resultado se van a presentar unos valores cuantificados de estas medidas.

3 Estructura de la memoria

La memoria de investigación comienza con la introducción de los sistemas tolerantes a fallos y se recuerda el estado del arte en este campo. Básicamente se ha tratado de introducir una terminología precisa que después nos permitirá referenciar de forma adecuada cualquier aspecto relacionado con el tema. A continuación se describe brevemente el proyecto FASST, y con más detalle se describe la implementación de los módulos de proceso tolerantes a fallos y del prototipo donde se va a realizar los experimentos . Dos aspectos fundamentales deben quedar claros después de estudiar este capítulo: la inclusión en el modulo de los nuevos mecanismos que garantizan el funcionamiento correcto del sistema a pesar de la ocurrencia de fallos y el desarrollo del software de demostración, que implementa los mecanismos de tolerancia a fallos y que permitirá realizar el test del sistema. También se describe la aplicación del usuario que servirá como carga estándar para realizar la evaluación del prototipo. En el Capítulo 4 se presenta el Futurebus+, poniendo especial énfasis en sus características de tolerancia a fallos y de tiempo real, que le hacen ser único en el género. Este bus es el precursor de un nuevo estándar IEEE de bus multimedia, más simplificado (más barato) pero con las mismas prestaciones*. La comprensión del funcionamiento del FB+, controlado mediante una serie de registros accesibles desde todo el sistema es fundamental para poder entender el funcionamiento del software del demostrador.

A partir del Capítulo 5 se describe la evaluación del módulo de proceso. En este capítulo se analiza la metodología empleada para validar la garantía de funcionamiento de los sistemas tolerantes a fallos y se describen los modelos utilizados para calcular los parámetros más representativos del sistema. En esta parte se incluye también la estimación de la tasa de fallos de los módulos de proceso, mediante el uso de un programa de fiabilidad basado en un estándar militar. Se introducen tanto los modelos probabilísticos, basados en el cálculo de una serie de medidas y de tiempos de latencia, como los modelos estocásticos, en donde destacan los modelos con Cadenas de Markov. Fruto del análisis de la topología del prototipo y de las características de los mecanismos de tolerancia a fallos implementados se va a presentar un nuevo modelo macroscópico del sistema que va a ser también adecuado para los sistemas con degradación.

En el siguiente capítulo, continuando con la descripción de la metodología empleada para realizar la validación, se explica como parametrizar los modelos vistos previamente. Se introduce la inyección de fallos como un método utilizado para ejercitar al sistema y producir errores de forma artificial, con el objetivo de medir o analizar los efectos que éstos producen. Se repasan los distintos tipos de inyección existentes y se describe la arquitectura del inyector de fallos utilizado en la experiencia.

En el Capítulo 7 se describe el experimento. Se construye el conjunto FARM que recoge todos los atributos de especifican la inyección de fallos que se va a llevar a cabo. Se realiza en primer lugar, el análisis de las coberturas de detección y de recuperación del sistema, basándose en un nuevo modelo microscópico de la inyección de fallos y a continuación se utilizan los modelos del Capítulo 5 para calcular los parámetros más característicos del sistema: fiabilidad, disponibilidad y seguridad de funcionamiento y se presentan los resultados estadísticos mediante una serie de tablas y gráficas que relacionan el efecto que causan los distintos tipos de fallos sobre el sistema.

En las conclusiones se resume el objeto de la investigación y sus resultados, realizando una breve discusión sobre todos los aspectos mencionados en la memoria y se añade una lista con las posibles líneas de investigación que emergen detrás de esta memoria.

Los resultados obtenidos describen los tiempos de latencia de detección y reconfiguración de errores del nuevo prototipo del sistema FASST. Se obtienen también las coberturas de detección y de

---

* En el momento en que se empezó a diseñar el sistema, este estándar aún no existía.

recuperación de fallos del prototipo mediante la resolución de dos modelos teóricos de cobertura: un modelo microscópico para la inyección de fallos y un modelo macroscópico de fiabilidad, disponibilidad y seguridad.

Capítulo 2

Conceptos y nomenclatura básica sobre Tolerancia a fallos

1 Introducción

En este capítulo se pretende dar una visión general sobre lo que significa la tolerancia a fallos dentro del campo de la arquitectura de computadores. Se van a presentar una serie de definiciones básicas y se va a explicar la terminología más comúnmente utilizada, ya que a veces resulta un poco confusa dado que la mayoría de los conceptos se derivan de palabras inglesas o francesas.

La tolerancia a fallos no es un tema nuevo. Los primeros ordenadores digitales, debido a la poca fiabilidad de sus componentes, ya incluían en sus diseños algunas técnicas que intentaban conseguir un funcionamiento correcto. Si miramos al pasado, sobre los años 50 se construyó el sistema BRC (Bell Relay Computer) que tenía dos CPUs [5]. En el momento en que una fallaba, la otra pasaba a ejecutar la siguiente instrucción. El IBM 650, el UNIVAC y el Whirlwind I incorporaban códigos de paridad para comprobar los resultados de las transferencias de datos [8]. En [4] se describe el ordenador EDVAC, diseñado en 1949, considerado como el primer ordenador donde se duplica la unidad aritmética. Con la llegada del transistor desapareció el énfasis que se ponía en la tolerancia a fallos, eclipsado por el ansia de aumentar el rendimiento y la velocidad. Años más tarde, con el comienzo de la era espacial, se volverían a utilizar técnicas de tolerancia a fallos en el diseño de ordenadores apropiados para realizar tareas críticas.

A partir de los años 70, se le ha prestado mucha atención a este campo, publicándose muchas revistas de esta índole y empezando en 1971 el simposium internacional sobre tolerancia a fallos (Fault Tolerant Computer Symposium - FTCS) que se celebra año tras año. Hoy en día existe una gran variedad de ordenadores comerciales clasificados como de tolerantes a fallos. Se diferencian especialmente en la aplicación que desarrollan y en los objetivos que persiguen.

2 Definiciones básicas

El término más utilizado en el campo de la tolerancia a fallos es la garantía de funcionamiento de un sistema informático (*dependability*), puesto que engloba todos los aspectos relacionados con el tema. En [7][6] se define como la propiedad que permite a los usuarios de un sistema depositar una confianza justificada en el servicio que éste les proporciona. El usuario puede ser otro sistema que interactúa con el primero o bien ser una persona. El servicio que proporciona el sistema depende de la apreciación que tenga el usuario de cuál es el comportamiento correcto que tiene que dar el sistema. Dependiendo de la aplicación, se tendrá que poner más énfasis en ciertos aspectos de la garantía de funcionamiento, como puede ser en que:

- El sistema funcione sin interrupciones (*reliability*)
- El sistema no provoque averías catastróficas (*safety*)
- El sistema esté disponible el máximo tiempo posible (*availability*)
- El sistema sea fácilmente reparable (*maintainability*)
- El sistema impida el acceso no autorizado (*confidentiality*)
- El sistema impida la alteración inadecuada de la información (*integrity*)

Históricamente los sistemas fiables sólo se utilizaban en aplicaciones militares, industriales, espaciales, aeronáuticas o telemáticas (comunicaciones), para evitar que los fallos produjeran un grave impacto económico o la pérdida de vidas humanas. Sin embargo, actualmente se aplican técnicas de tolerancia a fallos en los ordenadores de propósito general debido principalmente a varios factores:

- Se instalan en ambientes industriales mucho más ruidosos, con condiciones adversas de temperatura, de humedad y con interferencias electromagnéticas.
- Los utilizan operarios no especializados que pueden cometer errores, por lo que se hace necesario que el sistema tolere el mal uso que se pueda hacer de él.
- Se incrementa el costo de reparación (baja el hardware y sube la mano de obra).
- Los sistemas son más grandes, habiendo muchos más componentes que pueden fallar.

Figura 2.1: Arbol de la garantía de funcionamiento

La Figura 2.1 muestra un resumen de los aspectos más importantes que tienen cierta relación con la fiabilidad de los sistemas informáticos.

3 Causas de la disminución de la garantía de funcionamiento

Los *impedimentos* de la garantía de funcionamiento del sistema son circunstancias no deseadas (aunque no inesperadas) que se producen en un sistema y que reducen su garantía de funcionamiento. Son las causas que hacen que no podamos confiar en que nuestro sistema va a funcionar correctamente durante toda la vida y que siempre se va a comportar de la misma manera. Se dividen en tres tipos: fallos, errores y averías.

- Un fallo es un defecto o imperfección física en el hardware o software del sistema. El fallo, cuando se manifiesta da lugar a un error.
- Un error es un estado interno incorrecto en el sistema producido por la existencia de un fallo.
- Una avería se produce cuando el usuario aprecia que el sistema no funciona bien. Cuando se produce la avería, el servicio que entrega el sistema no coincide con el que debía suministrar teniendo en cuenta las especificaciones iniciales. Las averías se deben a errores.

La Figura 2.2 considera la relación que existe entre los fallos, los errores y las averías, teniendo en cuenta el momento en que se producen.

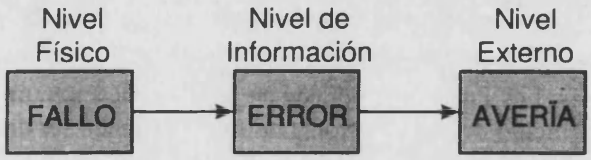


Figura 2.2: Relación entre fallos, errores y averías

Esta transformación no se produce de forma simultánea en el tiempo, sino que desde que se produce el fallo (t_f) hasta que se manifiesta el error (t_e) existe un tiempo de inactividad, llamado latencia del error (ver Figura 2.3). Durante este tiempo de latencia, se dice que el fallo no es efectivo y que el error está latente. De forma análoga se puede definir la latencia de detección del error y la latencia de producción de la avería.

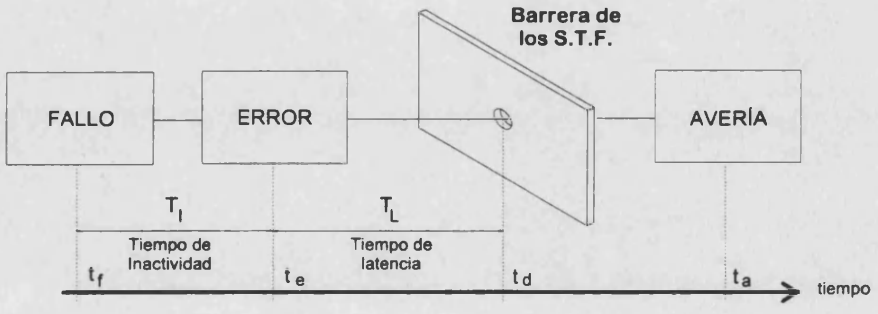


Figura 2.3: Patología de los fallos

3.1 Fallos

Los fallos son defectos o imperfecciones físicas que se producen en los componentes. Son de naturaleza muy diversa y se pueden clasificar teniendo en cuenta aspectos muy diferentes. A continuación se presenta un clasificación de los mismos según diversos factores:

Origen de los fallos: Dependiendo de su origen se clasifican en:

- **Momento de producción:** Dependen de la etapa de desarrollo del sistema donde se producen:
  - *Diseño y especificación*: Son fallos humanos que se deben a errores o a especificaciones incompletas.
  - *Implementación o instalación*: Pueden ser fallos tanto hardware como software.
  - *Operacionales*: Se producen durante la vida de operación del sistema debido bien a una causa física bien a un problema humano.
  - *Mantenimiento*: Se producen durante la fase de mantenimiento del sistema.
- **Lugar de producción:** Según su extensión con respecto a las fronteras del sistema:
  - *Internos*: Son defectos en algún componente hardware o software.
  - *Externos*: Debidos a las interferencias que tiene el sistema con el entorno físico (perturbaciones electrostáticas y electromagnéticas, radiaciones, perturbaciones de temperatura o de presión, vibraciones, polvo, contaminación atmosférica) o a la interacción con el entorno humano.
- **Causa fenomenológica que los produce:**
  - *Humanos*: Los fallos se producen en la especificación, implementación o mantenimiento debidos a errores humanos. Pueden ser debidos a la mala intención del operador o a intrusiones de otros usuarios no autorizados.
  - *Físicos*: Debidos a la interacción del sistema con el entorno.

Persistencia: Los fallos se clasifican dependiendo de su duración en:

- **Permanentes:** Es un cambio irreversible en el componente.
- **Temporales:** Presentes durante un periodo corto de tiempo. Se dividen a su vez en:
  - *Transitorios*: Se deben a interferencias externas y por lo tanto la forma en que aparecen y su duración es aleatoria.
  - *Intermitentes*: Aparecen también aleatoriamente pero de forma repetitiva. Tienen lugar cuando se produce cierta combinación en el sistema. Pueden ser fallos sensibles a patrones, producidos cuando se da una combinación determinada en los niveles de ciertas señales adyacentes al lugar del fallo (muy usuales en memorias estáticas), o fallos dependientes de la carga del sistema, que son fallos permanentes en estado latente que se manifiestan según el contenido de ciertas variables.

Naturaleza: Según la naturaleza del fallo se clasifican en:

- **Intencionados:** Se producen de forma deliberada. Pueden ser:
  - *Troyanos*: Son fallos internos que se deben a la incorporación de lógica malévola, como errores de diseño intencionados y virus.
  - *Intrusión*: Son fallos externos intencionados, que producen violaciones de seguridad del sistema.
- **Accidentales:** Son fallos que se producen de forma fortuita.

Valor: Dependiendo del estado final del fallo:

- **Determinados:** El sistema evoluciona hacia un valor fijo.

- **Indeterminados:** El fallo produce un estado aleatorio entre los niveles lógicos 0 y 1.

En la Figura 2.4 se muestra una combinación con los tipos de fallos que más comúnmente se producen. Es interesante observar la última columna que recoge la denominación más usual de esta clase de tipos de fallos.

Naturaleza		Origen						Persistencia		Denominación usual
		Causa fenomenológica		Entorno del sistema		Fase de creación				
Fallos accidentales	Fallos intencionados	Fallos físicos	Fallos humanos	Fallos internos	Fallos externos	Fallos de diseño	Fallos de operación	Fallos permanentes	Fallos temporales	
√		√		√			√	√		Fallos físicos
√		√			√		√	√		
√		√			√		√		√	Fallos transitorios
√		√		√			√		√	Fallos intermitentes
√			√	√		√			√	
√			√	√		√		√		Fallos de diseño
√			√		√		√		√	Fallos de interacción
	√		√	√		√		√		Lógica malévola
	√		√	√		√			√	
	√		√		√		√	√		Intrusiones
	√		√		√		√		√	

Figura 2.4: Combinaciones con los tipos de fallos

Los mecanismos de los fallos son las causas (generalmente internas) por las cuales se produce el fallo, mientras que el modo del fallo hace referencia a la forma en la que se manifiesta un fallo.

Los mecanismos de fallo en los circuitos integrados son muy diversos, puesto que podemos hablar de defectos de superficie, de óxido, de difusión, de metalización, defectos en las entradas y las salidas o defectos de volumen. Como se puede ver en la gráfica de la Figura 2.5 dependen altamente de la tecnología utilizada en la fabricación del integrado.

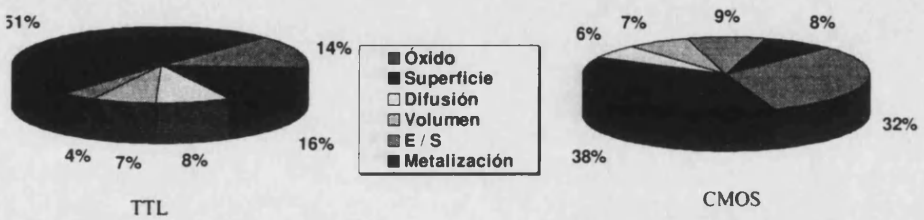


Figura 2.5: Mecanismos de los fallos en los circuitos integrados

Los modos de fallo dependen de la funcionalidad y del tipo de circuito que se esté considerando. Para poder tabularlos se realizan modelos de fallos que representen a la mayoría de fallos físicos que puedan haber en un sistema.

En circuitos lógicos digitales
Señales a nivel fijo (pegado-a o stuck-at)
Puentes entre conexiones
Circuitos abiertos
En memorias ROM
Direccionamiento incorrecto
Selección incorrecta del circuito
Contenido incorrecto de una celdilla
En memorias RAM
Escritura múltiple debida a un acoplamiento capacitivo entre celdillas
Aumento del tiempo de acceso por excesiva carga en la salida
Inercia en la escritura de largas series de '1' y '0' por saturación del amplificador sensor
Cambio transitorio del valor de una celdilla por acoplamiento de los caminos
Cambio del contenido de alguna celda por radiaciones a, o a ruido de recombinación
Ausencia de refresco por fugas en la retención de cargas entre dos refrescos sucesivos
En circuitos lógicos programables
Fallos a nivel fijo ('0' ó '1')
Fallos de contacto o fallos en puntos de cruce
Fallos por puentes
En microprocesadores
Señales a nivel fijo de forma transitoria
Retardos de propagación no especificados
Transformación de circuitos combinacionales en secuenciales
Cambios en las señales debidos a sensibilidad a patrones e instrucciones

Tabla 2.1: Modos de fallo en los circuitos integrados

3.2 Errores

Un error es un estado del sistema que a veces deriva en una avería. La producción de la avería depende de una serie de factores:

1. La composición interna del sistema: La topología del sistema puede enmascarar errores para que no den lugar a la avería. El método consiste en replicar los elementos del sistema (redundancia) y realizar una votación sobre los múltiples resultados obtenidos para estimar por mayoría cual es el resultado correcto. La redundancia no se aplica únicamente en el hardware, sino que también podemos hablar de redundancia temporal o de redundancia software en los datos. Un caso particular es la redundancia no intencionada que siempre incorpora cualquier sistema informático, que también enmascara los errores.

2. La actividad del sistema: Existirán errores que únicamente se manifiesten al activarse alguna parte del programa y que serán dependientes de la carga del sistema.

3. El concepto de avería que tenga el usuario. Aunque a veces una avería resulta evidente, hay otras que en que es un tema subjetivo del usuario. Pueden haber usuarios, por ejemplo, que consideren que un sistema está averiado cuando su rendimiento se reduce por debajo de cierto límite.

3.3 Averías

Las averías suponen que el servicio que proporciona el sistema no cumple con las especificaciones de diseño. Existen muchos tipos de averías que se pueden clasificar también de muchas maneras. Las taxonomías típicamente tienen en cuenta el dominio de la avería, que nos indica el factor al que afecta (valor, tiempo), la percepción que tiene el usuario de la avería dependiendo de cómo le afecte (consistentes o inconsistentes) y las consecuencias que tienen en el entorno (benignas o catastróficas).

4 Medios para obtener una garantía de funcionamiento

Los *medios* para obtener una garantía de funcionamiento del sistema son los métodos y técnicas que hacen que:

- el sistema pueda suministrar un servicio fiable en el que se pueda confiar.
- el usuario confíe en que el sistema tiene la capacidad de darle este servicio.

Las técnicas que se emplean para lograr la tolerancia a fallos consisten en la aplicación de una serie de barreras, a distintos niveles, que tratan de evitar que se llegue a producir una avería. Estas barreras son:

Barrera 1: Prevención de fallos (*Fault avoidance*): El objetivo es reducir la posibilidad de que se produzca un fallo en el sistema, y para ello se eligen componentes de alta fiabilidad, se realiza un diseño e implementación extremadamente cuidadoso y se trata de proteger al sistema contra los agentes externos provocadores de fallos.

Barrera 2: Enmascaramiento de fallos (*Fault masking*): Si la barrera uno no ha sido capaz de impedir que se produjera un fallo, la nueva estrategia consistirá en evitar que el fallo derive en un error. Mediante la aplicación de técnicas de redundancia estática, se va a suministrar al sistema la información necesaria para evitar los efectos de los fallos. La redundancia estática consiste en la construcción de sistemas con varios elementos replicados, llamados Sistemas N-Modular redundantes. Para evitar fallos en el diseño de estos elementos se utilizará la diversificación funcional, que es una técnica que intenta construir módulos equivalentes pero de diferente manera.

Barrera 3: Tratamiento del error: Una vez que se ha producido el error, la única solución que queda es tratar de eliminarlo antes de que se produzca la avería total del sistema. Para ello, el sistema debe iniciar un proceso de detección, diagnóstico, aislamiento, reconfiguración y recuperación del error. En este proceso el sistema se degrada para seguir funcionando a costa de reducir su rendimiento. Los sistemas que son capaces de reconfigurarse se denominan sistemas con redundancia dinámica.

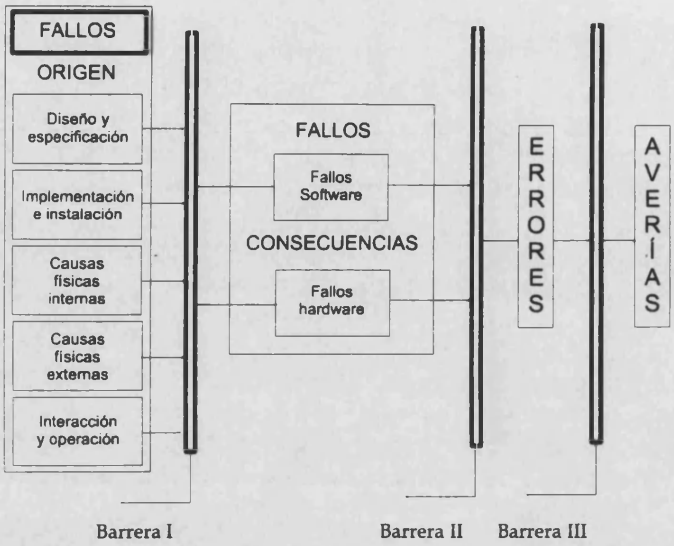


Figura 2.6: Medios para obtener una alta garantía de funcionamiento

Los sistemas tolerantes a fallos hacen uso intensivo de las técnicas de redundancia [25] para paliar los efectos de los errores. Estas técnicas se incluyen en niveles muy distintos dentro del diseño del sistema. Las técnicas más usuales son las siguientes:

- Utilización de componentes hardware extra (redundancia en el hardware).
- Repetición de las operaciones y comparación de los resultados (redundancia temporal).
- Codificación de los datos (redundancia en la información).
- Realización de varias versiones de un mismo programa y uso de técnicas de consistencia (redundancia en el software).

Una vez que se ha terminado el diseño del sistema, tendremos que evaluar su comportamiento para comprobar que cumple con las expectativas que se realizaron inicialmente de que iba a funcionar correctamente a pesar de la existencia de fallos. La evaluación se realiza a dos niveles:

- Eliminación de fallos (*Fault removal*): Consiste en reducir el número de fallos que se pueden producir en un sistema y en limitar su alcance. Para ello se realiza un mantenimiento consistente en un proceso de verificación que trata de averiguar si el sistema cumple con sus especificaciones de funcionamiento. En el caso de que se detecte alguna anomalía se ejecuta un proceso de diagnosis para encontrar las causas que evitan la correcta verificación. Una vez encontrado el error se elimina y de nuevo se vuelve a verificar el sistema.

- Predicción de fallos (*Fault Forecasting*): Consiste en la obtención a priori de la garantía de funcionamiento del sistema. Para ello se realiza una evaluación del comportamiento que tiene el sistema cuando se produce un fallo. La evaluación se puede hacer mediante la resolución de modelos teóricos o de forma experimental, introduciendo fallos artificialmente en el sistema (inyección de fallos) y midiendo sus consecuencias. La evaluación de la garantía de funcionamiento de un sistema real es uno de los objetivos de la presente investigación, por lo tanto se va a detallar de forma mucho más extensa en capítulos posteriores.

5 Atributos de la garantía de funcionamiento

Los atributos de la garantía de funcionamiento son una serie de propiedades diferentes pero complementarias que permiten valorar la calidad del sistema. Estos atributos definen las características del sistema teniendo en cuenta las facilidades de tolerancia a fallos que implementa. Muestran una serie de valores cuantificados que me permiten comparar unos sistemas con otros. A continuación se resumen los atributos más importantes:

- Fiabilidad R(t) (*Reliability*): Es la probabilidad condicional de que el sistema funcione correctamente en el intervalo $[t_0,t]$, dado que el sistema funcionaba correctamente en el instante t_0. La probabilidad de fallo (*infiabilidad*) es lo contrario del anterior. Caracteriza a sistemas que realizan funciones críticas que no se pueden parar ni siquiera para repararlos (aviones) y a sistemas en donde la reparación es muy cara (satélites).

- Disponibilidad A(t) (*Availability*): Es la probabilidad de que el sistema funcione correctamente y esté disponible para realizar sus funciones en un instante de tiempo determinado t. Cuando t tiende a infinito indica la fracción del tiempo en que el sistema está funcionado de forma correcta. Este parámetro depende de la fiabilidad y de la rapidez con la que se repare el sistema.

- Seguridad S(t) (*Safety*): Es la probabilidad de que el sistema esté funcionando correctamente o de que deje de funcionar de forma que no entorpezca la operación de otros sistemas ni comprometa la seguridad de alguna persona relacionada con dicho sistema.

- Capacidad de Rendimiento P(L,t) (*Performability*): Existen sistemas que ante la ocurrencia de un fallo disminuyen sus prestaciones (la degradación funcional se realiza de forma automática). El rendimiento es la probabilidad de que el sistema en un instante t tenga al menos unas prestaciones iguales a L. Se diferencia de la fiabilidad en que esta mide la disponibilidad a que todas las funciones se realizan de forma correcta y el rendimiento sólo tiene en cuenta un subconjunto de funciones.

- Mantenibilidad M(t) (*Maintainabiliy*): Mide la facilidad con la cual se puede reparar un sistema. Es la probabilidad que tiene un sistema que ha fallado de estar reparado en un periodo de tiempo t. La diagnosis automática incrementa M puesto que uno de los factores que más influye en el tiempo de reparación es el tiempo que se tarda en localizar el fallo.

- Comprobabilidad (*Testability*): Mide la facilidad con la que se puede aplicar un test al sistema para verificar su correcto funcionamiento. Se basa en la aplicación de una serie de técnicas de tolerancia a fallos especiales para la detección y localización de problemas.

En función de la aplicación que vaya a desarrollar el sistema, se tendrá que poner mayor énfasis en ciertos atributos de la garantía de funcionamiento. Básicamente se pueden distinguir cuatro tipos diferentes de sistemas:

1. Sistemas de larga vida: Los requerimientos típicos son del 95 % de probabilidad de estar operativo en 10 años. Estos sistemas permiten tiempos de reparación elevados, por lo que el operador puede reconfigurarlos de forma manual.

2. Sistemas para aplicaciones críticas: Los requerimientos son del 99'99999 % de probabilidad de no fallar en 3 horas. El funcionamiento es crítico para las vidas humanas, el medio ambiente o la protección de otros equipos (aviones, sistemas militares y controladores industriales).

3. Sistemas de alta disponibilidad: Son apropiados para aplicaciones en donde los usuarios deben tener una probabilidad alta de recibir servicio cuando lo requieren. Se utilizan en los sistemas bancarios y en los de tiempo compartido.

4. Sistemas de difícil mantenimiento: Se utilizan cuando las operaciones de mantenimiento son muy caras o imposibles de realizar. Tratan de retrasar el mantenimiento del sistema el mayor tiempo posible mediante la utilización de técnicas de reconfiguración. Un caso típico es el de las sondas espaciales.

6 Evolución del sistema en respuesta a la producción de un fallo

En el momento en el que se produce un fallo en el sistema, se activan una serie de mecanismos para prevenir la aparición de la avería. Estos mecanismos están ordenados de forma cronológica según el momento en que se activan. Se enumeran a continuación:

- Detección del fallo (*Fault detection*): Existe un periodo de tiempo (latencia del fallo) que transcurre desde que tiene lugar el fallo hasta que se detecta. La detección fuera de línea (*off-line*) utiliza programas de diagnóstico, mientras que la detección en línea (*on-line*) realiza una detección en tiempo real mediante una serie de mecanismos que están activos o que se ejecutan concurrentemente con la aplicación (Paridad y duplicación).

- Aislamiento del fallo (*Fault confinement*): Previene la contaminación de otras áreas del sistema, limitando el efecto del fallo.

- Diagnosis (*Diagnosis*): El objetivo del mecanismo es informar sobre la localización y las propiedades del fallo. Normalmente cuando se detecta el fallo ya se proporciona esta información.

- Reintento (*Rollback*): Se repite la operación para evitar fallos transitorios. Es importante tener en cuenta la latencia del fallo al hacer el reintento.

- Reconfiguración (*Reconfiguration*): Tiene lugar cuando se detecta un fallo permanente. El sistema se reconfigura bien sustituyendo el componente bien aislándolo del resto. También se puede degradar el sistema (*graceful degradation*) para seguir funcionando a costa de reducir el rendimiento.

- Recuperación (*Recovery*): Después de reconfigurar el sistema se debe dejarlo en el mismo estado previo a la avería. Trata de eliminar los efectos del fallo.

- Inicialización (*Restart*): Es una forma de recuperar el sistema. Dependiendo de la gravedad de la avería se intentará realizar de diversas formas:

    - "*Hot*": Si no ocurren daños, se continua con las tareas justo cuando se detectó el fallo.
    - "*Warm*": Sólo se puede continuar con algunos procesos.
    - "*Cold*": Se corresponde con una nueva carga completa de todo el sistema.

- Reparación (*Repair*): Se reemplazan los componentes que se ha diagnosticado que han fallado. El sistema se puede reparar *on-line* mediante la utilización de un repuesto, el enmascaramiento o la degradación funcional. También puede ser off-line, desconectando el sistema.

- Reintegración (*Reintegration*): El módulo reparado se reintegra en el sistema.

En la Figura 2.7 se puede ver la evolución de un sistema tolerante a fallos ante la presencia de fallos. El sistema deja de estar disponible desde el momento en que se produce el error hasta que se vuelve a reinicializar, en donde vuelve al estado de operación pero de forma degrada. Mientras tanto el componente defectuoso se repara y eventualmente se vuelve a incorporar en el sistema. En el gráfico se supone que la reintegración del componente defectuoso no se puede hacer en vivo, por lo que el sistema deja de estar disponible durante un lapso de tiempo.

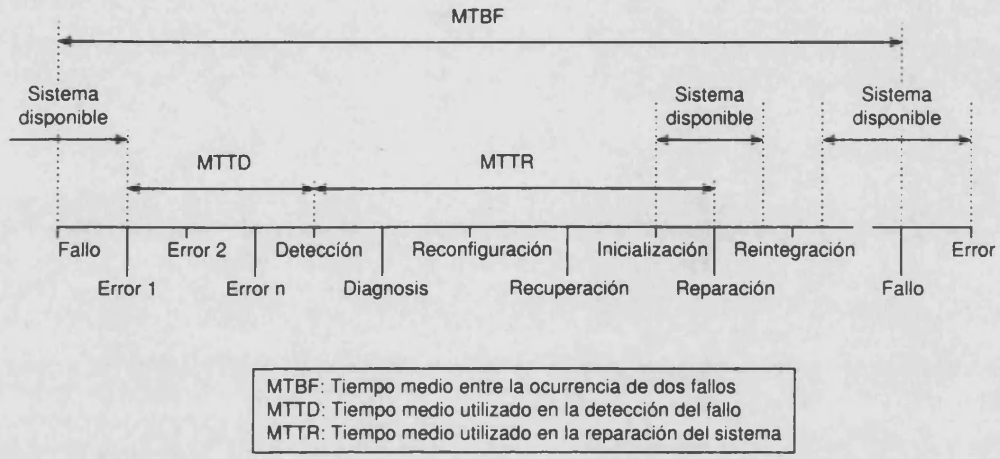


Figura 2.7: Evolución de un STF ante la presencia de fallos

7 Tolerancia a fallos en los sistemas multiprocesador

Desde la invención del primer ordenador, los avances tecnológicos han servido para incrementar las prestaciones de los nuevos sistemas. De la misma forma también se ha tratado de mejorar su topología, para conseguir con la misma tecnología un incremento en el rendimiento. Como consecuencia, en los últimos años ha emergido una nueva corriente basada en el procesamiento paralelo, donde se interconectan muchos nodos de proceso semejantes para forman un único sistema [98].

El problema que tienen los sistemas paralelos es que debido a su gran complejidad, por la existencia de tantos elementos replicados, tienen una tasa de fallos muy elevada [99], lo que los hace poco apropiados para las aplicaciones que requieren una alta fiabilidad o una alta disponibilidad. Es por tanto necesaria la consideración de una serie de medidas de tolerancia a fallos especiales, apropiadas para estos sistemas, que permitan la reconfiguración y degradación del sistema en el momento en que falla alguno de los nodos, y que como consecuencia aumenten el tiempo medio entre las averías.

La tolerancia a fallos se logra mediante la detección del error y la recuperación. Cuando un nodo se avería, el sistema debe reconfigurarse y redistribuir la carga del módulo averiado entre los demás procesadores. Este procedimiento se realiza en todos los niveles, desde los circuitos integrados hasta en la misma aplicación que ejecuta el usuario.

En los sistemas multiprocesador con memoria compartida, todos los elementos de proceso acceden a una zona de memoria común, por lo que se deben tratar con especial precaución los errores para evitar que un módulo averiado escriba en una posición de memoria global, corrompiendo su contenido. La solución pasa por comprobar que el módulo funciona correctamente antes de realizar la escritura. Las técnicas empleadas no difieren en gran medida de las utilizadas en los sistemas monoprocesador,

puesto que los principios son los mismos. Se basan en el uso de la redundancia estática en donde como mínimo existen 3 copias de cada procesador y se votan los resultados y en el uso de la redundancia dinámica en donde hay que detectar el procesador averiado y reemplazarlo. La técnica escogida dependerá de las necesidades de fiabilidad, de disponibilidad y de seguridad del sistema.

Normalmente los multiprocesadores utilizan redundancia dinámica puesto que es la más barata. En estos sistemas debe existir una mecanismo que indique la *detección* de un error en alguno de los módulos de proceso. En ese momento se debe activar un procedimiento de localización del fallo o de *diagnosis* para poder reemplazar el módulo con fallo por un repuesto mediante técnicas de *reconfiguración*. Finalmente se realiza la *recuperación* del error en donde el módulo de recambio, utilizando información que almacenó el módulo con fallo antes de averiarse, continúa con las tareas que éste estaba realizando.

7.1 Detección de fallos

Existen varias técnicas para detectar los fallos, dependiendo del nivel en donde se apliquen. En los sistemas de bajo coste, las transacciones críticas se duplican, bien utilizando redundancia espacial, bien redundancia temporal, para después comparar los resultados. Los sistemas de Stratus [100] y el sistema Intel 432 [101] implementan módulos de proceso con procesadores duplicados que comparan los resultados antes de enviarlos a otros pares de procesadores. Otra forma equivalente es la de dividir los P procesadores del sistema en P/2 pares y los M módulos de memoria en M/2 e implementar comparadores en cada módulo. Antes de realizar las operaciones se comprueba que las comparaciones son correctas. Si existen restricciones con respecto al coste, se puede utilizar redundancia temporal, donde se ejecutan varias veces las tareas en procesadores distintos y se comparan los resultados.

Otra forma de detectar los fallos es la utilización de programas de diagnosis para los fallos permanentes y el uso de técnicas de detección de fallos concurrentes como la utilización de códigos duplicados y el uso de métodos de autocomprobación [102].

7.2 Estrategias de recuperación

Ya que la mayoría de los fallos son de naturaleza transitoria o intermitente [120][121], una vez detectado el fallo, lo más fácil para recuperar el error sería el reintento de la operación. Sin embargo, hay sistemas que no se pueden permitir esta latencia o incluso el reintento de la operación no es suficiente para realizar el tratamiento del fallo, como es el caso de los fallos en las comunicaciones, y se debe recurrir a estrategias más complejas.

En los sistemas con memoria compartida, la estrategia más simple es la reinicialización del procesador con fallo y la repetición de las tareas que estaba realizando desde el comienzo (reinicialización global), sin embargo la pérdida de rendimiento es evidente. Lo que sería más interesante sería la continuación del trabajo justo desde el punto donde se detectó el error y utilizar algún otro mecanismo para asegurar la ejecución correcta del mismo (recuperación hacia delante). Si no es posible esta recuperación, sería incluso mucho más ventajoso continuar el trabajo no desde el principio, sino desde un punto intermedio en donde aún no se había producido el error (recuperación hacia atrás). Este mecanismo es el mismo que en los sistemas monoprocesador, la diferencia estriba en que en los multiprocesadores, como muchos elementos pueden acceder a las mismas variables en memoria, se tiene que implementar algún mecanismo que asegure que el estado es consistente y que la información puede ser utilizada para recomenzar de nuevo.

Recuperación hacia atrás (*backward recovery*)

El método de tolerancia a fallos más popular es el esquema que implementa puntos de recuperación (*checkpoints*). La técnica consiste en almacenar la información necesaria sobre el estado del procesador para asegurar que en el caso de que se produzca una avería en algún nodo, el trabajo puede ser echado hacia atrás hasta el lugar donde se estableció el último punto de recuperación y continuar como si no se hubiera producido ningún fallo. Normalmente el estado del procesador lo constituyen sus registros, el contador de programa, la cache y las zonas de memoria que han sido alteradas por el procesador desde que se estableció el último punto de recuperación. Esta técnica implica que los puntos de recuperación se deben de almacenar en una zona de memoria estable que no falla nunca.

Existen varias técnicas para el establecimiento de puntos de recuperación. Mediante el uso de la cache se establece un punto de recuperación cada vez que, debido a que se ha producido un miss, se tiene que llevar un bloque de la cache a memoria principal [13]. Para crear el punto de recuperación se guarda en memoria los registros del procesador y los bloques que se han modificado desde el establecimiento del último punto de recuperación. Para realizar la vuelta atrás, simplemente se invalidan los bloques modificados de la cache y los registros del procesador y se les restaura su contenido anterior usando los datos del punto de recuperación anterior.

El esquema anterior tiene el problema de que si la cache es de tamaño reducido, la frecuencia de establecimiento de puntos de recuperación es muy elevada [103] con lo que el rendimiento decrece. Otras alternativas se basan en el establecimiento de puntos de recuperación en memoria virtual que permiten distribuirlos de forma más adecuada en el tiempo [104][105].

Otro método que no necesita de hardware adicional para la recuperación es el llamado esquema por *pares de procesos* [106].Consiste en que dos procesadores ejecutan dos réplicas de un proceso. Una de ellas, la de respaldo, se mantiene actualizada ya que periódicamente recibe el estado del proceso activo. Los procesadores que ejecutan las copias tienen la característica de *parada después de la avería* (*fail-stop*) de forma que cuando ocurre una avería se detienen, impidiendo que se realicen operaciones incorrectas. Periódicamente cada procesador envía un mensaje de "*estoy-vivo*" [106] a los demás procesadores del sistema. Todos ellos monitorizan la llegada de estos mensajes. Si no llega un mensaje de un procesador en particular en un tiempo determinado, se supone que ese procesador está averiado y las copias de respaldo pasan a ejecutar los procesos activos que había en el procesador averiado. Ahora estas copias pasan a ser los procesos primarios.

Recuperación hacia delante (*fordward recovery*)

El sistema al detectar el fallo rechaza el estado actual erróneo y determina el estado correcto sin perder ni una sola operación. Las soluciones hardware utilizan la redundancia estática para enmascarar los posibles errores, o la redundancia híbrida, en donde los módulos tienen la característica de parada después de la avería (bien por duplicación, bien porque son autocomprobantes) y en el momento que se detecta que ha ocurrido una avería en alguno de los módulos se desconecta dicho módulo y el repuesto continúa con el servicio sin interrupciones ni vueltas atrás.

Las aproximaciones hardware que utilizan la redundancia dinámica resuelven la recuperación hacia adelante con la ayuda del establecimiento de puntos de comprobación. Por ejemplo, consideremos un sistema dual compuesto por dos módulos de proceso trabajando en paralelo. Periódicamente los módulos establecen puntos de recuperación y los van comparando para detectar si ha habido algún error. En el caso de que éste se produzca, en vez de realizar la vuelta atrás para repetir las operaciones desde el punto de comprobación anterior, lo que se hace es averiguar cuál de los dos módulos es el que ha fallado y se continúa trabajando con el módulo bueno. En [107] se discute un esquema en donde se utiliza, aparte del sistema dual otro módulo de repuesto. En este caso se establecen puntos de recuperación de forma periódica que además sirve para poder detectar errores. En el caso de que exista

alguna discrepancia en el punto de recuperación del sistema dual, se carga en el módulo de repuesto el punto de recuperación correcto anterior y se continua con el trabajo. En el establecimiento del siguiente punto de recuperación se sabrá cual de los dos módulos había fallado. Las ventajas de este método es que no se pierden prestaciones y que la degradación en el rendimiento de la E/S es mínima.

Las estrategias software se basan en el establecimiento de bloques de recuperación [108]. Este esquema trata de asegurar un objetivo T, llamado test de aceptación, mediante una serie de reintentos, primero con el bloque B1, luego con el B2 y así hasta terminar con todos los bloques de recuperación Si no se ha conseguido finalizar el test con éxito se produce la avería del sistema.

7.3 Reconfiguración en multiprocesadores

En el caso de que se produzca un fallo permanente, es necesario desconectar el módulo que ha fallado y reconfigurar el sistema, para que siga trabajando aunque con un nivel de prestaciones inferior. Si la topología del sistema está basada en la existencia de un único bus, o varios buses replicados, la degradación funcional del sistema se realiza con facilidad, teniendo simplemente que reconfigurar los árbitros para que trabajen con un módulo menos.

La mayoría de los multiprocesadores tolerantes a fallos utilizados en las aplicaciones transaccionales se basan en estos diseños: Tanden Non-Stop, Cyclone /R, Stratus XA/R, Sequoia S400.

8 Ejemplos de Sistemas Tolerantes a Fallos

La gran expansión que ha tenido el uso del ordenador en los últimos tiempos ha hecho que los sistemas tolerantes a fallos ya no se utilicen únicamente en las aplicaciones en donde las averías producen consecuencias catastróficas (gran conste económico o pérdida de vidas humanas) o en aplicaciones militares. Las técnicas de fiabilidad se incorporan cada vez más en aplicaciones generales, debido a que los ordenadores se sitúan en entornos mucho más ruidosos, son utilizados por personal no experto, se incrementa el coste de reparación frente al coste de producción y se fabrican máquinas mucho más complejas en donde al haber muchos más componentes aumenta la probabilidad de fallo.

En el momento de decidirse por utilizar un ordenador u otro, se debe ponderar cual va a ser el coste de un error (por ejemplo la parada del sistema o la realización de alguna operación incorrecta) frente al coste adicional del diseño de los mecanismos de tolerancia a fallos del sistema. Dependiendo de la aplicación, se tendrán que elegir un diseño particular.

Por este motivo es más significativo realizar una clasificación de los sistemas tolerantes a fallos en función de su aplicación, en vez de enumerar todas las características propias de la arquitectura de estos sistemas. La Tabla 2.2 muestra algunos ejemplos clasificados según las áreas de aplicación:

Sistemas comerciales de propósito general	Alta disponibilidad	Larga vida	Computaciones críticas
VAX 8600 IBM 3090	AT&T Tandem Stratus VAXft 310	Voyager Galileo	SIFT Space Shuttle

Tabla 2.2: Clasificación de los sistemas dependiendo de la aplicación

A continuación veremos algunos ejemplos de sistemas tolerantes a fallos, todos ellos basados en una topología de bus que sirve para interconectar a todos los módulos de proceso con los demás elementos del sistema.

8.1 Tandem NonStop Cyclone

Esta máquina [109] es un multiprocesador tolerante a fallos con memoria distribuida que tiene como objetivo el eliminar cualquier punto único de fallos mediante la duplicación del hardware. Utiliza el sistema operativo GUARDIAN 90 que mantiene copias de respaldo de todos los procesos de usuario en otros procesadores distintos y los elementos principales están diseñados para que se paren en el momento en que se detecta el error.

El sistema utiliza secciones múltiples separadas físicamente y cada sección, a su vez, puede estar compuesta de hasta cuatro procesadores conectados entre sí a través de un bus tolerante a fallos llamado Dynabus (ver Figura 2.8). Las secciones se comunican entre sí a través de otro bus similar llamado Dynabus+ que utiliza fibra óptica para tener un ancho de banda similar al Dynabus, pero en distancias mucho mayores (hasta 50 metros). Cada procesador puede manejar uno o dos sistemas de E/S. Cada uno de estos sistemas lleva conectados dos canales para poder controlar múltiples dispositivos, todos conectados con controladores de doble puerto y discos espejo.

El sistema trata de tener una alta disponibilidad, por lo que se permite la diagnosis en línea y la operación continuada en el caso de que falle un procesador o uno de los módulos de memoria SEC-DED* que posee. Los componentes están diseñados de forma que si fallan se pueden reemplazar en vivo sin la necesidad de parar el sistema.

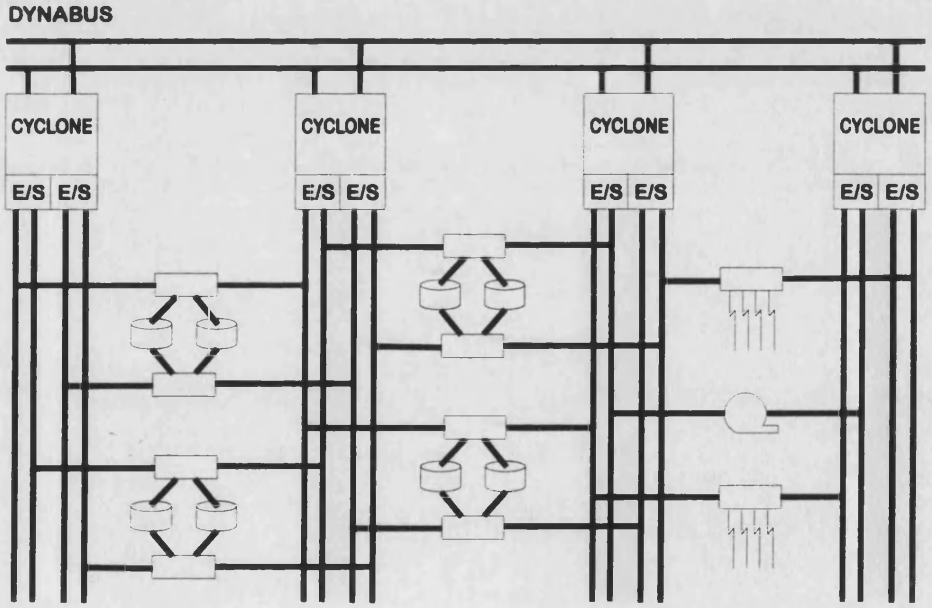


Figura 2.8: Ejemplo de una sección del Cyclone

* Memoria que detecta los errores dobles y es capaz de corregir los errores de un bit

El Sistema Operativo detecta los fallos utilizando comprobaciones de la consistencia y mensajes de "*estoy vivo*" que se envían cada segundo sobre todos los buses y a todos los procesadores del sistema para que estos puedan detectar los errores.

8.2 Sistema Stratus XA/R Serie 300

El sistema [111] es un multiprocesador tolerante a fallos con memoria compartida adecuado para las aplicaciones transaccionales. Está compuesto por uno o varios módulos (hasta 32) conectados a través de un bus redundante el llamado StrataLINK. Cada módulo funciona como una máquina independiente ya que consta de varias placas: de 1 a 6 placas con procesadores duplicados, memoria (hasta 512 Mb) y varias placas de E/S duplicadas. Todas las placas se comunican entre sí dentro del módulo a través de otro bus tolerante a fallos propiedad de Stratus (Ver Figura 2.9). El sistema operativo se puede elegir entre un sistema compatible con Unix SVR4 FTX o el sistema VOS de Stratus.

La tolerancia a fallos se logra por el uso de módulos de proceso duales que funcionan sincronizados. En el momento en que el comparador detecta una discrepancia entre los procesadores del módulo, un interruptor desactiva sus salidas dejándolos inhabilitados. Cada módulo de proceso tiene un compañero de respaldo (también dual) que trabaja utilizando dos posibles estrategias. Puede estar sincronizado con él realizando las mismas tareas al mismo tiempo, con lo que cuando se detecta el error simplemente el interruptor activa las salidas del módulo de respaldo, sin tener necesidad de esperar y sin reducir las prestaciones. En la otra estrategia el módulo de respaldo funciona como un par lógico independiente. Cuando el módulo principal falla, el sistema operativo lo sincroniza y queda encargado de retomar las tareas del módulo averiado, con la consiguiente pérdida de prestaciones.

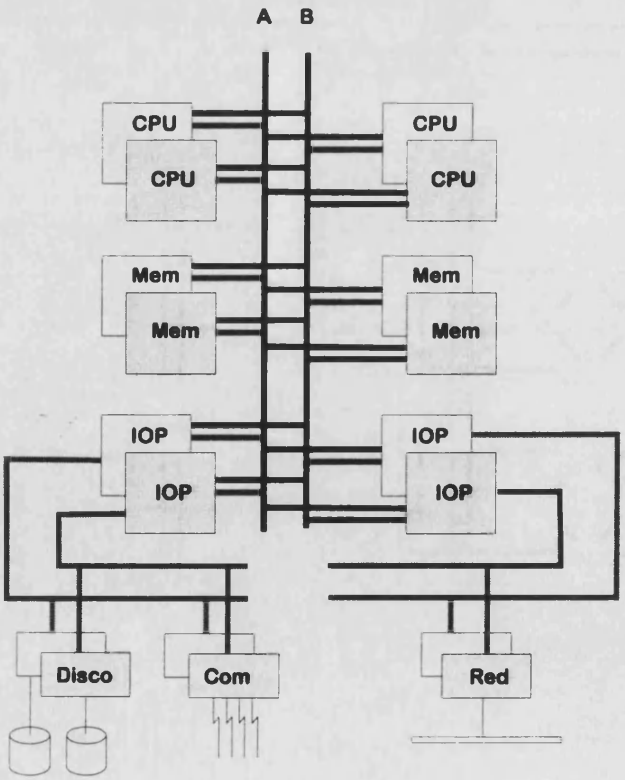


Figura 2.9: Módulo del sistema Stratus

La memoria, aparte de estar duplicada, esta protegida mediante códigos de corrección de errores y observadores espía que se encargan de comprobar posiciones de memoria para descubrir errores latentes y corregirlos. Los módulos de E/S también son duales. Los buses están duplicados y protegidos por paridad. Cuando se detecta un error, el bus se deshabilita, pero como todos los módulos están conectados a los dos buses esto no supone que algún módulo deje de funcionar.

Debido a que los mecanismos de tolerancia a fallos están implementados por hardware, el sistema operativo no necesita implementar otras técnicas de tolerancia a fallos como son los puntos de comprobación, la duplicación de procesos y el envío de mensajes de "estoy vivo". Sin embargo existe un procedimiento de mantenimiento y de diagnóstico encargado de diagnosticar la causa de los fallos y de notificarla. En el momento en que el módulo falla, se inicia un autotest. Si termina de forma satisfactoria el módulo se devuelve a su trabajo, en caso contrario se supone que el fallo es permanente y se marca para que sea reemplazado. Este procedimiento lleva cuenta del tiempo medio entre fallos del módulo. Si el módulo tiene varios fallos sucesivos en un tiempo inferior al tiempo medio estimado, se supone que el fallo es intermitente y también se marca el módulo para reemplazarlo. También está implementado un procedimiento de red de servicio remoto que avisa comunicándose con el centro de control de Stratus que existe un módulo que hay que reemplazar.

8.3 Sistema Sequoia Serie 400

Este sistema [31] utiliza múltiples módulos dúplex que son autocomprobantes y fabricados en base a componentes estándar, para formar un multiprocesador tolerante a fallos con memoria compartida y orientado a las aplicaciones transaccionales[110]. Un sistema completo está formado por 32 elementos de proceso, 16 elementos de memoria de 256 Mb cada uno y 30 elementos de E/S.

Cada módulo de proceso está dotado con dos procesadores que trabajan sincronizados y votadores duplicados. La memoria también está dividida en dos arrays a los que se accede de forma independiente e incluye la implementación hardware de operaciones *test-and-set*. La información está protegida con códigos SEC-DED. Las placas de E/S tienen dos interfaces duplicados, el primero se conecta al bus de memoria local y el segundo al Multibus y se utilizan como almacenamiento temporal de los datos que van desde los elementos de proceso en la cabina del sistema al adaptador de Multibus que se encuentra en la cabina de los periféricos. Todos los periféricos son de doble puerto y se conectan a diferentes elementos de E/S para evitar que les afecte el fallo de uno de ellos.

Los mecanismos de detección de fallos que implementa este sistema incluyen el uso de códigos de control de errores, la realización de comparaciones en las operaciones redundantes y la monitorización de los protocolos. Cada elemento de proceso dispone de un bloque de estado de 128 bytes que describe su estado y que actualiza cada 100 ms. Los demás elementos de proceso lo verifican periódicamente para ver si el módulo funciona correctamente.

Estos modelos ejecutan el sistema operativo TOPIX, que es propiedad de Sequoia, que suministra soporte para el multiprocesado simétrico, el equilibrado dinámico de la carga, sincronización mediante operaciones de *test-and-set*, el manejo de los errores software y la detección y recuperación de fallos. La consistencia del sistema se mantiene en todo momento mediante el establecimiento de puntos de recuperación, primero en la memoria de almacenamiento de seguridad y luego en la principal. Los errores del software se detectan mediante la incorporación de aserciones en ciertas instrucciones. Cuando se ejecuta la instrucción se verifica que el resultado cumple con las suposiciones esperadas.

8.4 Sistema FASST

El objetivo principal de la investigación ha sido la evaluación de la garantía de funcionamiento del sistema FASST. Este sistema es un multiprocesador que logra la tolerancia a fallos mediante la combinación de varios requisitos indispensables:

1. Incorporación de módulos de proceso denominados de "parada tras la avería" ó *fail-stop*, o algunas veces también llamados de "parada y silencio" ó *fail-silent*. De esta forma se garantiza que los módulos averiados no van a contaminar el sistema con datos erróneos, puesto que en el momento en que se detecta el error se paran.

2. Incorporación de un sistema de almacenamiento estable (original de INRIA/IRISA) donde ir guardando periódicamente el estado del sistema para poder establecer puntos de recuperación en caso de fallo.

Los objetivos del proyecto fueron los siguientes:

- Desarrollar una arquitectura abierta al incorporar tantos componentes estándar como fuera posible. El sistema debe de ser fácilmente escalable como se define en [10].

- Investigación de las metodologías apropiadas y de las técnicas de modelado [115].

- Construcción de una serie de bloques funcionales para sistemas tolerantes a fallos basados en un soporte hardware a bajo nivel, almacenamiento estable, software para el sistema operativo y soporte para las aplicaciones.

- Demostración de las aplicaciones en sistemas prototipo.

Teniendo en cuenta estas características, se decidió el diseño de la arquitectura tolerante a fallos FASST, que incorporaba los siguientes bloques funcionales:

- **La DPU**: Es una unidad de procesamiento basada en el R4000[†], con mecanismos de comprobación del hardware de bajo nivel y una interfaz para Futurebus+. La única característica que debe tener es que debe ser *fail-stop* o *fail-silent* [14] [15] [16], para no propagar los errores al sistema en caso de fallo.

- **La Memoria estable (STM)**: Es una memoria redundante de acceso aleatorio que está dividida en bloques de tamaño fijo [9][113]. Está compuesta por una serie de elementos, cada uno de ellos formado por dos bancos de memoria manejados por un controlador inteligente. El almacenamiento de un bloque se produce en los dos bancos, por lo que se está duplicando la información para poder detectar los errores.

  La memoria estable realiza las operaciones de forma atómica. En las escrituras, primero se escribe en un banco todos los bloques de la transacción y cuando se termina, internamente la STM realiza la copia al otro banco. La transacción no se habrá terminado con éxito hasta que no se han almacenado satisfactoriamente todos los bloques que pertenecen a la transacción en ambos bancos. Si se produce un error en la escritura en el primer banco, la STM recuperará los datos que había antes leyendo la información del segundo banco.

[†] Debido a los problemas de disponibilidad de software y de falta de información del R4000, se sustituyeron los procesadores por los Intel 486DX2.

La memoria también soporta la realización de forma atómica de un conjunto de transacciones dependientes. Es decir, que el conjunto de transacciones no se dará por terminado hasta que no se termine de realizar la última de ellas. En caso de error, se recuperará el estado de la memoria anterior a la escritura del primer bloque de la primera transacción.

La STM se utiliza tanto para almacenar los puntos de comprobación de los procesos (*checkpoints*) [12] cuando es necesario [13] como para mantener la consistencia de los estados de los procesos con respecto a la memoria principal y permitir la recuperación después del fallo.

- **El Disco estable (SD)**: El disco estable [114] es una unidad SCSI que implementa una organización de ficheros especial (*logging file system*) [35] que maximiza el rendimiento del disco. En las escrituras de datos, primero se realiza la operación en un espacio de almacenamiento intermedio y luego se escriben en el disco tratando de minimizar el tiempo de búsqueda. Para dotarlo de las características de tolerancia a fallos necesarias se ha basado su construcción en una tecnología RAID [11]. Está implementado mediante una serie de discos de tamaño pequeño, que trabajan todos en paralelo, que permite el aumento del rendimiento en las lecturas en los sistemas con alto grado de fragmentación.

- **Futurebus+**: El bus estándar más apropiado para la aplicación tolerante a fallos es el Futurebus+, puesto que soporta la inserción y extracción de módulos en vivo, tiene suficientes señales para ser definidas por el usuario (en nuestro caso servirán para la transmisión de identificación de módulo en las tareas de dependencia que van a la memoria estable), y está lo bastante extendido como para suponer que lo van a soportar los principales fabricantes de ordenadores. (Ver Capítulo 4).

- **Microkernel tolerante a fallos**: Un sistema operativo basado en Mach que ofrece al usuario una interfaz libre de averías mediante el uso de un micronúcleo y un hardware tolerante a fallos. La tolerancia a fallos se consigue principalmente mediante el establecimiento de puntos de recuperación utilizando la memoria estable y la recuperación de la E/S [116]. También se implementó un soporte ejecutivo en tiempo real para ejecutar aplicaciones [117].

- **Demostradores**: Dos aplicaciones de demostración para comprobar el hardware a bajo nivel y el soporte para la tolerancia a fallos a nivel de aplicación: una aplicación transaccional y una aplicación de tráfico.

Para las distintas aplicaciones que van a funcionar dentro de esta plataforma se han implementado varios sistemas funcionales: un sistema ejecutivo en tiempo real, un sistema de ficheros y un sistema transaccional. En la Figura 2.10 se puede ver un esquema de la plataforma FASST.

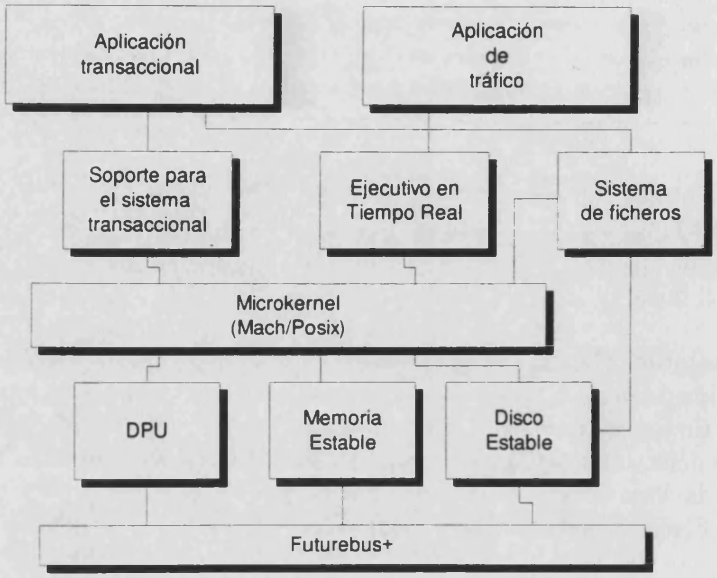


Figura 2.10: Bloques funcionales en FASST

Debido a la necesidad de validar todos los componentes del Proyecto desde las primeras fases del mismo, no se podía esperar a disponer del prototipo, por lo que se eligió la maquina Corollary con el bus Cbus como sistema alternativo donde demostrar todas las características del sistema. La máquina de Corollary es un multiprocesador con memoria compartida basado en unidades de proceso con el procesador Intel 486 y una adaptación del microkernel Mach. Tanto la memoria estable como el disco estable se han integrado en esta máquina, implementándose funciones de acceso de bajo nivel necesarias para la aplicación transaccional.

La aplicación de tráfico consistió en el desarrollo de unos drivers tolerantes a fallos y de un hardware estable especial, que permite el establecimiento de un sistema de comunicación fiable entre una estación de trabajo UNIX y un sistema de control de tráfico.

La muestra un diagrama de bloques de la arquitectura FASST tal y como se especificó inicialmente. Destacan el bus duplicado y la existencia de un puente encargado de conectar ambos buses. Tanto la memoria estable como el disco estable disponen de interfaces duplicados para evitar los posibles puntos únicos de fallo del sistema.

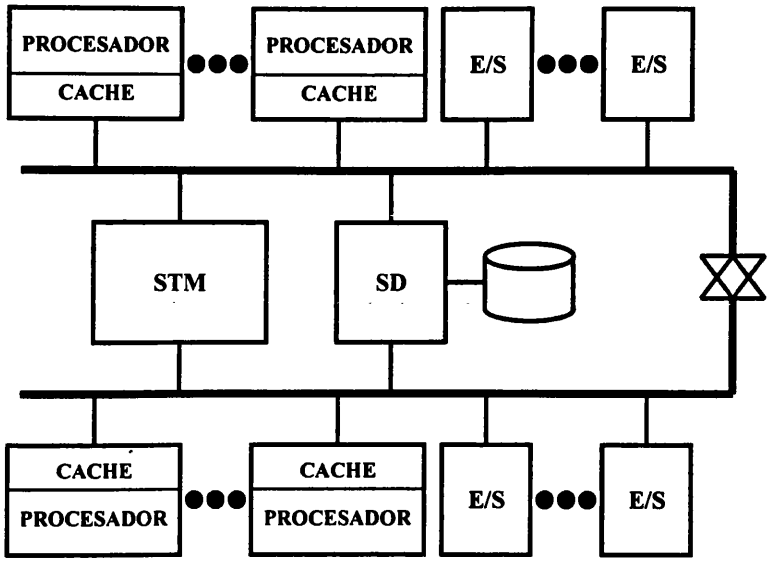


Figura 2.11: Diagrama de la arquitectura FASST

9 Conclusiones

En este capítulo se han introducido los conceptos fundamentales sobre fiabilidad y tolerancia a fallos y se ha explicado la terminología básica para poder comprender todos los aspectos desarrollados en la presente memoria y poder identificar las cosas con mayor rigurosidad.

Se ha incluido una sección para hacer mayor hincapié en los sistemas multiprocesadores con alta garantía de funcionamiento. En ella se han explicado las técnicas más usuales utilizadas en los sistemas con memoria compartida.

Al término del capítulo se repasa la arquitectura de algunos sistemas comerciales y se presentan las características del sistema FASST, necesarias para poder entender la metodología utilizada en el diseño de sus módulos de proceso.

Capítulo 3

Diseño de los módulos de proceso

1 Introducción

En el momento de diseñar un sistema, es primordial lograr un alto índice de rendimiento para que pueda realizar las operaciones a la mayor velocidad posible. El índice de crecimiento de las necesidades de prestaciones superó en gran medida al avance tecnológico, por lo que de tener un sistema que ejecutaba secuencialmente operaciones sobre datos pasaron a implementarse sistemas multiprocesador en donde varios módulos cooperaban en paralelo sobre múltiples datos para realizar una tarea en común. En los sistemas con memoria compartida, o también llamados fuertemente acoplados, todos los módulos se comunican entre sí a través de un mismo bus accediendo a una zona de memoria común. La topología de estos sistemas consiste en uno o varios módulos de proceso, cada uno dispuesto físicamente en una placa de circuito integrado distinta, uno o varios módulos de memoria, que normalmente no tienen capacidad para ejecutar operaciones y una serie de módulos que se encargan de realizar las operaciones de E/S. Cada uno de estos módulos ocupa una ranura distinta en el bus del sistema.

Si se desea construir un sistema tolerante a fallos basado en esta topología (como es el caso del sistema FASST descrito en el Capítulo anterior) la estrategia que se utiliza para obtener cierta confianza en su funcionamiento es la de tratar de detectar los fallos de los módulos de proceso lo antes posible y darlos de baja para que no corrompan la integridad de los datos de la memoria. Además para evitar que es sistema deje de funcionar porque se avería un módulo será necesaria la incorporación de una serie de métodos que permitan la reconfiguración del sistema, para que siga funcionando aunque sea en un modo degradado con menos prestaciones, o incluso en un modo no tolerante a fallos. Si el sistema no es capaz de reconfigurarse, cuantos más módulos tenga, menor será el tiempo que está funcionando, puesto que aumenta la probabilidad de que falle alguno de sus elementos.

A continuación se describen las características de los módulos de proceso del sistema FASST, haciendo un mayor hincapié precisamente en los métodos hardware y software que dispone el módulo para detectar los errores. Se incluyen también los aspectos relativos a la implementación de las placas y la puesta de funcionamiento del prototipo del sistema.

2 Descripción de los módulos de proceso

El prototipo del sistema FASST cuenta actualmente con dos módulos de proceso idénticos. Cada uno se ha implementado en una placa estándar de Futurebus+ y cuentan con una interfaz al bus basada en los integrados de Texas Instruments de la familia de Futurebus+. A lo largo de la memoria se va a utilizar el término DPU's (*Data Processign Units*) [28] para referenciar a estos módulos. El acrónimo proviene de las palabras Unidades de Procesamiento de Datos, si bien, particularmente prefiero denominarlas Unidades de Procesamiento Duales, para resaltar su topología singular.

Todo el diseño ha estado dirigido por dos objetivos fundamentales, aparte, claro está, de las consabidas necesidades de aumento de prestaciones y de reducción de costes:

- Realización de módulos *fail-stop* (parada después de la avería): En el momento en que los mecanismos de detección de fallos indiquen que un fallo ha tenido lugar (normalmente serán fallos permanentes o intermitentes), el módulo tendrá que detenerse para evitar la contaminación de otras partes del sistema con datos corruptos.
- Utilización de componentes estándar para evitar los problemas de la no disponibilidad de los componentes al tener que implementar la placa o al tener que reparar averías y abaratar costes. También se ha hecho uso intensivo de lógica reprogramable que disminuye la producción de errores, disminuye el tiempo al mercado y, puesto que se trata de un prototipo, permite el rediseño para la corrección de errores.

2.1 Mecanismos de tolerancia a fallos

En los módulos de proceso es fundamental conseguir latencias de detección de errores mínimas. Se ha elegido el sistema más simple y rápido en detectar los errores que se basa en la implementación de un sistema dual, dejando de lado los sistemas con redundancia dinámica basados en la autocomprobación de los componentes que requieren mayores tiempos de detección.

La placa incorpora dos procesadores Intel 486DX2-66 que funcionan al unísono (modo *lock-step*). La estrategia consiste en que ambos procesadores van ejecutando simultáneamente las mismas instrucciones con los mismos datos; para ello los dos procesadores comparten la misma señal de reloj y las mismas señales de entrada, tanto las de control como las de datos. En cada ciclo de reloj, se comparan todas las señales de salida de los procesadores y si se detecta alguna discrepancia, inequívocamente significa que alguno de ellos ha fallado, lo que produce la activación de una señal de error.

Mediante esta configuración no es posible discernir cual de los procesadores ha fallado (a no ser que se realice una autocomprobación del mismo), por lo que el módulo no podrá seguir funcionando ni siquiera en modo degradado y tendrá que aislarse del sistema (el proceso de aislamiento, como se verá más adelante, es programable y consiste básicamente en la desconexión de los drivers del bus). La tolerancia a fallos se basa en la degradación del sistema por lo que debe existir una migración de las tareas del módulo averiado hacia otro módulo. Para ello es necesario el establecimiento de puntos de recuperación, y es conveniente la implementación de un sistema de acceso a los recursos internos del módulo que ha fallado para poder realizar una diagnosis fuera de línea o ejecutar tests.

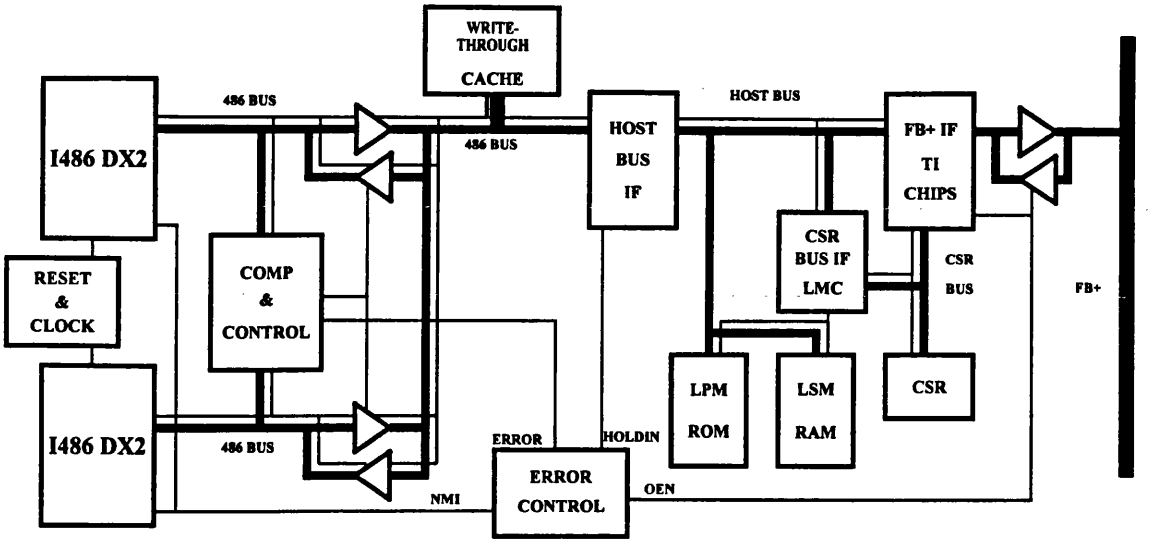


Figura 3.1: Diagrama de bloques de la DPU

El módulo incluye dos pares de comparadores que se encargan de comprobar el correcto funcionamiento de los procesadores. Dos comparadores idénticos funcionalmente verifican las líneas de dirección y las líneas de control y otros dos, también idénticos, se encargan de las líneas de datos. El objeto de la duplicación ha sido evitar fallos en modo común que afecten por igual a todos los integrados, y por ello se ha incorporado en su diseño la técnica de la diversificación funcional. Esta técnica consiste en utilizar lógica complementaria para implementar circuitos que realizan la misma función, con lo que se aumenta la cobertura de detección de errores de todo el conjunto. La señal de error de los comparadores, en consecuencia, está codificada en un código redundante 1-de-2 que permite diferenciar entre tres posibles estados: funcionamiento correcto o incorrecto de alguno de los procesadores, siempre que la salida pertenezca al código (10 ó 01), o fallo en el propio comparador cuando la salida no pertenece al código (00 ó 11). Las salidas de los comparadores se dirigen a un votador central que es el que verdaderamente decide si se ha producido un error o no. La fiabilidad del sistema depende del correcto funcionamiento del votador, por lo que se le ha prestado especial atención tanto en su diseño como en su localización física dentro del circuito impreso.

Los comparadores están monitorizando continuamente la actividad del bus y por lo tanto introducen retardos. Para minimizar la pérdida de prestaciones del módulo como consecuencia de su topología dual, los dispositivos se han diseñado de forma que para una tecnología en concreto se alcanza la frecuencia máxima de funcionamiento. La implementación se ha realizado utilizando dispositivos programables (EPLD's de la casa Altera de la serie 7000) y con el objetivo de evitar los retardos internos de las señales dentro del integrado, se han aplicado técnicas de segmentación que evitan el uso de realimentaciones internas en el camino de los datos dentro del dispositivo.

El diseño se ha realizado utilizando un lenguaje de descripción de hardware (Altera AHDL) debido primordialmente a la estructura repetitiva de las funciones a implementar. Aunque una simple comparación parece una función fácil, en el diseño hay que contar que no siempre los comparadores deben estar activos, por lo que se debe de añadir mucha lógica adicional para evitar la detección de errores no existentes. Para evitar bucles internos en las señales se han intentado sacar el mayor rendimiento a cada bloque lógico que constituye la EPLD. Se han utilizado dos técnicas para evitar que se detecte un error de comparación cuando las señales no tienen significado dentro de la transacción del bus:

1. Deshabilitación de la comparación
2. Enmascaramiento de la señal de error

Como resultado se han obtenido unos comparadores con un retardo de propagación que viene dado por el retardo mínimo que sufre una señal desde un pin de entrada a un pin de salida pasando por un registro en la EPLD.

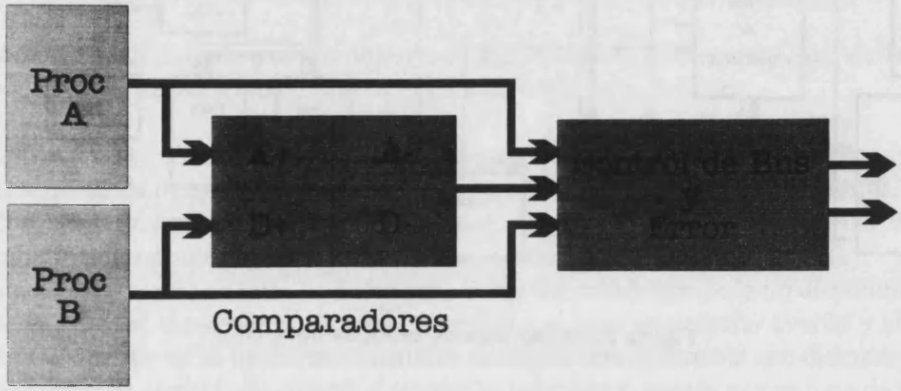


Figura 3.2: Estructura de los comparadores

El votador final, aparte de decidir si ha habido un error, también controla los drivers que conectan las señales de los procesadores con la memoria. En un sistema dual, como los dos procesadores están, conectados al mismo bus, sólo uno de ellos, el maestro, deberá de activar sus señales de salida para evitar contenciones. El votador se encarga de duplicar las señales de entrada para que lleguen a los dos procesadores y de multiplexar las señales de salida. En el módulo existe un registro programable que decide cuál de los dos procesadores es el activo y cuál es el repuesto. Las funciones principales, a parte de la de generar la señal de error son:

- Deshabilitar los transceivers cuando el procesador pone el bus en alta impedancia
- Cambiar la dirección de los transceivers de datos dependiendo de si el ciclo es de lectura o escritura
- Cambiar la dirección de los transceivers de direcciones en los ciclos de invalidación de las líneas de cache

Para verificar el correcto funcionamiento de los comparadores, se realizó una simulación utilizando el simulador digital *QuickSim* de *Mentor Graphics*. Este simulador es muy potente puesto que utilizando librerías de la casa *Logic Modeling* permite incluso implementar un programa en la memoria que quieres simular que el procesador ejecuta. Se diseñó un sistema virtual con los dos procesadores, la memoria, el conjunto de comparadores y el votador. Al ritmo de la señal de reloj, el procesador genera las señales apropiadas de los ciclos de bus a medida que lee las instrucciones de la memoria. Estas señales excitan las EPLD's, con lo que se puede comprobar si el diseño funciona correctamente. En [17][29] se puede ver con más detalle una descripción del diseño de los comparadores.

2.2 Jerarquía de buses

Por el hecho de usar el FB+ como la base del sistema y de haber elegido el conjunto de integrados de Texas Instruments de FB+ para realizar la interfaz con el módulo [30], debemos definir una jerarquía interna de buses especial para que los módulos sean compatibles con los numerosos estándares que propone FB+.

El host bus

Conecta a los procesadores y a la memoria local con el FB+. Es un bus síncrono de 32 bits de datos que tiene 32 ó 36 líneas de direcciones. Este bus acepta múltiples tipos de transacciones, con la particularidad de que el tipo de transacción puede ser negociado dinámicamente entre el maestro y el esclavo. Como características principales destacan la posibilidad de realizar transacciones divididas para aumentar el rendimiento, la posibilidad de agrupar múltiples transacciones al Host Bus en una única transacción de FB+ y la existencia de una línea de bloqueo para realizar transacciones indivisibles.

Este interface ha sido implementado por medio de una EPLD, que realiza las siguientes funciones:

- Aislar al host bus de los buses de los procesadores. Cuando se detecta un fallo se le piden de forma indefinida los buses a los procesadores, forzando a que los dos 486 se queden en estado de HOLD.

- Traduce los ciclos de lectura y escritura del 486 en las correspondientes transacciones múltiples no-cacheables que define el estándar de Futurebus+.

- Maneja los ciclos cacheables del procesador generando transacciones cacheables en ráfaga. Para evitar que el FB+ intente utilizar una transacción con los datos que se encuentran en la cache, si se produce un acierto en el bloque, que se detecta por la activación de la señal *MATCH* del controlador de caches de Headland HT44, se activa la señal *IGNORE* con lo que FB+ se desentiende del manejo de dicha transacción.

- Realiza los ciclos de tratamiento de interrupción y los de invalidación de cache.

El bus del CSR

El CSR (*Configuration and Status Registers*) hace referencia a un conjunto de registros de configuración y de estado que define el estándar IEEE P1212 y que además el FB+ los incluye en su especificación.

Este bus se utiliza para acceder a los dispositivos de E/S o controladores que posee el módulo. Es un bus de 8 bit de datos, aunque los accesos se realizan siempre desde el procesador con transacciones de 32 bits. Si se accede a los registros que define FB+ en el CSR, o a la ROM, una transacción de 32 bits se traduce en cuatro accesos a posiciones consecutivas. Los datos se almacenan en registros intermedios y se devuelven como una sola palabra de 32 bits. El resto de los dispositivos son de 8 bits, pero están alineados en direcciones de doble palabra para facilitar el diseño de la interface (los dos últimos bits de la dirección son 00). En este caso cuando se realiza el acceso, como éste es de 32 bits, los datos ocupan el byte menos significativo y en los otros tres bytes se devuelven ceros.

El diseño lógico de la interfaz se divide en dos unidades funcionales: el controlador de memoria local (LMC) que maneja los accesos a la memoria local y el controlador del CSR que maneja el acceso a los dispositivos del CSR y decodifica las direcciones de los puertos de E/S.

Las características más sobresalientes que se implementan en la interface son:

- Generación de las señales de selección y de reconocimiento de los distintos dispositivos localizados en el CSR. Una de las características principales del Futurebus+ es que permite la reconfiguración dinámica de su espacio de direccionamiento. Para realizar la decodificación de las líneas de dirección es necesario utilizar las señales de selección de

memoria (MS[1..0]) que proporciona uno de los integrados de la familia de Futurebus+ de Texas Instruments, el TFB2022, que nos indica a que espacio se está accediendo en función del contenido de sus registros de dirección. Se definen cuatro espacios distintos:

- ◊ Espacio del CSR local
- ◊ Espacio de las unidades locales extendidas, usado para la memoria local privada
- ◊ Memoria local, usado para la memoria compartida del sistema
- ◊ Espacio del FB+

- Generación de las direcciones bajas para el acceso a byte (CA0 y CA1) y multiplexación del bus de datos de 8 <> 32.

- Generación de la paridad para todos los dispositivos conectados al CSR, a excepción de los integrados de Texas Instruments que ya la proporcionan.

- Control de los errores de direccionamiento: Cuando se detecta un error se termina inmediatamente la transacción en progreso y si es un error severo en donde se produce un acceso con datos incorrectos (combinación inválida del tamaño de la transacción y de los bits menos significativos de la dirección) se genera una interrupción de nivel 5. Se definen varios tipos de errores:

  - ◊ Escrituras en la memoria de sólo lectura
  - ◊ Lecturas al registro de control
  - ◊ Acceso a direcciones no definidas en el espacio de direccionamiento
  - ◊ Acceso a direcciones no alineadas a 32 bits

- Control de acceso a la memoria EPROM del sistema: Cuando arranca el sistema, una de las primeras tareas que el monarca debe realizar es la configuración del mapa de memoria de Futurebus+ escribiendo en los registros del CSR. Mientras tanto como no se han definido los espacios de direccionamiento del módulo, no se pueden utilizar las señales de selección de memoria MS[1..0] generadas por los integrados del FB+.

Al bus del CSR se encuentran conectados varios dispositivos, aparte de los registros que implementa la unidad de procesamiento de datos (TFB2002 DPU) y el controlador de arbitraje (TFB2010 ABC) definidos en el estándar de Futurebus+:

- La ROM de características, que guarda los valores por defecto que se van a cargar en la inicialización en los CSRs, además de información de identificación del módulo.

- Un reloj en tiempo real con RAM no volátil (DS1397)

- Un controlador serie dual (82510)

- Un registro latch (74BCT8373) que sirve para controlar el estado de apagado o encendido de los 3 leds definidos por el FB+ (Ejecución/Fallo/Intercambio), para decidir que procesador es el procesador activo o el de repuesto y para configurar las acciones a realizar en los drivers de Futurebus+ cuando se detecta un error.

- Un manejador de interrupciones, el 82C59A.

La implementación física de la interfaz a este bus se ha hecho utilizando dos EPLD's, aunque por su complejidad podía haberse utilizado un único dispositivo. El motivo fue que en cada EPLD sólo

existen dos líneas para controlar la activación de sus señales triestado y se tenían tres grupos funcionales distintos: los datos, las líneas de control y las señales de reconocimiento de datos*.

El bus Futurebus+

Se ha elegido al bus FB+ como el bus del sistema que conecta a todos los módulos. Para su implementación, como ya se ha comentado previamente, se han utilizado los integrados de Texas Instruments que implementan una configuración de tipo B (*Profile B* según el estándar de FB+). Presentan el inconveniente de que únicamente soportan un protocolo de arbitraje distribuido. A continuación se dan las particularidades de estos integrados:

- El controlador de E/S TFB2002 IOC se encarga de implementar la interfaz entre el FB+ y el Host Bus (HB). Debido a ello, es el encargado de realizar las peticiones de bus al árbitro.
- La unidad de datos TFB2022 DPU realiza la decodificación de las direcciones y la transferencia de datos entre los buses.
- El árbitro TFB2010 ABC se encarga de implementar el protocolo de arbitraje distribuido, del manejo de los mensajes de arbitraje y del manejo de las interrupciones de FB+.

El FB+ utiliza lógica BTL (*Bus Transceiver Logic*) para transmitir las señales por el bus. Incorpora por tanto una serie de transceivers que se encargan de realizar la traducción de señales BTL a TTL y viceversa. Estos transceiver están preparados para la inserción y el reemplazamiento de módulos en vivo, aprovechando la forma especial del conector de FB+. Esta característica es de vital importancia para asegurar una alta disponibilidad del sistema.

El bus de comprobación del sistema (JTAG)

Con el objetivo de facilitar la tediosa y complicada tarea de depuración de los módulos de proceso, se han incluido dos facilidades para permitir una depuración fuera de línea de los dispositivos y del circuito impreso y una depuración en línea mediante la incorporación de un monitor hardware.

Para aumentar al máximo la testabilidad del sistema, cuando ha sido posible se han utilizado integrados compatibles con el estándar IEEE 1149.1 de comprobación periférica de dispositivos (*Boundary Scan*) y se han conectado entre si mediante un bus denominado bus del JTAG. Mediante esta configuración y a través de una conexión muy sencilla con únicamente 4 señales se puede acceder a cualquier dispositivo conectado en la cadena para localizar errores, tanto a nivel de conexión al sistema - circuito impreso, como a nivel interno. Los dispositivos compatibles con este estándar incorporan una serie de registros internos de instrucciones y de datos que permiten la ejecución de una serie de rutinas internas de test para poder comprobar su correcto funcionamiento.

Dentro de este nivel cabe destacar la inclusión en el módulo de proceso de un monitor de bus digital controlado a través de JTAG. El monitor se ha colocado en el Host Bus para espiar la actividad de las señales de direcciones, datos y control. Funciona de la misma forma que lo hace un analizador lógico, permitiendo la incorporación de relojes internos, disparos programables y con una memoria extensa para poder capturar cualquier evento. A pesar de que se pierde en flexibilidad, tiene la ventaja de que evita la inclusión de las puntas de prueba necesarias en cualquier analizador lógico convencional. Este aspecto hay que tenerlo muy en cuenta puesto que cada día se van haciendo integrados de montaje

* Actualmente los dispositivos de Altera ya incluyen esta característica.

superficial más complejos, con mayor número de patas y de tamaño más reducido, lo que hace prácticamente imposible buscar algún lugar en el circuito impreso donde conectar la punta de prueba en la señal deseada. Se han utilizado cinco integrados de Texas Instruments SN74ACT8994, que se pueden programar para capturar los datos cuando se detecta algún tipo de evento en el bus, guardarlos en una memoria RAM interna de 1024 bytes y utilizar una fuente de reloj interna o externa para sincronizar todo el proceso.

Los dos buses de JTAG se utilizarán de forma diferente según el estado de depuración del sistema. El primero se pretende que se utilice en los primeros estados de desarrollo del prototipo, para comprobar la integridad de los diferentes buses del módulo. Una vez que han sido comprobadas las características eléctricas de la placa, se puede utilizar el monitor de bus para depurar el software. Debido a la gran complejidad de las transacciones del bus, es importante poder ser capaz de monitorizar la información que pasa por el bus para comprobar su correcto funcionamiento.

2.3 Otras características

Para evitar los posibles errores en las señales de reloj producidos por la distribución de las señales a lo largo del módulo (ruido cruzado, reflexiones, retardos ...), la señal de reloj se propaga por la placa a la mitad de su frecuencia y al llegar al procesador y a los integrados del Futurebus+ se restaura su valor inicial. Además se genera mediante unos dispositivos de reloj programables, con lo que en la fase de depuración del sistema, por medio de una serie de jumpers, se puede reducir la frecuencia para evitar los errores producidos por retardos y malas temporizaciones. Cuando se haya comprobado que todo funciona correctamente se puede poner al módulo a funcionar a toda potencia.

El módulo está provisto de una cache *write-through* de 256 Kb mapeada directamente con un tamaño de bloque de 16 bytes. El controlador de cache es el integrado HT44 de la casa *Headland Technologies*. Presenta el problema que es capaz de direccionar únicamente un gigabyte de los cuatro que consta el espacio de direccionamiento de los procesadores. Para solventarlo se utilizó una PAL reprogramable que elegía qué parte del espacio de direccionamiento iba a ser cacheada. Para invalidar los bloques de la cache y poder soportar por tanto un protocolo de coherencia para el sistema multiprocesador, se ha implementado un sistema híbrido hardware-software debido a las deficiencias del HT44. Para invalidar un bloque se debe activar la línea de *flush* y realizar un ciclo de lectura no cacheable. Aunque no se produzca un acierto en el acceso, el bloque que ocupa esa posición quedará invalidado. El proceso comienza ejecutando una instrucción *WBIND* que pone al controlador de bus en un modo que no propaga las lecturas al sistema, limitando así el efecto de la instrucción.

Los integrados de Texas Instruments no soportan actualmente el protocolo de coherencia de caches definido por FB+, por lo que se ha tenido que definir la cache como *write-through* para evitar problemas de consistencia con la memoria.

Las interrupciones dentro del módulo las maneja el controlador de interrupciones I8259C de Intel que es el encargado de recoger las posibles fuentes de interrupción, vectorizarlas y pasárselas a la pareja de procesadores. Las fuentes de interrupción pueden ser:

- Internas: Para la notificación de errores, del reloj en tiempo real, del interruptor hardware incorporado y de entrada/salida.
- Externas: Provocadas por los integrados de Futurebus+.

El propio módulo puede enviar interrupciones a través del bus del sistema a otros módulos, bien enviando un mensaje de arbitraje a través del bus de arbitraje (interrupción dirigida), o directamente escribiendo en alguno de los registros del CSR local al módulo que se quiere interrumpir mediante un ciclo normal de datos del FB+. Cabe destacar la posibilidad de enviar una interrupción a todos los

módulos del sistema a la vez para indicar algún evento especial o para realizar funciones de sincronización.

3 Sistema de detección de errores de la arquitectura FASST

A lo largo de las secciones anteriores se ha hecho una descripción de la estructura de los módulos de proceso. La principal característica que deben de tener es que deben de ser *fail-stop*, puesto que toda la garantía de funcionamiento que posee el sistema se basa precisamente en esta restricción, necesaria para poder implementar un mecanismo de reconfiguración y vuelta atrás para recuperar el sistema cuando se avería algún módulo. Dada la importancia que tiene la detección de los errores en el sistema, a continuación se detallan de forma explícita todos los mecanismos de detección de errores dentro del módulo, diferenciados por niveles de complejidad.

3.1 Fallos internos al procesador

Los procesadores de Intel incluyen una serie de mecanismos de protección internos para detectar cualquier anomalía en su funcionamiento. Más concretamente, el 486 dispone de el pin de *FERR#* que se activa cuando tiene lugar un error de coma flotante y frente a un error externo también permite la repetición de los ciclos del procesador mediante la activación de la señal de *BOFF#*.

La principal característica de este tipo de errores es que se producen por un mal funcionamiento de la aplicación que se está ejecutando en el procesador. El procesador puede detectar fallos en operaciones de división, sobrepasamientos, códigos de operación inválidos, errores en las operaciones en coma flotante y fallos de protección general dependiendo del nivel de privilegio utilizado. Cuando se detecta el fallo en la ejecución de la instrucción, se pasa a ejecutar una rutina de tratamiento de excepción.

Debido a la sencillez de la aplicación que se va a ejecutar en el prototipo es difícil que se produzcan errores de este tipo. Si por el contrario se llega a esta situación, será un síntoma inequívoco de que se ha producido alguna anomalía en el hardware del sistema y por lo tanto difícilmente se podrán corregir el error. Como el objetivo principal es evitar que se contamine el sistema con datos incorrectos, para hacer frente a estas situaciones se ha definido una rutina de tratamiento de excepción que simplemente notifica al usuario el tipo de excepción que se ha producido y a continuación detiene al procesador mediante la ejecución de la instrucción HALT. Este ciclo mantiene al procesador parado hasta que se recibe una interrupción externa o se activa la señal de reset.

Para evitar que la llegada de una interrupción despierte de nuevo al procesador, la interfaz con el procesador se ha diseñado de forma que una vez que se entra en el estado de HALT, se desactiva permanentemente la generación de las líneas de *RDY#* ó *BRDY#* . De este modo nunca se va a reconocer el ciclo de tratamiento de la interrupción y el procesador seguirá parado insertando estados de espera.

3.2 Detección del mal funcionamiento de alguno de los procesadores

Para detectar errores en el procesador se ha optado por implementar una arquitectura dual, duplicando el procesador e incorporando una serie de comparadores que monitorizan las señales del bus en busca de alguna discrepancia entre las señales duplicadas. Los comparadores, que ya han sido descritos

anteriormente, activan una señal de error indicando un fallo en la comparación de alguna línea del bus Procesador-Memoria (bien sea en las direcciones, bien en los datos, bien en las señales de control)[†].

Como es muy elevada la probabilidad de que el error sea de tipo transitorio, se intenta repetir la instrucción después de esperar un pequeño lapso de tiempo. Para ello, ante la detección del primer error, se activa la señal de *back-off* durante 16 ciclos de reloj que detiene la actividad del procesador en espera a que remitan los factores externos que produjeron el error transitorio. Cuando se desactiva la señal, el procesador repite la instrucción anterior.

Si el error se reproduce de nuevo, se desencadenan una serie de acciones con el objetivo de aislar al módulo del sistema, dejándolo a éste en un modo de funcionamiento degradado. El conjunto de eventos que tienen lugar se detalla a continuación:

- Desactivación de los procesadores (*bus hold*). Con esta medida se logra que el Host Bus, encargado de conectar el conjunto de integrados de la familia de FB+ con los recursos internos de la placa (RTC, Memoria y Registro de Control), que a su vez están compartidos por todos los módulos, quede disponible a los accesos externos.

- Desactivación de los drivers encargados de enviar las señales de arbitraje del FB+, con el objetivo de no entorpecer el buen funcionamiento del sistema de arbitraje distribuido de FB+.

- En función de la configuración inicial del sistema, se pueden habilitar permanentemente los receivers de entrada de FB+ para permitir el acceso a los recursos internos del módulo averiado o dejarlos inhabilitados.

3.3 Detección del mal funcionamiento de alguno de los comparadores

También es posible que se produzca algún fallo en los comparadores. Para garantizar una mayor cobertura de detección de errores se han duplicado los mismos, y las réplicas se han implementado utilizando lógica complementaria para evitar fallos de modo común.

El votador general recibe la información proveniente de los comparadores en un código redundante 1-de-2. Si la información que recibe sobre el estado de los procesadores no está codificada en una palabra que pertenece a este código, obviamente se deduce de que se trata de un error en los comparadores.

Si se produce este error se actúa de forma análoga al caso de un error de comparación y por lo tanto el módulo queda aislado del sistema.

[†] El error se detectará dependiendo de la cobertura de los comparadores. Se han diseñado para soportar todos los ciclos de bus posibles en el 486, por lo que la probabilidad de detección del fallo es muy elevada. Unicamente resaltar que como los comparadores funcionan de forma síncrona, si la duración del fallo es menor que el tiempo de ciclo del reloj de la CPU (33 Mhz), no se garantiza la detección del error. Sin embargo, el sistema no estará averiado puesto que el error no va a influir sobre el mal funcionamiento del procesador, cuyos ciclos también son síncronos y van gobernados por la misma señal de reloj que los comparadores (el Procesador y los comparadores funcionan en modo *lock-step).*

3.4 Detección de errores en los datos del bus del procesador

Cada uno de los buses de datos que conectan a los procesadores con el Host Bus está protegido por una línea de paridad. El procesador comprueba la paridad en el ciclo de reloj posterior a las lecturas de código, lecturas de memoria y lecturas del espacio de Entrada/Salida. Cuando se detecta un error de paridad, el procesador activa la línea de PCHK# que es recogida por una PAL que realiza el manejo global de los errores.

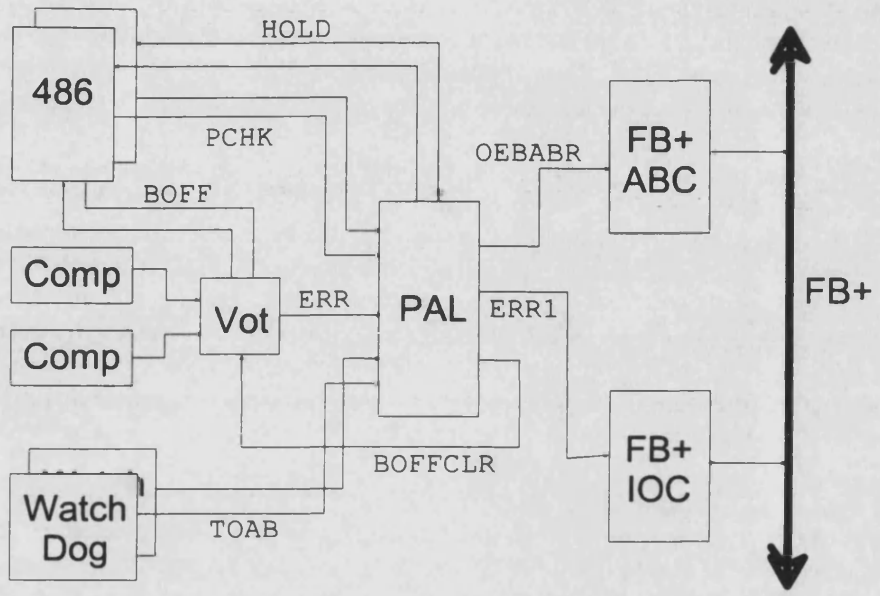


Figura 3.3: Sistema de detección de errores en el bus del procesador

El procesador comprueba la paridad en el ciclo de reloj siguiente al de la activación de la señal de RDY#, una vez están validados los datos. En el caso de que se detecte un error de paridad ya no se puede repetir el ciclo mediante la activación de la señal de BACKOFF# puesto que esto provocaría la repetición de la instrucción siguiente. Por lo tanto estos errores no se pueden solucionar y se procede al aislamiento del módulo.

3.5 Detección de errores en el flujo de ejecución de la aplicación actual

Se han implementado un par de temporizadores de guardia (*watch-dog timers*) para prevenir que el sistema quede en un estado desconocido debido a la pérdida del flujo normal de ejecución de las instrucciones. Este error no sería detectado si no se dispusiera de estos dispositivos.

Para que el manejo de los temporizadores de guardia sea lo más transparente posible al usuario, la iniciación de la temporización se va a realizar de forma automática cada vez que se ejecute alguna instrucción que produzca un ciclo bloqueado de bus (instrucción *exchange* o un ciclo de reconocimiento de interrupción) que se identifica exteriormente por la activación de la señal de *LOCK*. Esta señal es fácil de detectar, no debe aparecer en los bucles y además el Sistema Operativo utiliza de forma periódica para guardar los puntos de recuperación en la memoria estable y para enviar los menajes de "*estoy-vivo*".

La detección de errores de este tipo se activa escribiendo en el registro de configuración del módulo, de forma que en la inicialización del sistema no hay que preocuparse por activar la señal de *LOCK*. Al igual que en el caso anterior, si se produce un error, el sistema no tratará de recuperarse sino que se aislará del exterior mediante la desactivación de los drivers de FB+.

3.6 Detección de errores en el Host Bus

Aparte de los sistemas especiales de detección de errores que se incluyen en el módulo de proceso y que le dan las características de tolerancia a fallos al sistema, también se han implementado una serie de mecanismos de detección de error que permiten, en cierta manera, realizar una monitorización sobre el correcto funcionamiento de la placa y en su caso activar mecanismos de recuperación de error.

La señalización del error se va a realizar por medio de interrupciones. El controlador de memoria, implementado por medio de una EPLD, envía una interrupción al controlador de interrupciones (interrupción #5) para avisar al procesador de que se ha producido un error. El conjunto de errores que se van a detectar hacen referencia al direccionamiento de los distintos dispositivos que se definen en el espacio de direccionamiento del CSR. Cuando se detecta un error, la transacción en progreso se termina inmediatamente, mediante el reconocimiento de la misma, y se espera a que las señales de protocolo se desactiven (/hip="1" y /hds="1"). El resultado en las transacciones de lectura es que se escriben 0's en el bus de datos del Host Bus y 1's en los bits de paridad. Podemos distinguir los siguientes errores:

- Se accede al interfaz del CSR y se produce una transacción de escritura a la PROM, una transacción de lectura al LATCH, una acceso a una dirección de memoria que no está mapeada o un acceso a una dirección que no está alineada a palabra.

- Se accede a la LPROM y se produce una transacción de escritura, un acceso a una dirección mayor que 0x7FFFF o la combinación de las líneas que definen el tamaño de la transacción o el alineamiento de la misma es incorrecta.

- Se accede a la RAM y se produce un acceso a una dirección mayor que 0x1FFFF o la combinación de las líneas que definen el tamaño de la transacción o el alineamiento de la misma es incorrecta.

3.7 Detección de errores de Futurebus+

En el último nivel del sistema de manejo de errores se incluyen los errores ocurridos, bien en los accesos al conjunto de integrados del FB+, bien en las transacciones externas que tienen lugar a través del bus de expansión del sistema.

Estos errores se hacen visibles mediante una interrupción al procesador, siendo el manejador de la misma el encargado de realizar las acciones pertinentes de tratamiento del error. Las interrupciones provenientes del conjunto de integrados de FB+ se pueden habilitar o deshabilitar a voluntad mediante la escritura en el registro de máscara perteneciente al CSR.

En el caso de que se detecte un fallo en la alimentación del sistema se va a provocar una interrupción número 0, si el fallo se detecta en el proceso de arbitraje se producirá una interrupción número 1 y si el fallo de detecta en el controlador de FB+ se generará una interrupción número 2. (El conjunto de errores que detecta FB+ se puede ver en el capítulo anterior). El tratamiento que van a tener estos errores va a ser dependiente de la aplicación, según el contenido de la rutina de tratamiento de la interrupción.

Si no se produce la recuperación del estado de error, el sistema evolucionará debido a los demás mecanismos de detección de errores existentes en el módulo.

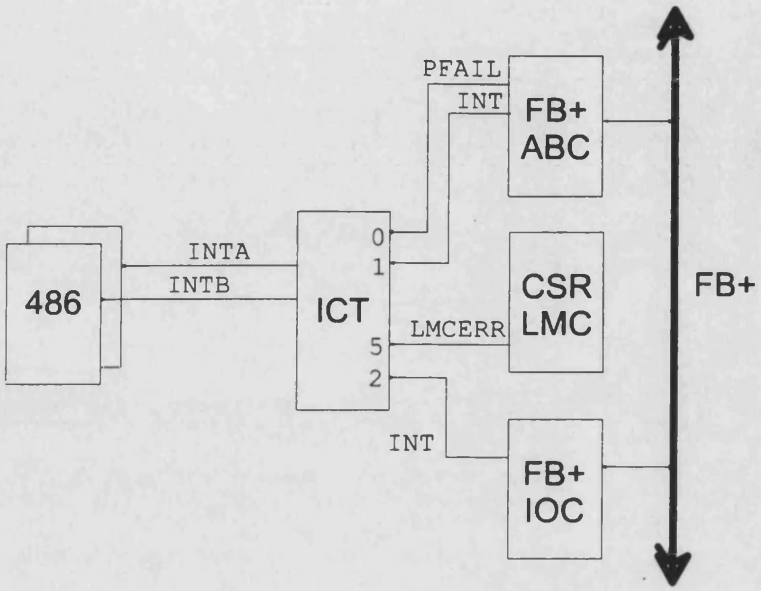


Figura 3.4: Detección de errores del Host Bus y de FB+

Futurebus+ permite el acceso a los recursos internos del sistema. Entre ellos se encuentran los registros del CSR del FB+ interface y el registro de configuración del sistema. A través de ellos se puede inicializar el módulo de forma local e incluso cambiar el procesador activo ante la detección de un error.

El análisis de la cobertura de recuperación de fallos a este nivel lo dejamos para un posterior estudio.

4 Construcción del demostrador

Una vez implementados los módulos de proceso, el siguiente objetivo fue la construcción de un demostrador que implementa un sistema real, basado en las especificaciones del Sistema FASST y que justifica cómo el sistema convive con los fallos en los módulos de proceso sin interrumpir su servicio y sin la pérdida de datos.

Debido a la gran complejidad del sistema FASST que se describe en el Capítulo 2, se decidió construir un sistema más simple, cuyos componentes serían placas comerciales, y que sirviera en cualquier caso para demostrar la validez del diseño hecho con las DPU's y como plataforma para la demostración del software tolerante a fallos.

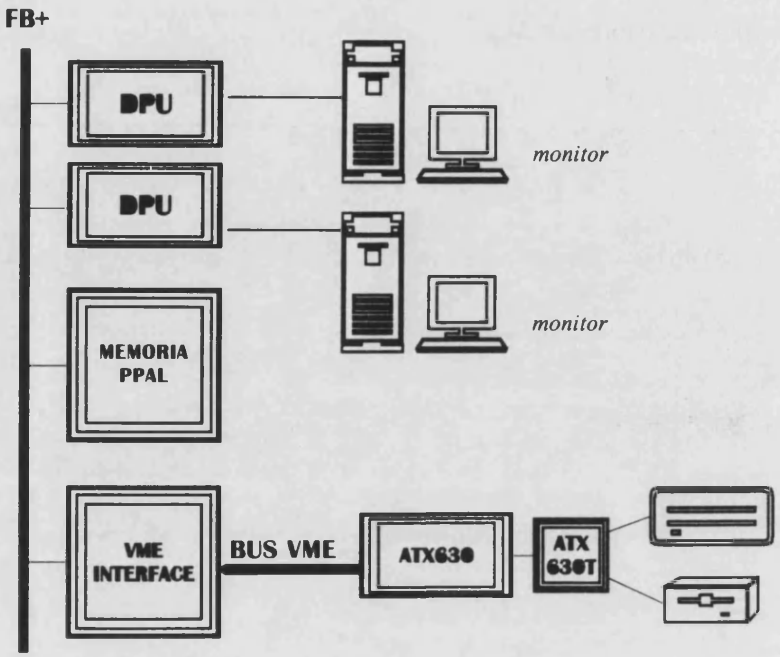


Figura 3.5: Diagrama del demostrador

La arquitectura del nuevo prototipo se muestra en la Figura 3.5. Está constituida por un *rack* de Futurebus+, dos módulos de proceso, un módulo de memoria, un módulo que hace de puente (*bus bridge*) entre el Futurebus+ y el bus VME y una placa de Entrada/Salida de propósito general conectada al bus VME. Todos los elementos del sistema, a excepción de las DPU son placas comerciales. En la Tabla 3.1 se puede ver una descripción más detallada de todos los módulos.

MUPAC 512 series FB+	Carcasa de FB+ con 5 ranuras y fuente de alimentación
NMEM-1	16 Mb de memoria Diseñada para ser compatible con las configuraciones A, B and F del estándar IEEE 896.2 FB+. Direccionamiento de 32 bits. Protocolo de coherencia de caches implementado Ancho del bus de datos de 32 y 64 bits.
FBV68LC040	Adaptador de bus FB+ a VME. Microprocesador Motorola 68LC040 (25 MHz). RAM de doble puerto de 16 Kb (accesible desde el 68LC040, el bus VME y FB+). SRAM de 512 Kb (accesible desde el bus VME y el 68LC040). Puerto RS232. Interface serie IEEE 1394.
ATX-630	Procesador MC68030. RAM de doble puerto de 2 Mb Controlador SCSI Controlador de disco flexible 2 puertos serie 1 puerto paralelo Interfaz Ethernet watch-dog 2 temporizadores programables

Tabla 3.1: Componentes del prototipo FASST

4.1 Mapa de memoria

El espacio de direccionamiento utiliza direcciones de 32 bits, por lo tanto disponemos de hasta 4 Gbytes de memoria física. Principalmente podemos diferenciar dos partes, los 256 Mbytes de la parte alta del espacio de direccionamiento, que están reservados para el CSR de todos los módulos, y los 3840 Mbytes restantes, que son la memoria del sistema (memoria compartida, memoria local compartida y memoria local privada)

- **Memoria Compartida:** Este espacio está dedicado a los módulos que van a funcionar como esclavos en las transacciones y que por tanto no necesitan tener capacidad de arbitraje.

- **Memoria Local Compartida:** Esta memoria es accesible desde cualquier módulo de FB+. La máxima cantidad permitida son 8 Mb por módulo para un total de 31.

- **Memoria Local Privada:** Esta memoria es privada para el módulo y contiene información de diagnóstico o de inicialización. La cantidad máxima son 8 Mbytes por módulo, y se mapea en la misma dirección para todos los módulos, más concretamente utiliza la dirección del nodo 31 que no existe).

La memoria local de la DPU está formada por 512 Kbytes de memoria ROM y 128 Kbytes de memoria RAM, que están mapeados respectivamente en las direcciones bajas del espacio de direccionamiento local privado y compartido.

FB+ permite establecer un mapa de memoria dinámico que se configura a través de los registros del CSR de los módulos. Es importante tener en cuenta la topología del sistema puesto que el software de configuración inicial de las placas reconfigura dinámicamente el mapa de memoria dependiendo del contenido de las líneas de dirección geográfica de FB+ (GA[4..0]). De esta manera se puede implementar el software para que funcione independientemente de la posición que ocupa el módulo. A continuación presentamos el mapa de memoria según como queda configurado en el demostrador. El módulo puente entre FB+ y VME está insertado en la ranura 3 del *rack*, la memoria global del sistema ocupa la ranura 2 y las dos DPU's ocupan respectivamente las ranuras 1 y 5 del bus.

MEMORIA GLOBAL FB+				
Nº Slot	Tipo de placa	Dir inicio	Dir final	Descripción
Slot 1	Technobox	200 0000	200 3FFF	Memoria de doble puerto
Slot 2	Nanotek	1000 0000	1FFF FFFF	Memoria global
Slot 3	DPU	60 0000	68 0000	Memoria local compartida
Slot 4	-	-	-	
Slot 5	DPU	A0 0000	A8 0000	Memoria local compartida

CSR LOCAL A LAS DPU's		
	CSRs DPU-3	CSRs DPU-5
PROM	FFFC 6400	FFFC A400
TIMER	FFFC 6800	FFFC A800
UART	FFFC 6900	FFFC A900
INTC	FFFC 6A00	FFFC AA00
LATCH	FFFC 6B00	FFFC AB00

En la Figura 3.6 se puede observar una fotografía del sistema con todos los módulos insertados.

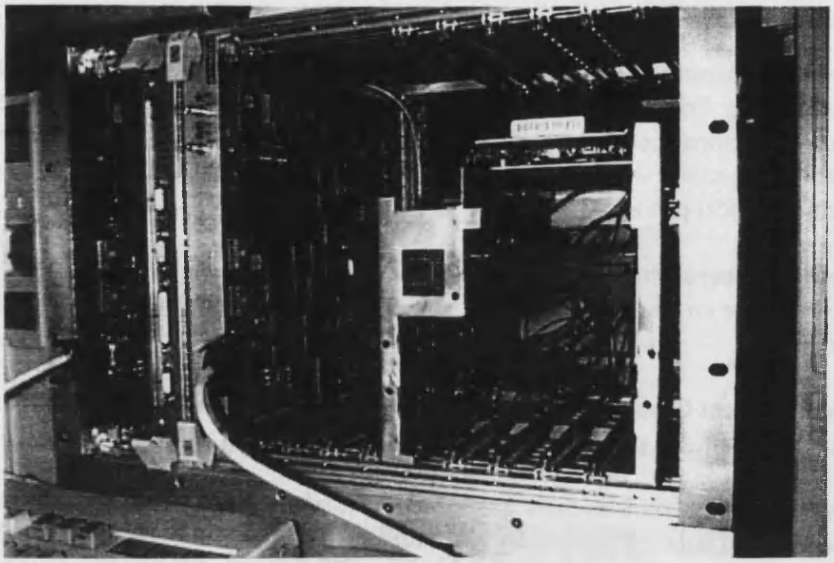


Figura 3.6: Fotografía del Sistema FASST

5 Desarrollo del software

En paralelo a la implementación del sistema prototipo, se ha ido desarrollando software para poder comprobar el correcto funcionamiento de los módulos del sistema y para demostrar sus características de tolerancia a fallos. La primera tarea que se realizó fue el desarrollo de un programa monitor con dos objetivos:

1. Permitir el acceso a cualquier posición del espacio de direccionamiento del sistema, incluyendo también las posiciones de E/S y los registros del CSR. De esta manera ya se puede configurar el sistema y comprobar el funcionamiento de todas sus capacidades: las transacciones al Futurebus+, las alarmas, las interrupciones, el reloj en tiempo real, el puerto serie, comprobación de la ROM y de la RAM interna y configuración de los integrados del Futurebus+.

2. Implementar una serie de funciones de bajo nivel, almacenadas en la EPROM, que faciliten la programación de aplicaciones y la construcción de drivers. Con el uso de esta librería se obtendrán programas más compactos que utilizan menos memoria RAM, ya que no incluyen el código de estas funciones, sino llamadas a las mismas. Aspecto muy importante puesto que la cantidad de memoria local al módulo es bastante limitada.

Cuando se arranca el sistema, el procesador se coloca en modo de 32 bits, se inicializan los registros de segmento y los vectores de interrupción, se pasa un test de comprobación de la memoria local del sistema y se presenta por pantalla el mensaje de bienvenida. A continuación se comprueba si el operador está pulsando alguna tecla. Si no se ha pulsado nada, se ejecuta la aplicación de usuario almacenada en la ROM. En caso contrario se pasa a ejecutar el monitor del sistema descrito anteriormente. La Tabla 3.2 muestra el conjunto de funciones que implementa el monitor. En la primera columna se puede observar los elementos del menú principal y en las siguientes los diversos submenús.

La aplicación de la Figura 3.7 muestra dos DPU's funcionando en paralelo, y cada una ejecutando sus propias tareas. Cuando un módulo detecta un fallo se aísla del sistema, permitiendo a la otra DPU continuar las tareas del módulo averiado junto con las suyas. Todo este proceso se basa en el

establecimiento de puntos de recuperación para mantener la consistencia de los datos del sistema y en el envío de mensajes de "*estoy vivo*" y el uso de temporizadores para detectar las averías en los módulos. Las averías se pueden introducir de forma artificial mediante la activación de un interruptor. En este momento la tarjeta deja de enviar los mensajes simulando una avería. Activando otra vez el interruptor se pone a funcionar de nuevo, con lo que se puede simular el proceso de reparación e inserción en vivo de los módulos.

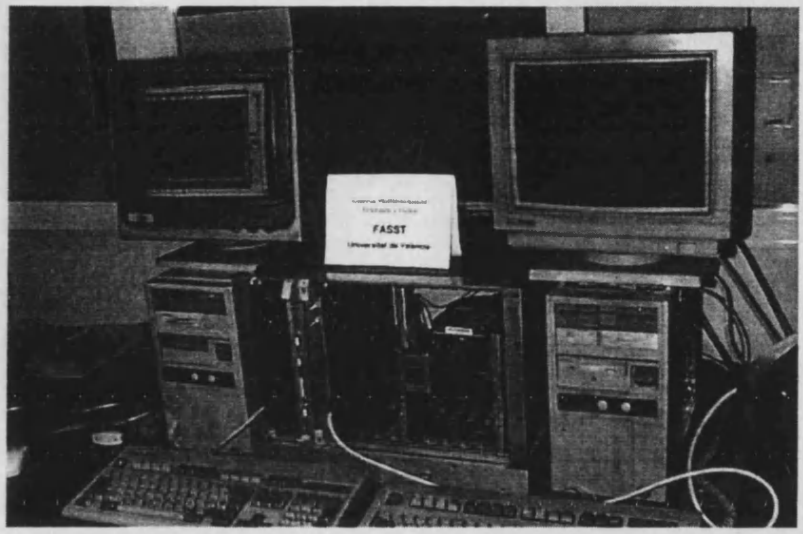


Figura 3.7: Fotografía del entorno de ejecución de la aplicación

La segunda tarea realizada fue el desarrollo de una aplicación de usuario que funcionara dentro de un sistema prototipo basado en la arquitectura FASST y que demostrara las características de tolerancia a fallos que dispone el sistema. El software desarrollado, aunque está definido para funcionar en el demostrador, intenta mostrar un amplio abanico de posibilidades ofrecidas por Futurebus+ y por la DPU. El software ha sido estructurado en dos módulos de forma que se facilita su revisión, ampliación y portabilidad a otros entornos. Los módulos son:

- La máquina de estados dirigida por eventos (ESM)
- El ejecutor de la aplicación (AE)

El primero se comunica con el hardware para realizar la detección de errores y la reconfiguración dinámica del sistema, construyendo una interfaz que puede ser utilizado por las aplicaciones de alto nivel. El segundo representa este software de alto nivel, y proporciona a las tareas de usuario un entorno cooperativo multitarea que maneja la recuperación del error a través de la migración de tareas. En la inicialización el monarca configura el sistema y luego transfiere el control al ESM de las DPU. En [20] se especifica un algoritmo para la selección del monarca.

MENU PRINCIPAL	Submenús
Datos monitor	
Memoria	Volcar
	Llenar
	Buscar
	Comparar

	Copiar	
	Escribir 1-byte	
	Leer 1-byte	
	Escribir 32 bits	
	Leer 32 bits	
CSR local	Escribir registro	
	Leer registro	
	Volcar registros	
Puerto Serie	Configurar	Longitud
		Bits de stop
		Paridad
		Tipo paridad
		Break
	Comprobar	
	Cambiar velocidad	ninguno
		122.070 us
		244.141 us
		488.281 us
		---------
		250 ms
		500 ms
		no cambiar
Reloj en Tiempo Real	Leer reg.	
	Escribir reg.	
	Leer reg. hora	
	Escribir reg. hora	
	Poner en marcha (ON)	
	Parar (OFF)	
	Poner en pausa (HOLD)	
	Ver hora	
	Poner hora	
	Ver hora alarma	
	Poner hora alarma	
	Interrupciones	Habilitar
		Deshabilitar
		Ver estado
		Establ. periodo
		Reg. manejador
		Desreg. manejador
Ficheros S-RECORD	Cargar	
	Ejecutar	
Transferir control		
Interrupciones	Inicializar	
	Poner a cero contadores	
	Mostrar estado	
	Cambiar mascara	
	Habilitar	
	Deshabilitar	
	Reg. Man. IRQ	
	Desreg. Man. IRQ	
	Reg. Man. Excepcion	
	Desreg. Man. excepcion	

Tabla 3.2: Funciones implementadas en el monitor

5.1 ESM

Las interrupciones hardware y otra serie de situaciones del sistema se convierten en eventos, que producen cambios de estado en el sistema. La detección de errores y la recuperación son cambios de estado que se producen precisamente debido a la generación de un evento.

Una de las DPU hace de monarca del sistema, siendo por tanto responsable de la reconfiguración del sistema en caso de fallo o de inserción en vivo. Si falla el monarca se debe de elegir otro, aunque esta situación no tenga mucho sentido puesto que en el demostrador sólo existen dos módulos, pero sin embargo se puede utilizar el algoritmo especificado en [21].

Los ESM que se ejecutan en DPU's diferentes no comparten memoria, sino que intercambian mensajes periódicos de "*Estoy vivo*", que se implementan por medio de interrupciones dirigidas de FB+. En un sistema con más de dos módulos de proceso debe implementarse una estructura global de datos en donde cada procesador actualice periódicamente un bit indicando que está vivo. Comprobando esta estructura se pueden detectar las averías de los módulos. El fallo debe notificarse a todos los módulos, utilizando una interrupción dirigida para hacer el envío simultáneo.

En la tabla de la Tabla 3.3 se muestran los posibles estados del sistema y sus significados:

Estados del ESM	**Descripción**
OK	El sistema está libre de fallos
SOLO	La otra DPU ha fallado y ahora soy el monarca
DEBUG	En modo de test. Estoy deshabilitado mientras dure el test
FALLO	La DPU está averiada hasta que haya una inserción en vivo
LISTO	La depuración ha finalizado o se ha insertado una nueva tarjeta. En espera de la llegada de un mensaje de "Estoy vivo"

Tabla 3.3: Estados del ESM

El estado del sistema está definido por cuatro banderas (*AmIAlive*, *AmIOnline*, *IsAlive* y *AmIMonarch*) y dos variables (*Tics* y *TicsLastAlive*). La variable *Tics* se utiliza para llevar cuenta del tiempo transcurrido desde la puesta en marcha del sistema y la variable *TicsLastAlive* guarda el instante en el que se recibió el último mensaje de "estoy vivo", para poder calcular si se ha producido un error de temporización.

En la Figura 3.8 se muestra la máquina de estados. Los nodos representan a los estados y los arcos indican las posibles transiciones. A cada arco se le ha puesto una etiqueta que indica el evento que produce la transición. Los posibles eventos se describen en la Tabla 3.4.

"Estoy vivo"	Es una interrupción dirigida de Futurebus+ cuyo manejador escribe el tiempo de generación de la interrupción en *TicsLastAlive*. El manejador del temporizador detecta un "*timeout*" restándole a este valor el valor de *Tics*.
"alineamiento"	El monarca debe de configurar la otra DPU cuando es insertada en vivo. Este evento lo detecta el manejador de la interrupción del temporizador, al comprobar el bit de "*alineamiento ocurrido*" en el registro del CSR "IOC STATUS CLEAR". La comprobación se realiza siempre que el monarca esté en el estado de "SOLO".
"*timeout*"	Se han contabilizado demasiados espacios de tiempo sin recibir un mensaje de "estoy vivo". Esta situación la detecta el manejador de la interrupción del temporizador.
"excepción"	Excepción del 486 manejada por el depurador. Se dejan de enviar mensajes de "I'm alive" para que el otro módulo pueda recuperar el error.
"depuración terminada"	Se retorna de la excepción de depuración, se empiezan a enviar mensajes pero se espera a que la otra DPU reciba el primer mensaje para continuar con la demostración.
"fallo de	Se aísla el módulo del sistema y se abandona toda actividad (no se

alimentación"	envían mensajes, ni se accede a los datos de la demostración).
"nmi"	Los comparadores han detectado un error por lo que hay que aislar al módulo del sistema y abandonar cualquier actividad. Se debe de encender el led de "FALLO" del panel frontal.
"inserción en vivo"	Ha ocurrido una inserción, por lo que hay que esperar a que el monarca configure el módulo. Cuando se reciba un mensaje de "estoy vivo" es que ya se habrá completado la configuración.
"arranque"	Envía mensajes de "estoy vivo" y se espera a que llegue una respuesta. Todos los módulos realizan esta tarea.

Tabla 3.4: Eventos del sistema

Detección de errores y reconfiguración del sistema

Cuando una DPU detecta un error, la placa puede estar programada para que genere una interrupción no enmascarable y se aísle del sistema o para que simplemente genere la interrupción NMI y espere a que se produzca un nuevo error antes de aislar la placa. En el primer caso el procesador se para, mientras que en el segundo se ejecuta una rutina de tratamiento de la interrupción no enmascarable que envía un mensaje al terminal y entonces se para. La otra DPU eventualmente detecta un error de "*timeout*" y cambia el estado a "SOLO" para pasar a realizar todo el trabajo.

Inserción en vivo

La placa que acaba de ser insertada debe de esperar a que el monarca active el bit de habilitación del maestro en el registro del CSR LOGICAL_MODULE_CONTROL. A continuación la placa pasa al estado de "DISPUESTO" y empieza a enviar mensajes de "estoy vivo". La otra placa pasa del estado "SOLO" al estado "OK" en el momento que recibe el primer mensaje del módulo que acaba de incorporarse y entonces empieza a enviar mensajes que hacen que el otro módulo también pase a "OK".

Figura 3.8: Estructura del módulo ESM

No se pueden producir situaciones erróneas o condiciones de carrera, ya que el módulo que se incorpora no empieza a acceder a los datos hasta que el monarca le dice que lo haga. Y además el monarca espera para acceder a los datos hasta que el AE le indica.

Depuración

Para realizar propósitos de depuración se puede detener a cualquier placa en cualquier momento, pasando al estado de "DEPURACIÖN". El sistema evoluciona como si se tratara de un fallo del módulo, con la única salvedad de que no se puede acceder a los datos de la demostración para que no se produzcan condiciones de carrera. Cuando termina la depuración la DPU entra en el estado "DISPUESTO" y espera al reconocimiento de la misma forma que cuando se produce una inserción en vivo.

Para realizar la depuración se ha utilizado el depurador simbólico de GNU [34] *gdb* [33]. Se puede ejecutar en una gran variedad de máquinas UNIX y permite la depuración remota a través del puerto serie. Para poder utilizarlo se debe de incluir en la aplicación una pequeña cantidad de código que le permita comunicarse con el host central. Este código viene suministrado junto con la distribución para un gran número de plataformas distintas, incluyendo las basadas en procesadores i486 de Intel.

La gran ventaja que tiene es que este código es casi independiente de la plataforma, únicamente hay que programar tres funciones (*putDebugChar*(), *getDebugChar*() and *exceptionHandler*()) para comunicarse por el puerto serie y para controlar los manejadores de interrupción. Además se le han

añadido unas pocas líneas para que se produzca un cambio de estado en el sistema cuando se ejecute la rutina de tratamiento del depurador.

Interfaz con el usuario

Para poder ejecutar el depurador, el usuario puede activar el interruptor del panel frontal de la placa que produce la interrupción del depurador. El depurador suministra información para poder acceder a los datos del sistema y una forma de simular los errores de la DPU sin necesidad de eliminar la placa. La inserción en vivo puede simularse también permitiendo la continuación de la ejecución del programa al terminar el depurador.

5.2 AE

Estos son módulos independientes que no tienen información sobre los eventos o mensajes del sistema. Simplemente llevan cuenta del estado del ESM que se ejecuta en su DPU y de la aplicación del usuario. Continuamente se le da tiempo de CPU a la aplicación para que se ejecute y establezca un punto de recuperación. A continuación se le devuelve el control al AE que vuelve a ceder la CPU a una nueva tarea. Para los propósitos de la demostración, se han implementado dos tareas que pueden ir de una DPU a la otra tan pronto como se vayan produciendo los errores. En este caso, la tarea vuelve a empezar la ejecución desde el último punto de recuperación.

El AE se va reconfigurando dinámicamente según el estado del ESM siguiendo los pasos de la siguiente tabla:

- Cuando el estado es "OK" se le da todo el tiempo de la CPU a la tarea inicial de la DPU
- Cuando el estado cambia a "SOLO" se recupera el estado de la tarea que se ejecutaba en el módulo averiado desde el último punto de recuperación, y se comparte el tiempo de la CPU entre las dos tareas.
- Cuando el estado cambia de "SOLO" a "OK", debido a que se ha recuperado la DPU o se ha sustituido por una nueva, el AE informa al ESM de que el nuevo módulo puede seguir con la tarea desde el último punto de comprobación establecido. El AE de la nueva DPU espera a que su ESM entre en el estado de "OK" antes de continuar con la tarea para evitar condiciones de carrera.

El software diseñado para la demostración está formado por dos módulos: el primero, llamado *módulo local*, reside en ROM, y es responsable de realizar la comprobación inicial de arranque del sistema (*Power On Self-Test - POST*). Si el test termina con éxito se comprueba si existe un terminal conectado al puerto serie. En caso afirmativo se empieza a ejecutar un monitor local que permite la carga de programas y su depuración, sin necesidad de acceder a los recursos del FB+. Si el módulo está en línea (terminal no conectado) se transfiere el control al segundo módulo.

El segundo módulo es el llamado *módulo global* y se encarga de realizar la demostración propiamente dicha. Lo primero que hace es comprobar si se ha producido un alineamiento del módulo, producto de una inserción en vivo. Si no es así comienza la fase de elección del monarca, puesto que se trata de un arranque del sistema. Todos los módulos envían un mensaje de arbitraje que contiene su dirección geográfica y esperan la llegada de un nuevo mensaje que indica el módulo que va a ser el próximo monarca. El nuevo monarca está encargado de realizar las siguientes tareas:

- Autoconfiguración para poder acceder al FB+ e identificar cuál es su identificativo de nodo. A continuación, para poder acceder a sus recursos internos, programa su mapa de memoria.
- Identifica la configuración del sistema, comprobando las otras DPU's que residen en el bus, la memoria principal y las placas de conexión al VME. En esta fase también programa los registros de FB+ con los parámetros por defecto.
- Carga el software de alto nivel del disco a la memoria principal.
- Configura los registros del CSR de otros módulos.
- Permite a los demás módulos acceder al FB+ una vez está todo configurado.

Si el módulo ha sido insertado en vivo, debe de esperar a que el monarca del sistema le permita el acceso al FB+. Para ello comprueba periódicamente el bit de habilitación del maestro en el registro del CSR "LOGICAL MODULE CONTROL". Cuando se le pasa el control, como argumentos se le pasa su identificativo de nodo. Este módulo permite la depuración simbólica a través del puerto serie (usando el depurador de gnu).

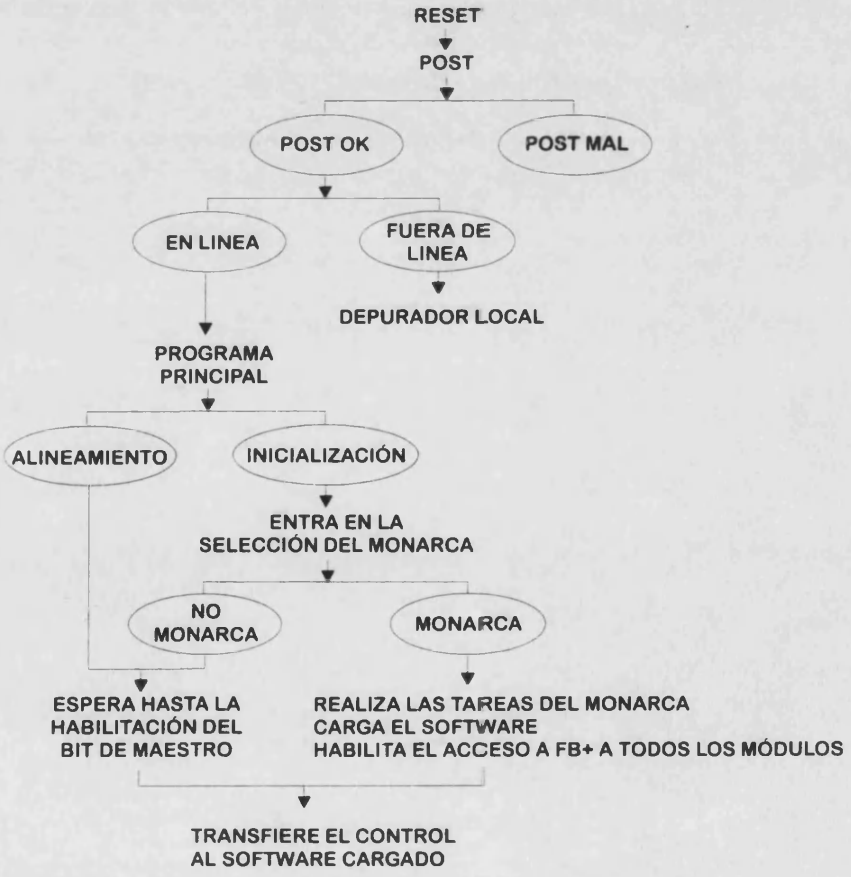


Figura 3.9: Proceso de inicialización de la DPU y el FB+

6 Conclusiones

En este capítulo se ha descrito la estructura del Sistema FASST. Es indispensable conocer por dentro al sistema para poder entender cómo funcionan, o lo bien que funcionan sus mecanismos de tolerancia a fallos.

Se ha puesto especial énfasis en explicar cuales son estas características, puesto que son los que justifican la garantía de funcionamiento del sistema. Dependiendo de estos mecanismos se van a tener que implementar nuevos modelos estocásticos y dirigir la inyección de fallos para poder obtener medidas cuantitativas de sus coberturas de detección de errores.

El manejo de errores transitorios por parte del comparador ha sido rediseñado, verificado e implementado (más bien diría "parcheado", si se me permite la vulgaridad), reprogramando las EPLD's, cortando pistas de la placa de circuito impreso y sustituyendo las conexiones por hilos de grapinar. He considerado oportuno incluir en el apéndice una lista con los errores detectados y subsanados de los módulos de proceso del sistema FASST.

Otra aportación más ha sido la construcción de un monitor para el prototipo. Este monitor permite acceder a los recursos internos de los módulos de proceso y por lo tanto ha servido para depurar el sistema. Permite programar el reloj en tiempo real, activar interrupciones periódicas, manejar las interrupciones, acceder a los registros del CSR, acceder a todo el espacio de direccionamiento del Futurebus+ y cargar y ejecutar programas mediante el puerto serie. Se ha implementado una librería de funciones en ROM para facilitar el desarrollo de nuevas aplicaciones.

Aparte del software desarrollado, también se ha programado la aplicación que justifica la tolerancia a fallos del sistema.

Capítulo 4

Futurebus+

1 Introducción

Futurebus+ [18][27] es un estándar del *Institute of Electrical and Electronic Engineers, Inc (IEEE)* que define los niveles lógicos, físicos y el entorno necesario para construir sistemas y periféricos compatibles con el bus Futurebus+. La idea principal que se tuvo en cuenta en el diseño de este bus es que fuera un bus de muy altas prestaciones, totalmente definido a través de estándares, y sobre todo escalable, sin necesidad de depender de la tecnología, para poder hacer frente a las necesidades de ancho de banda de los futuros sistemas multiprocesador. Futurebus+ es tal vez el mayor estándar que incluye un extenso soporte adecuado para los sistemas tolerantes a fallos [26], como conexiones distribuidas y protocolos de arbitraje distribuidos. Por estos motivos se eligió este bus como la base para el sistema FASST .

Futurebus+ utiliza la misma estructura que los buses VME, pero incorpora ciertas características para aumentar al máximo la velocidad de transferencia de datos a través del bus. Uno de los métodos que utiliza para aumentar el ancho de banda es el aumento del ancho de los buses de datos. Futurebus+ tiene un bus de 64 bits que se puede utilizar tanto en modo de 32 bits como aumentar el ancho a 128 y 256 líneas. El resultado es que puede transferir datos 4 veces más rápido que el VME64 utilizando la misma frecuencia de reloj. También utiliza lógica BTL, que elimina los tiempos de set-up y de hold, incorpora baja capacidad en las líneas y suministra gran cantidad de corriente, lo que permiten aumentar la frecuencia hasta 100 MHz.

Profile	Propósito
Profile A	Propósito General
Profile B	Entrada / Salida
Profile C	Cables
Profile D	Sobremesa
Profile F	Workstation
Profile M	Militar
Profile T	Telecomunicaciones

Tabla 4.1: Ejemplos de profiles definidos en FB+

Aparte del ancho de banda, también incorpora diversos tipos de arbitraje (centralizado o distribuido), la posibilidad de elegir transferencias empaquetadas o no, la posibilidad de añadir prioridades a la hora

de utilizar el bus, protocolos de coherencia de cachés para los multiprocesadores con memoria compartida, la inserción y la extracción de módulos en vivo y un interface estándar a través de registros de estado y de control [22].

Futurebus+ fue diseñado inicialmente como un bus para mantener la coherencia de las caches en memoria compartida, sin embargo también maneja transferencias de bloques grandes y el paso de mensajes. Sus desventajas son que necesita demasiadas patas en el conector y que no está definido un modo de corrección de errores. Si se quiere lograr este modo, se requiere el uso de buses duplicados, con lo que se duplica también el número de pines.

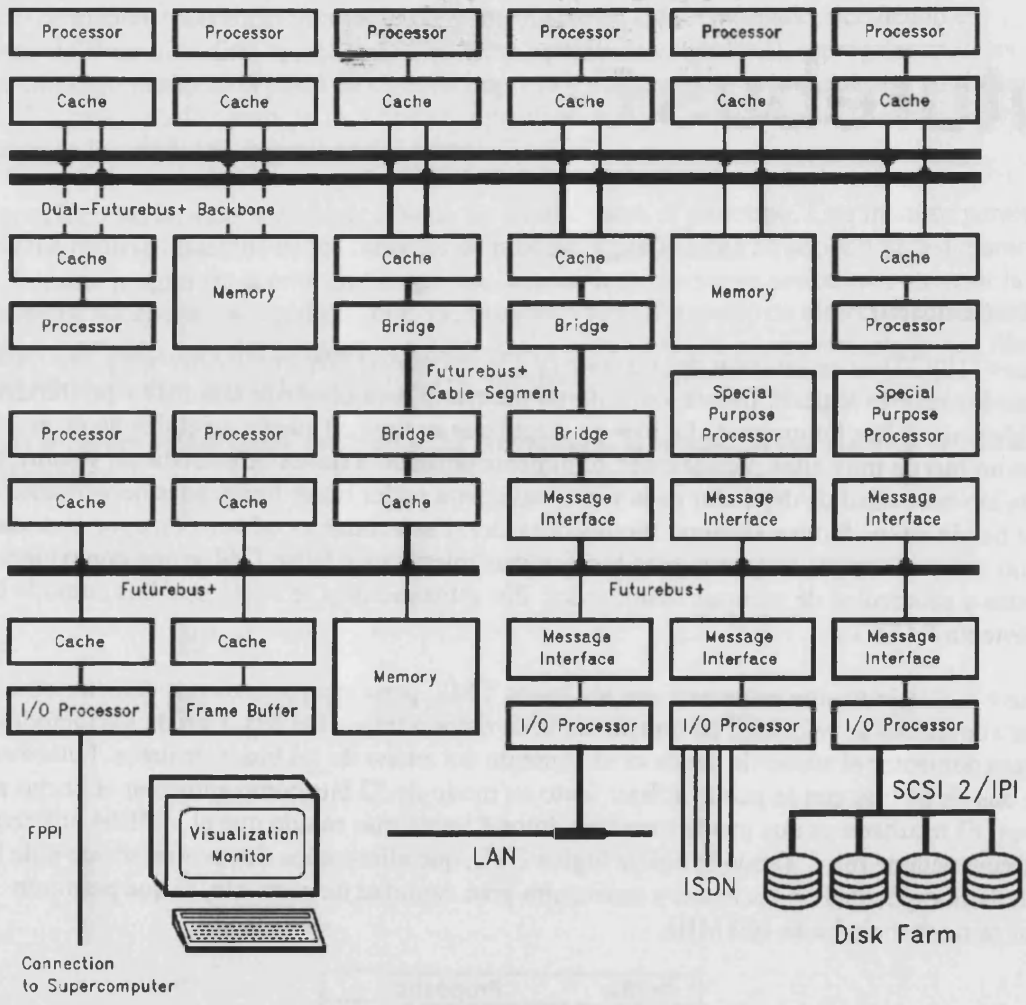


Figura 4.1: Aplicación típica de FB+

FB+ soporta una serie de configuraciones diferentes apropiadas para distintos sistemas. A estas configuraciones se les denomina *profiles* como muestra la Tabla 4.1. La función de un profile es la de asegurar la compatibilidad con la clase de productos que se adhieren a sus normas, delimitando de esta manera la vasta cantidad de opciones que ofrece FB+. Esto se logra teniendo en cuenta tres niveles totalmente independientes:

- Definición a través del nivel lógico de las señales necesarias, sus temporizaciones y cómo interactúan unas con otras. Definición de las funciones de protocolo soportadas y del comportamiento jerárquico definido por la arquitectura del sistema.

- Especificación a nivel físico de las conexiones eléctricas y mecánicas, de los parámetros eléctricos de las señales y de la alimentación y de sus tolerancias para asegurar el funcionamiento correcto.

- Especificación del entorno tanto para las aplicaciones como para los parámetros que controlan la fiabilidad del sistema.

En la Figura 4.1 se puede ver una aplicación típica de Futurebus+ en donde existe una estructura jerárquica de buses todos conectados a través de puentes.

2 Líneas del bus

Las señales se agrupan en varios grupos funcionales:

Información: Son las líneas por donde circulan los datos y son conocidas, junto con sus correspondientes líneas de sincronismo, como bus paralelo. Posee 256 líneas de datos, las cuales las 64 primeras estén multiplexadas con las direcciones (D[255.64], AD[63..0]), líneas de comandos para indicar el tipo de transacción (CM[7..0]), líneas de estado y habilidades, para indicar las funcionalidades implementadas en el módulo, así como dar respuesta a las peticiones de transacción (ST[7..0], CA[2..0]) y una serie de líneas sin especificar que pueden ser utilizadas y definidas por el usuario con cualquier propósito (TG[7..0]). Aparte de este conjunto de líneas existen una línea de paridad bit por byte para comprobar la integridad de los datos.

Sincronización: Se utilizan en los protocolos de transferencia asíncronos. AS, AK y AI están activas durante la fase de direccionamiento y DS, DK y DI durante la transferencia de datos. Respectivamente la S indica la validez de los datos, la K el reconocimiento y I el reconocimiento inverso.

Arbitraje: Para el arbitraje distribuido se utilizan las líneas de sincronización AP, AQ, AR y las líneas AB[7..0] para transmitir el número de arbitraje. Dos líneas más (AC[1..0]) dan información adicional para el control de errores.

Inicialización: Esta compuesta por una sola línea que inicializa el sistema y que se llama RE. Esta línea tiene varias funciones según el tiempo que se mantenga activa. Sirve para inicializar el sistema completo, para reinicializar solamente las interfaces al bus, y para alinear los módulos después de que son insertados al sistema.

Dirección Geográfica: Son 5 líneas (GA[4..0]) que indican la posición física del módulo dentro del bus del sistema. Son usadas para varios propósitos, entre ellos el de la configuración del mapa de memoria o el del establecimiento de los números de arbitraje ya que se exige que dos módulos nunca pueden poseer el mismo número.

Es destacable el hecho de que no posee ninguna línea dedicada para las interrupciones, ni ninguna línea para reloj, ya que como se comentó anteriormente, puesto que es un bus básicamente asíncrono, que no precisa de reloj compartido. Tampoco existen líneas de DMA puesto que ya vienen implementadas las transferencias por bloques en el protocolo paralelo.

3 Características

Futurebus+ es un estándar que trata de ser válido para generaciones futuras de ordenadores, por lo tanto no ha sido diseñado para una arquitectura o procesador específico. Las características más importantes de FB+ son las siguientes:

- El bus está pensado para que sea tecnológicamente independiente, es decir que se establezca como una arquitectura abierta que se adapte a cualquier tipo de tecnología. Para ello soporta protocolos de transferencia asíncronos, cuyo funcionamiento es independiente de la frecuencia de reloj escogida.

- Contempla la posibilidad de utilizar protocolos síncronos para la transferencia de datos, lo que supone lograr velocidades de transferencia muy altos.

- Permite la utilización de protocolos tanto de arbitraje distribuido, interesante desde el punto de vista de la tolerancia a fallos, ya que evita la inclusión de puntos únicos de fallo en el sistema (avería del árbitro central), como de arbitraje centralizado que ofrece un mayor rendimiento, si bien necesita un backplane más complejo.

- Ofrece mecanismos de protección de la integridad de los datos, como control de paridad para todas las líneas del bus y la posibilidad de realizar un mantenimiento en línea, que implica la posibilidad de extraer e insertar las tarjetas del sistema a pleno funcionamiento.

- Su sistema de arbitraje flexible permite hasta 256 niveles de prioridad, con la posibilidad de obtener un acceso rápido al bus (*preemtion*), la posibilidad de acceso igualitario (*fairness*) y la posibilidad de programar de forma dinámica las prioridades de los módulos. Estas características lo hacen especialmente apropiado para los sistemas en tiempo real.

- Es uno de los pocos buses que existen en la actualidad que incorpora protocolos de coherencia de caches en su estándar, apropiados para sistemas con memoria compartida, y permite la implementación de complejas jerarquías de buses.

- Implementa interrupciones virtuales. No existen líneas dedicadas para las interrupciones, sino que se produce una interrupción mediante una transacción de escritura por el bus de datos a un registro especial del módulo que se quiere interrumpir, o mediante el envío de un mensaje de arbitraje con un código especial a través del bus de arbitraje.

- Lógica BTL: Inclusión de una nueva tecnología que aumenta considerablemente las prestaciones, superando a cualquier bus basado en lógica TTL [23]. Todas las señales eléctricas del bus utilizan niveles de voltaje BTL (*Backplane Transceiver Logic*). Esta tecnología fija los niveles lógicos en 1 y 2 voltios, con lo que la transición entre un estado y otro es de tan solo un voltio [118] y tiene la ventaja que los amplificadores de bus presentan una capacidad pequeña pero que la vez son capaces de suministrar mucha corriente. El resultado es que las señales tienen unos retardos de propagación menores a lo largo del bus.

- Las especificaciones de Futurebus+ incluyen el estándar del IEEE del CSR (*Command and Status Registers)* [22].Son una serie de registros que mantienen información general del sistema y que suministran una serie de métodos estándar para producir interrupciones, notificar errores, ejecutar comprobaciones, sincronización del reloj en tiempo real y otra serie de funciones. Futurebus+ sigue el estándar CSR para el direccionamiento de registros lo cual lo convierte en un sistema abierto y fácilmente portable, además de poder reconfigurar de forma dinámica el mapa de memoria del sistema.

4 Protocolos distribuidos

Las partes más criticas de un sistema tolerante a fallos son aquellas que se denominan los puntos únicos de fallos, en donde un único fallo produce la avería del sistema completo. Futurebus+ trata de evitar estos casos de formas muy variadas, incluyendo redundancia temporal y utilizando protocolos distribuidos.

4.1 Arbitraje del bus

En el caso del arbitraje, el árbitro central constituye un punto único de fallos, sin embargo FB+ permite la reconfiguración dinámica del sistema de arbitraje de centralizado a distribuido en el momento en que se produzca un fallo. La combinación de los dos esquemas permite disfrutar de las ventajas de prestaciones del sistema centralizado y de la alta garantía de funcionamiento del sistema distribuido.

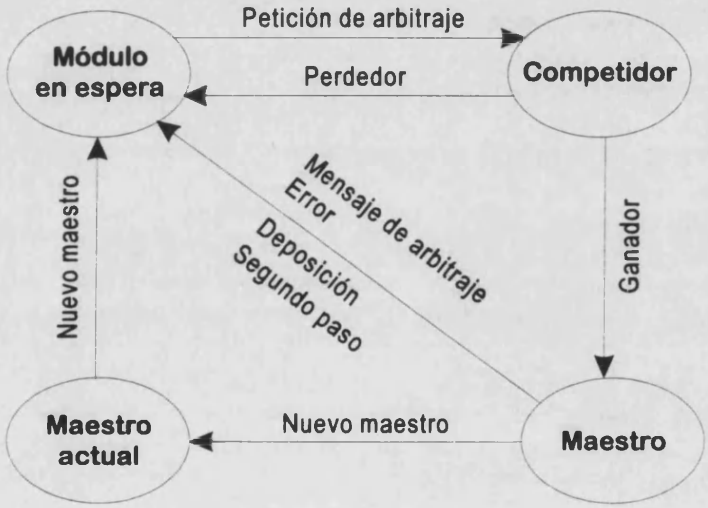


Figura 4.2: Estados en la adquisición del bus

En el esquema distribuido, cada módulo tiene número de arbitraje compuesto por dos partes: una parte alta con la prioridad del módulo, permitiendo tener hasta 256 niveles de prioridad y una parte baja única para cada módulo que decide en el caso de empate. El arbitraje tiene lugar en una o en dos fases dependiendo del tipo de competición. En cada paso el módulo pone en el bus un número de arbitraje de 8 bits,", indicando su prioridad, un bit de "*round-robing* para decidir en el caso de que varios módulos tengan la misma prioridad y su identificación. Si los módulos tienen prioridades diferentes, el ganador es el que tiene el mayor número de competición, en caso contrario, el ganador es el módulo que tiene el bit de "*round-robing*" activo. De esta forma se evita que para los módulos con la misma prioridad, siempre gane el bus el que tiene una dirección geográfica mayor. La Figura 4.2 representa el diagrama de control con los posibles estados de los módulos en el proceso de adquisición del bus.

Mediante el protocolo de arbitraje es posible enviar mensajes a todos los módulos del bus. Este mensaje consiste en un número de 8 bits. Para enviarlo se produce un arbitraje, pero poniendo todos los bits a 1 en la primera fase y luego indicando el número del mensaje en la segunda. Estos mensajes tienen una prioridad máxima y se suelen utilizar para enviar interrupciones o avisos a todos los módulos del sistema a la vez, como por ejemplo el fallo en la alimentación. El resto pueden ser definidos como se quiera.

En el esquema centralizado cada módulo tiene dos líneas de petición, una de concesión y una de inicio de rearbitraje (*preemption*), que fuerza a que se realice un nuevo arbitraje mientras el módulo que ha ganado está esperando a que el bus esté libre. La prioridad de cada módulo la guarda el árbitro central, y para cambiarla, los módulos deben de enviar un mensaje de arbitraje especial al árbitro central.

Para sincronizar los ciclos de arbitraje se utiliza un protocolo de arbitraje a tres hilos totalmente asíncrono. Esta característica es especialmente interesante en los sistemas tolerantes a fallos, puesto que es muy fácil detectar los errores en el protocolo. Para avanzar un paso en el protocolo se debe esperar a que todos los módulos se hayan puesto de acuerdo, por lo tanto mediante la programación de temporizadores de guardia se puede detectar si el sistema se queda detenido en alguna fase.

4.2 Transacciones tolerantes a fallos

Cuando un módulo ha conseguido ser dueño del bus, tiene la capacidad de poder realizar transferencias de datos. Las transferencias se realizan en paralelo con el arbitraje del bus, por lo tanto no existe pérdida de rendimiento cada vez que se realiza un nuevo arbitraje. El protocolo paralelo de transferencia de datos permite el intercambio de datos de cualquier tamaño siguiendo un protocolo asíncrono para asegurar la independencia de la tecnología. Las características del protocolo permiten de nuevo la detección de errores mediante el uso de temporizadores de guardia sobre las líneas de protocolo. Si falla el maestro de la transacción, el esclavo detectará el problema y viceversa.

En Futurebus+ una transacción se divide en tres fases: Una primera fase de conexión, en la que el maestro y los esclavos, deciden el tipo de transacción que va a tener lugar, una fase de transmisión, en donde se envían los datos de la forma en la que se acordó previamente y por último, una fase de desconexión, donde se comprueba el estado de la transacción y se termina con la transferencia.

Futurebus+ también soporta un protocolo paralelo síncrono de transferencia de datos llamado modo empaquetado. Estas transferencias son de longitud fija (1, 2, 4, 8 ... palabras). La sincronización se realiza mediante el uso un reloj interno en cada módulo con lo que las prestaciones de este protocolo son mucho mejores que las del asíncrono.

Las transacciones pueden se de tipo conectado (*connected*) o divididas (*split*). En las primeras, la transferencia de datos tiene lugar durante un único ciclo de posesión del bus, mientras que las segundas, en la fase de conexión, el esclavo informa al maestro que la transacción ha de ser dividida, entonces el maestro deja libre el bus, y cuando el esclavo tiene el dato, coge el bus y se lo manda a quien se lo pidió, evitando de esta forma el tener que esperar cuando un módulo es lento. La tolerancia a fallos se logra realizando operaciones de reintento para subsanar los errores transitorios que se puedan producir.

Para poder soportar los protocolos de coherencia de caches, existe otro tipo de operación en el bus paralelo, es lo que se llama Intervención. En ocasiones hay módulos que quieren leer determinadas posiciones de memoria, pero en un sistema basado en cache, puede ocurrir que no sea la memoria quien posea la información válida, sino que sea otro módulo quien posea una copia más reciente, en este caso, este módulo con la copia más reciente debe suplir los datos requeridos en ves de la memoria. Para ello existe esta operación especial en la que cualquier módulo con una copia más actualizada que la memoria puede intervenir en la transacción y así entregar el dato.

Otra característica del bus paralelo es su facilidad para bloquear las transacciones y conseguir así que sean indivisibles. Para ello existen comandos especiales de bloqueo que permiten realizar varias transacciones y operaciones en una o varias posesiones del bus, sin que se produzcan conflictos con otros módulos. Las operaciones de bloqueo básicas son *Swap*, *Fetch and Add*, y *Compare and Swap*.

4.3 Otros protocolos distribuidos

Futurebus+ especifica un protocolo de coherencia de caches que se basa en la monitorización de la actividad del bus (*bus snooping*). Cada nodo comprueba la actividad del bus y reacciona a las transacciones coherentes cambiando el estado de sus bloques en la cache y suministrando datos o información al sistema. Este esquema está distribuido en todos los nodos con lo que de nuevo se evitan los puntos únicos de fallos.

En cualquier sistema con características de tiempo real es necesaria la inclusión de un reloj en tiempo real para mantener información temporal consistente y precisa [24]. Un único reloj no puede suministrar la suficiente información a todos los módulos del sistema sin incluir un gran tráfico por el bus, aparte de que se constituiría en un punto único de fallo. Futurebus+ soluciona este problema permitiendo que cada módulo posea su propio reloj. Para realizar la sincronización de todos los relojes del sistema se utiliza un protocolo [20], que mediante el uso de los registros del CSR logra que todos los relojes marquen la misma hora. Además la posibilidad de forzar el abandono del bus de un módulo (*preemption*) en el proceso de arbitraje, también lo hace interesante para aplicaciones de tiempo real [24].

Otro protocolo distribuido utilizado es la elección del monarca. El monarca de un sistema es el módulo de proceso encargado de ejecutar las rutinas de inicialización del sistema, que en el caso del FB+ puede ser cualquier módulo, independientemente de la posición física que ocupe en el bus. En el algoritmo de puesta en marcha del sistema se produce una competición entre todos los módulos que pueden llegar a ser monarcas para conseguir el bus y comenzar la inicialización. Es evidente que un módulo averiado no participará en este proceso y que siempre habrá algún módulo en el sistema capaz de llevar a cabo la inicialización.

A pesar de los numerosos mecanismos de tolerancia a fallos que implementa FB+, la cobertura de detección de fallos de bus no es del 100 %. En [20][21] se describe la posible implementación de un temporizador de guardia, que se basa en la existencia de un registro en cada módulo en donde se mantiene información temporal para que los módulos comprueben y notifiquen la existencia de errores.

5 Inserción en vivo

En los sistemas de alta disponibilidad cuando se detecta una avería, el sistema pasa por una fase de localización y de aislamiento del fallo, para después reconfigurarse y continuar funcionando en modo degradado. De esta forma el sistema funciona hasta que falla el último módulo. Para aumentar al máximo la disponibilidad del sistema, es necesario que los módulos se puedan reparar. FB+ suministra en su estándar la información física y lógica necesaria para realizar la extracción y la inserción de los módulos en vivo, sin necesidad de detener el sistema [19].

Para evitar los posibles errores en los datos derivados de los transitorios producidos al extraer o al insertar una placa en el sistema, FB+ define varios niveles de inserción dependiendo del grado de la avería producida y de la pérdida de rendimiento que queramos tolerar. El nivel de mayor seguridad (nivel 0) no permite ninguna extracción o inserción en vivo, evitando de este modo los posibles fallos producidos por los transitorios y manteniendo la integridad de los dato; sin embargo es el de menos utilidad puesto que no admite reparaciones. El nivel 1 permite la inserción en vivo, pero mientras se realiza se detiene cualquier operación que se vaya a realizar por el bus, dejando al sistema parado durante segundos o incluso minutos. El nivel 2 permite la inserción en vivo, pero sólo cuando se realizan una serie de transacciones seguras, todas asíncronas que toleran mejor los transitorios en el bus. Por último está el nivel 3 en donde en cualquier momento se puede extraer o insertar un módulo.

En FASST se permite cualquier nivel de inserción, por lo tanto es tarea del administrador del sistema decidir cuando se va a poder extraer un módulo.

6 Interrupciones

Futurebus+ define dos formas diferentes de producir interrupciones, las interrupciones dirigidas y los mensajes de arbitraje. Las primeras se generan mediante la escritura en un registro del módulo que se quiere interrumpir. Tienen la desventaja de que para producir la interrupción, el módulo que interrumpe debe de ser maestro de bus y realizar una transacción de escritura por el bus de datos para producir la interrupción, con el consiguiente gasto de ancho de banda.

Los mensajes de arbitraje por el contrario se envían a través del bus de arbitraje y por lo tanto no suponen ninguna carga para el bus de datos, a parte de ser un método mucho más rápido. Tienen el problema que la interrupción la reciben todos los módulos conectados al sistema, sin embargo, se puede configurar el módulo a través de dos registros en el CSR para que sólo atienda a ciertos mensajes de arbitraje.

7 Conexión con el Futurebus+

Como se ha podido contemplar en los párrafos anteriores, Futurebus+ es un estándar muy complejo, por lo que realizar el diseño de un interfaz que soporte todas las características que define resulta bastante costoso. A parte del espacio necesario para implementar la lógica de conexión, se tendrían que utilizar dispositivos programables, con el consiguiente aumento del coste y la pérdida de prestaciones, sin tener en cuenta la posibilidad de cometer errores. Por ello se hace necesario la utilización de integrados específicos controladores de bus que minimicen al máximo todos estos problemas.

Los integrados de Texas Instruments [30] cubren todas las operaciones del bus, desde el arbitraje a las transacciones de datos. La filosofía de estos integrados es traducir las transacciones del Futurebus+ a transacciones de un bus más simple que es fácilmente accesible desde un procesador estándar.

El arbitraje está controlado por el TFB2010 en el caso de que sea distribuido, y por el TFB2011 en el caso en que se prefiera un arbitraje centralizado*. Ambos pueden convivir en un mismo bus presentando la ventaja de poder reconfigurar el sistema en el caso de que el árbitro central falle.

* Texas Instruments no comercializó ningún integrado de este tipo, por lo tanto sólo se soporta en Futurebus+ el arbitraje distribuido si se utiliza el chipset de Texas.

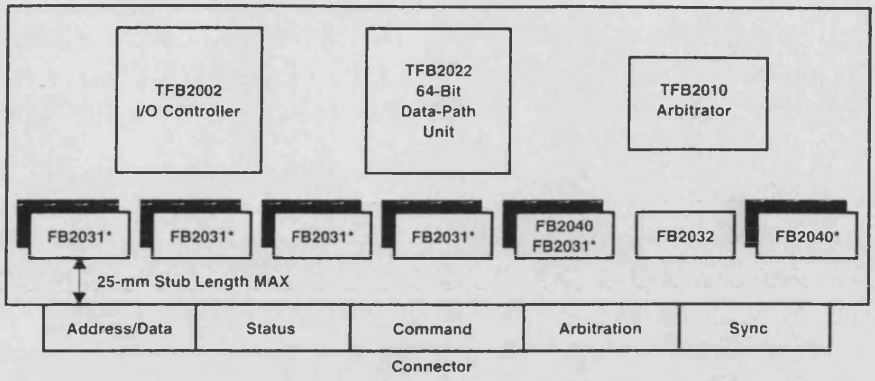


Figura 4.3: Esquema de conexión con FB+ utilizado en FASST

El protocolo paralelo de transmisión de datos está implementado en el controlador de E/S TFB2002 (I/O Controller) que está conectado con la unidad de datos TFB2022 (Data Path Unit) que hace de buffer entre el bus y el módulo. Estos integrados llevan asociados un conjunto de registros compatibles con la arquitectura CSR que permite su programación. Tienen la desventaja de que no incorporan el protocolo de coherencia de caches. Para conectarse al bus se utilizan un conjunto de drivers de tecnología BTL que también implementa Texas Instruments. En la Figura 4.3 se puede ver un diagrama de conexión de los integrados.

En respuesta a la necesidad de utilizar arquitecturas más simplificadas que eliminen la necesidad de utilizar montaje superficial por ambas caras del circuito impreso, Texas Instruments desarrolló una serie de transceivers de 18 bits, la serie FB16xx. Estos integrados están diseñados tanto con las capacidades de almacenamiento de datos de los de los transceivers de direcciones y datos FB2031 y la lógica separada de E/S de tecnología TTL que implementa el transceiver de control. La Figura 4.4 representa un diagrama de bloques de una placa de Futurebus+ que utiliza los nuevos transceivers.

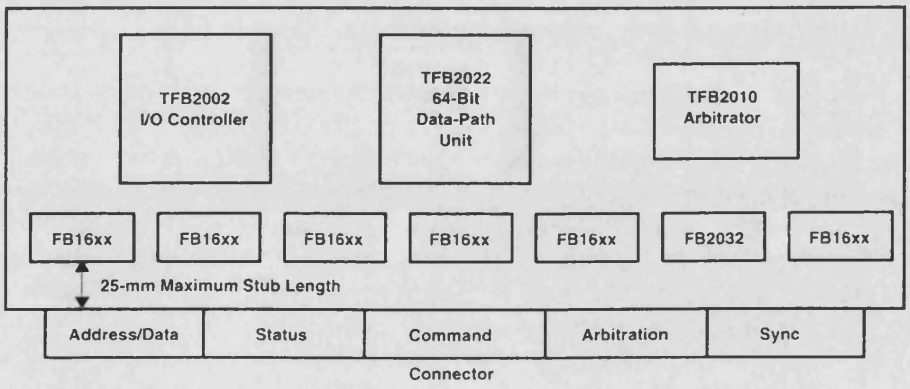


Figura 4.4: Simplificación al usar los nuevos transceivers

Actualmente existen otros integrados que simplifican el diseño. El TFB2003 es un controlador de l protocolo paralelo que integra actualiza al TFB2002B Futurebus+ I/O controller y al TFB2022A Futurebus+ data path unit. Al combinar estos dispositivos en un único integrado se incrementa el rendimiento (hasta 140 Mbyte/seg), el espacio disponible en la placa y la fiabilidad del sistema y a su vez se decrementa el coste. En el momento en que se diseñaron los módulos de proceso del sistema FASST no existía ni el TFB2003 ni los nuevos transceivers.

Los integrados de Natiònal Semiconductor permiten implementar los requerimientos completos de arbitraje distribuido del protocolo IEEE 896.1. Estos son el DS3875 Futurebus+ Arbitration Controller, el DS3885 Futurebus+ Arbitration Transceiver y el DS3884A BTL Handshake Transceiver. Para el arbitraje centralizado National tampoco a implementado un dispositivo específico y recomienda un diseño a propósito.

Por último la casa Philips Semiconductors también comercializa los drivers BTL para implementar la conexión con el Futurebus+ (FB2031, FB2033, FB2040 y FB2041). A diferencia de las dos compañías anteriores, también dispone del integrado FB2012A que es un árbitro de protocolo centralizado de bus. El integrado dispone de dos niveles de prioridad, cada uno con 14 líneas, tal como se describe en la especificación de Futurebus+ 896.1.

8 Conclusiones

En contraposición, decir que este estándar está siendo sustituido por otro más barato, y más simple. Sin embargo siempre resulta interesante conocer al predecesor.

Como contribución, comentar que el demostrador hace uso intensivo de las interrupciones dirigidas para enviar los mensajes de "estoy vivo" entre los módulos. Se ha implementado de esta manera ya que las interrupciones dirigidas se envían como mensajes de arbitraje por el bus de arbitraje, por lo tanto no se ve decrementado en absoluto el ancho de banda del bus.

En este capítulo se ha descrito brevemente las características del bus Futurebus+ escogido para ser implementado en el prototipo del sistema FASST. Simplemente se trata de una pequeña introducción, necesaria para poder comprender los mecanismo de tolerancia a fallos implementados que se ha considerado oportuno incluir porque:

- Es un estándar poco utilizado en Europa, y por tanto poco conocido
- Sobresale de los demás buses por sus características de fiabilidad y de tolerancia a fallos, así como por las prestaciones que ofrece y las facilidades de tiempo real

En el Apéndice II, sin embargo se describe con muchos más detalles técnicos el estándar IEEE 896.1.

La conclusión que se obtiene es que la tecnología avanza rápidamente, dejando en pocos meses a un diseño totalmente obsoleto. Como se ha podido ver, ya en 1995 se sustituyeron los dos integrados que manejan las transferencia de FB+ por uno sólo, más económico y fiable y poco después ni siquiera los drivers de bus eran los mismos. Si a esto añadimos las dificultades obtenidas en su día, años atrás, para poder conseguir en Europa la distribución del interfaz para Futurebus+ de Texas Instrumens nos damos cuenta que el tiempo al mercado debe ser mínimo para poder sacar un producto competente. Es para desanimarse pero la conclusión es que o conoces a alguien que te suministre información preliminar y te das prisa en implementar el sistema o no haces nada.

Por si esto fuera poco, las últimas tendencias es abandonar al Futurebus+ para dejar paso a un nuevo bus, el FMEbus. Este bus es un sucesor del Futurebus+ de alto rendimiento y alta fiabilidad pero optimizado para conseguir un bajo coste. Desarrollado en el comité P896 del IEEE Computer Society BASC (Bus Architecture Standars Committee) el FMEbus (Fast Multimedia Enhanced Bus) empezó a desarrollarse en 1995. El objetivo principal fue crear un bus tecnológicamente competitivo que tuviera la suficiente fiabilidad y rendimiento para hacer frente a las necesidades actuales de comunicación y multimedia.

Capítulo 5

Técnicas de evaluación de la fiabilidad

La evaluación de la garantía de funcionamiento de un sistema informático es necesaria para demostrar el buen funcionamiento de los distintos mecanismos de tolerancia a fallos que éste incluye y consecuentemente poder poner cierta confianza en el servicio que proporciona el sistema. Para realizar la evaluación se pueden utilizan técnicas diferentes de modelado dependiendo de las suposiciones de operación del sistema, como la posibilidad de la degradación y la posibilidad de la reparación. Se puede distinguir entre dos grupos principales: el primero lo forman los sistemas que se reconfiguran a medida que se van produciendo las averías. El sistema va poco a poco perdiendo prestaciones hasta que se terminan los recursos disponibles y se avería. El segundo grupo lo forman los sistemas en donde a parte de permitir la configuración, los módulos averiados se pueden reparar, volviendo el sistema a su estado inicial. En la Figura 5.1 se puede ver una taxonomía de las diferentes técnicas de modelado empleadas [50].

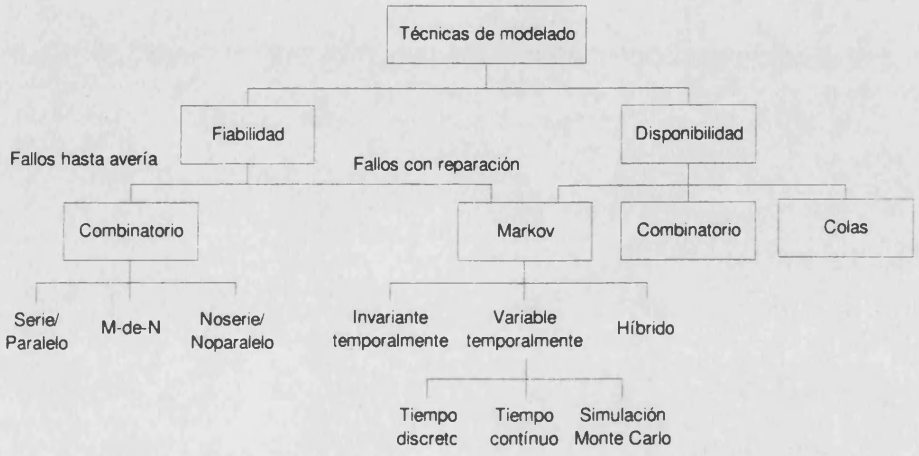


Figura 5.1: Clasificación de las técnicas de modelado

Los criterios de evaluación se pueden considerar de acuerdo a dos técnicas básicas: el modelado determinista, utilizado en los sistemas sin reparación y el modelado probabilístico utilizado en los demás sistemas. La Tabla 5.1. resume los distintos tipos de modelado.

MODELO	CRITERIO
Determinista	Se sobrevive a al menos k fallos
Probabilístico	*Funciones* Función tasa de fallos $z(t)$ Fiabilidad $R(t)$ Tiempo de misión $MT(r)$ Tasa de reparación μ Disponibilidad $A(t)$ *Parámetros* Tiempo medio al fallo (MTTF) Tiempo medio de reparación (MTTR) Tiempo medio entre fallos (MTBF) Cobertura *Medidas comparativas* Diferencia en la fiabilidad $R_2(t) - R_1(t)$ Ganancia de fiabilidad $R_2(t) / R_1(t)$ Incremento del tiempo de misión $MT_2(r) / MT_1(r)$ Indice de incremento en la fiabilidad $\log R_{viejo} / \log R_{nuevo}$

Tabla 5.1: Criterios de evaluación de la fiabilidad

La técnica de modelado más simple es el *modelo determinista* en donde se trata de averiguar cuál es el número mínimo de fallos de los componentes que se pueden tolerar sin que el sistema falle, para determinar la bondad del sistema. Este modelo no es muy práctico, puesto que los componentes muy fiables pagan las consecuencias de los poco fiables, ya que hay que duplicarlos también. Se utiliza para comprobar la no existencia de puntos únicos de fallos. El *modelo probabilístico* basado en las tasas de fallos, tasas de averías y tasas de reparación relativas de los componentes, es el método que más se utiliza para evaluar la fiabilidad/disponibilidad del sistema.

Normalmente se suele modelar en cuatro niveles distintos. El de más alto nivel es el *nivel de sistema*, en donde se considera al sistema como una caja negra. Se recogen estadísticas sobre los fallos y se sugiere un modelo que represente estas estadísticas de forma lo más aproximada posible. Para realizar un modelo correcto se requiere gran cantidad de datos. El *nivel de módulo* es el siguiente nivel, en donde el sistema queda dividido en varios módulos que fallan de forma independiente. Se utiliza para modelar por ejemplo sistemas redundantes. Más abajo queda el *nivel de puertas* seguido del *nivel de componentes* que se utiliza para representar los sistemas en donde existe redundancia en los niveles más bajos (transistores, diodos, resistencias, ...).

1 Modelos probabilísticos

A continuación se van a introducir una serie de definiciones básicas necesarias para aplicar estas técnicas de evaluación.

1.1 Función tasa de fallos y Función de fiabilidad

La función tasa de fallos define la evolución temporal del número medio de fallos que sufre un dispositivo por unidad de tiempo. Se cuantifica usualmente en fallos ocurridos cada millón de horas, ó en fallos cada 10^9 horas. Se utiliza para comparar la fiabilidad de sistemas o componentes.

Normalmente se supone que la tasa de fallos es constante con el tiempo y cuando esto ocurre se le denomina λ.

Si considero una variable aleatoria *T* que me indica el tiempo que tiene que transcurrir hasta que se produzca el próximo fallo de un sistema o componente, y llamo F(t) a la función de distribución de dicha variable, puedo definir la *función de fiabilidad R(t)* como:

$$R(t) = 1 - F(t) = P[T > t]$$

La *función de fiabilidad* es por tanto la probabilidad de que el componente esté funcionando correctamente dentro del intervalo $[t_0, t]$, suponiendo que en el instante t_0 estaba funcionando correctamente. Si tenemos *N* componentes idénticos que están funcionando desde el instante t_0, y llamamos $N_f(t)$ al número de componentes que han fallado en el instante *t* y $N_0(t)$ al número de componentes que están funcionando, y suponemos que cuando falla un componente permanece en este estado indefinidamente, la fiabilidad de los componentes en el instante *t* viene dada por la expresión:

$$R(t) = \frac{N_0(t)}{N} = \frac{N_0(t)}{N_0(t) + N_f(t)}$$

que es la probabilidad de que el componente haya sobrevivido en el intervalo $[t_0, t]$. La probabilidad de que el componente no haya sobrevivido para este intervalo viene dada por la *función de no-fiabilidad* o de *probabilidad de avería* [36] *Q(t)* (*unreliability*)

$$Q(t) = \frac{N_f(t)}{N} = \frac{N_f(t)}{N_0(t) + N_f(t)} = 1 - R(t)$$

Si derivamos la función de fiabilidad con respecto al tiempo obtenemos:

$$\frac{dR(t)}{dt} = -\frac{1}{N}\frac{dN_f(t)}{dt}$$

Si diferenciamos el número de componentes que han fallado en un instante *t* con respecto al tiempo, obtenemos la tasa instantánea de fallo de esos componentes. Como en el instante *t* aún quedan $N_0(t)$ componentes buenos, podemos obtener la función tasa de fallos o función de riesgo del componente (*hazard function, hazard rate* o *failure rate function*) mediante la siguiente expresión:

$$z(t) = \frac{1}{N_0(t)}\frac{dN_f(t)}{dt}$$

Las unidades de esta expresión se miden en fallos por unidad de tiempo. *z(t)* se puede expresar también en función de la fiabilidad o de la no-fiabilidad mediante la siguiente expresión

$$z(t) = \frac{1}{N_0(t)}\frac{dN_f(t)}{dt} = \frac{1}{N_0(t)}\left(-N\frac{dR(t)}{dt}\right) = -\frac{\frac{dR(t)}{dt}}{R(t)} = \frac{\frac{dQ(t)}{dt}}{1 - Q(t)}$$

Si diferenciamos la función de no-fiabilidad con respecto al tiempo obtenemos la función de densidad de fallos.

La función tasa de fallos se define también en la teoría probabilística como:

$$z(t) = \frac{f(t)}{1 - F(t)}$$

que se demuestra en [62] que aplicando técnicas de renovación de procesos se puede mostrar que $z(t\text{-}\tau)\Delta t$ es la probabilidad condicional de que el fallo número n ocurra en el intervalo infinitesimal $[t\ ,\ t + \Delta t]$, dado que el fallo número $(n - 1)$ ocurrió en el instante τ. Por tanto la función de riesgo muestra cómo la probabilidad instantánea de fallos evoluciona con el tiempo.

La función tasa de fallos es claramente dependiente del tiempo, sin embargo la experiencia muestra que para los componentes electrónicos existe un periodo de tiempo en donde es prácticamente constante. La relación existente entre la función de la tasa de fallos y el tiempo se le llama curva de la bañera y representa la evolución de la tasa de fallos a lo largo de la vida de un dispositivo. En ella se diferencian tres zonas [37] como se puede ver en la Figura 5.2:

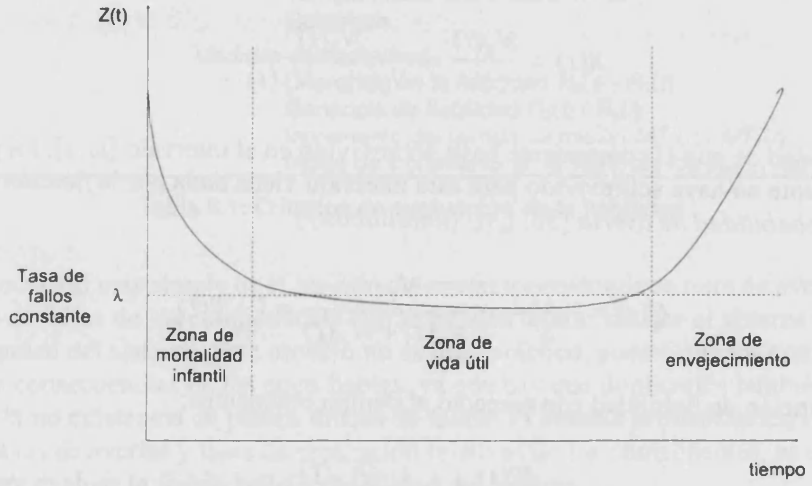


Figura 5.2: Evolución de la tasa de fallos con el tiempo

- Zona de mortalidad infantil: Los componentes cuando son fabricados tienen un alto riesgo de no funcionar correctamente debido a defectos de fabricación. Los fabricantes, antes de suministrar sus productos los ponen en condiciones extremas para acelerar los mecanismos de producción de fallos.
- Zona de vida útil: Se asume que la tasa de fallos es constante, de valor λ, y se expresa en número de fallos por hora ó cada 10^6 ó 10^9 horas.
- Zona de envejecimiento: Los materiales de los componentes se han desgastado por su uso y por tanto la tasa de fallos aumenta con el tiempo.

Durante la fase de vida útil, el sistema está dando el servicio más fiable a los usuarios. Normalmente se evita suministrar componentes en la fase de mortalidad infantil, eliminando los componentes defectuosos, y utilizar componentes en la fase de envejecimiento, reemplazando los componentes antes de que pasen a esta fase.

Para tener una función de la tasa de fallos, podemos integrar la tasa de fallos instantánea de la siguiente ecuación

$$\frac{dR(t)}{dt} = -z(t)R(t) \qquad \text{con lo que se obtiene} \qquad R(t) = e^{-\int z(t)dt}$$

Si asumimos que el sistema se encuentra en el periodo de vida útil, podemos considerar que la tasa de fallos es constante, por lo tanto la solución a la ecuación diferencial anterior es una función exponencial de parámetro λ

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

A la relación que existe entre la fiabilidad y el tiempo se le denomina la *ley de los fallos exponencial* y dice que para una función constante de la tasa de fallos, la fiabilidad varía exponencialmente como una función del tiempo.

Existen algunos casos en donde no es apropiado usar esta tasa de fallos constante, como por ejemplo en la detección de los fallos en un diseño software. Los usuarios al utilizar la aplicación van descubriendo los errores y los van corrigiendo, por lo tanto la fiabilidad del software se va incrementando con el tiempo y como consecuencia la función de la tasa de fallos decrece. Para realizar estos modelos se utilizan funciones tasa de fallos que varía con el tiempo basadas en la distribución de Weibull [39]

$$z(t) = \alpha\lambda(\lambda t)^{\alpha-1}$$

donde α y λ son constante que controlan la variación de la función tasa de fallos con el tiempo. Si α es 1, $z(t)$ es la constante λ. Si α es mayor que 1, $z(t)$ aumentará con el tiempo y si α es menor que 1, $z(t)$ disminuirá con el tiempo.

Para esta tasa de fallos, la función de fiabilidad $R(t)$ se calcula resolviendo la ecuación diferencial

$$\frac{dR(t)}{dt} = -z(t)R(t) = -\alpha\lambda(\lambda t)^{\alpha-1}R(t)$$

y viene dada por

$$R(t) = e^{-(\lambda t)^{\alpha}}$$

esta expresión se puede verificar calculando la derivada de $R(t)$

$$\frac{dR(t)}{dt} = -e^{-(\lambda t)^{\alpha}}\alpha\lambda(\lambda t)^{\alpha-1} = -\alpha\lambda(\lambda t)^{\alpha-1}e^{-(\lambda t)^{\alpha}} = -z(t)R(t)$$

La función tasa de fallos es fácil de medir si tenemos en cuenta la historia de la vida del componente ya que se calcula teniendo en cuenta el tiempo entre los fallos.

La técnica más común para estimar la tasa de fallos de un componente es el estándar MIL-HDBK-217 [61], desarrollado por el Departamento de Defensa de los Estados Unidos de América. El objetivo del estándar fue el de desarrollar un modelo para calcular la tasa de fallos de los componentes electrónicos usando los datos experimentales obtenidos del análisis de los fallos en los modelos reales.

El modelo predice que la tasa de fallos constante de un circuito integrado viene dada por la expresión:

$$\lambda = \pi_L \pi_Q (C_1 \pi_T + C_2 \pi_E) \pi_P \qquad \text{fallos por millón de horas}$$

siendo:

- π_L (factor de aprendizaje): Representa la madurez del proceso de fabricación utilizado para producir el circuito. Obtiene valores entre 10 y 1 dependiendo si el proceso es nuevo o si está suficientemente probado.

- π_Q (factor de calidad): Representa la cantidad de tests que ha pasado el circuito antes de ser vendido. El nivel más bajo indica que no se le ha pasado ningún test. Este factor oscila entre 1 y 300 dependiendo de la clase del circuito. Para aplicaciones militares se utiliza la clase A y B que tienen un factor de calidad de 1 y 2 respectivamente. Los componentes comerciales de gran calidad pertenecen a la clase C que se corresponde con un factor 16 y por último está la clase D, con un factor de 150, que se corresponde con los componentes estándar que están herméticamente sellados.

- π_T (factor de temperatura): Su valor es una función de la tecnología con la que esté hecho el circuito, de la temperatura ambiente, del encapsulado y de la disipación de potencia que presenta. Se utilizan dos ecuaciones dependiendo si el circuito es lineal o bipolar, siendo T_j la temperatura de la unión del integrado en grados Celsius.

$$\pi_T = 0.1e^{-8121\left(\frac{1}{T_j+273}-\frac{1}{298}\right)} \qquad \text{Circuito lineal}$$

$$\pi_T = 0.1e^{-4794\left(\frac{1}{T_j+273}-\frac{1}{298}\right)} \qquad \text{Circuito Bipolar}$$

- π_E (factor ambiental): Es una función que depende de lo ruidoso que sea el entorno donde funciona el circuito. Varía desde 0'2 en los componentes instalados en una sala de computador con aire acondicionado, hasta 6 para los instalados al aire libre y e incluso 10 para los componentes de un misil.

- π_P (factor de terminales): Es una función del número de terminales (*pins*) que tiene el encapsulado del circuito integrado. Varía desde 1.0 para los circuitos LSI de menos de 26 patas, 1.1 para los circuitos entre 26 y 64 patas y 1.2 para los de más de 64 patas.

- Los factores de complejidad son una función del número de puertas en los circuitos lógicos, del número de transistores en los circuitos lineales y del número de bits en las memorias.

$100 < N_g < 1300$	$N_g < 100$	Circ. Lineales	ROM	RAM
$C_1 = 0.0187e^{0.00471N_g}$	$C_1 = 0.0129N_g^{0.677}$	$C_1 = 0.00056N_t^{0.763}$	$C_1 = 0.0114B^{0.603}$	$C_1 = 0.00199B^{0.603}$
$C_2 = 0.013e^{0.00423N_g}$	$C_2 = 0.00389N_g^{0.359}$	$C_2 = 0.0026N_t^{0.547}$	$C_2 = 0.00032^{0.646}$	$C_2 = 0.00056^{0.644}$

N_g = Nº de puertas del IC
N_t = Nº de transistores del IC
B = Nº total de bits de la memoria

Actualmente existen un gran número de programas comerciales que permiten calcular la tasa de fallos de un sistema. Utilizan un modelo de bloques serie-paralelo del mismo y a partir de la descripción del

sistema y calculando por separado la tasa de fallos de los componentes basándose en este estándar, calculan la tasa de fallos global.

1.2 Cálculo de la tasa de fallos del módulo de proceso del Sistema FASST

Se ha utilizado el programa *Relex for Windows (v. 5.3)*, de Innovative Software Designs, Inc. para calcular de forma teórica la tasa aproximada de producción de fallos en el módulo de proceso del sistema FASST. Este software utiliza el estándar referenciado en el punto anterior, el MIL-HDBK-217F.

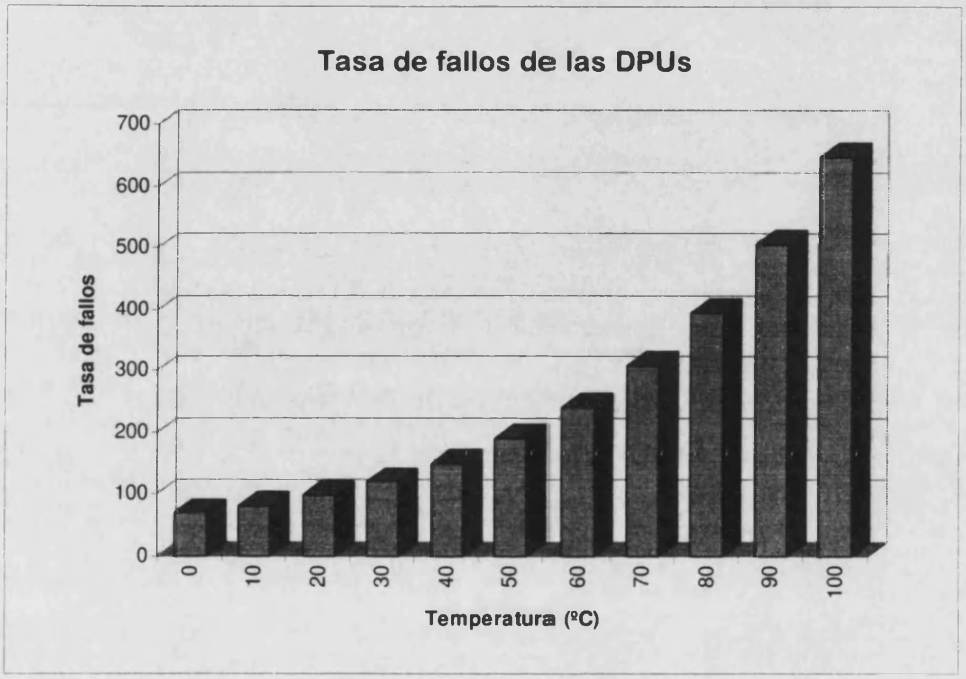


Figura 5.3: Gráfica de la tasa de fallos del sistema FASST

El tiempo medio de producción de fallos se ha calculado mediante un diagrama de bloques en serie, puesto que cualquier componente del módulo es imprescindible para el correcto funcionamiento del mismo. No se han tenido en cuenta todos los componentes que influyen en la fiabilidad (resistencias, ventiladores, circuito impreso, tipos de soldaduras, tipos de encapsulado), sino que sólo se han considerado los elementos del sistema sometidos a un mayor estrés y, por lo tanto, los más significativos. La Tabla 5.2 recoge los elementos que se han considerado para el cálculo de la tasa de fallos del sistema.

Cantidad	Nombre	Función
2	486DX2	Procesadores
4	CY7C188	Memoria RAM
5	29F010	Memoria ROM
2	EPM7032	Votador
1	EPM7064	FB+ Interface
4	EPM7128	Comparadores
1	EPM7192	CSR Interface
1	TFB2002	DPU FB+
1	TFB2010	Arbitro FB+
1	TFB2022	IOC FB+

Tabla 5.2: Componentes utilizados en el calculo de la tasa de fallos

Como se puede observar en la gráfica, la tasa de fallos depende en gran medida de la temperatura de funcionamiento del sistema, por lo tanto también es interesante medir las condiciones ambientales de funcionamiento.

1.3 Función del tiempo de misión

Esta función $MT(r)$, calcula el tiempo en el cual la fiabilidad del sistema se reduce por debajo de un cierto nivel r. Se aplica en sistemas que necesitan un tiempo de vida mínimo, bien debido a la imposibilidad o el coste excesivo de la reparación, o bien debido a que el sistema sufre de intervalos fijos de mantenimiento en donde se deja al sistema en su estado original, es decir "*como nuevo*".

La relación existente entre $R(t)$ y $MT(r)$ es la siguiente:

$$R[MT(r)] = r \qquad MT[R(t)] = t$$

Para un sistema con una tasa de fallos constante λ, la función del tiempo de misión del componente viene dada por la expresión

$$MT(r) = \frac{-\ln r}{\lambda}$$

1.4 Tiempo medio hasta la avería (MTTF)

El *MTTF* se define como el tiempo medio en el que trabajará un sistema antes de que se estropee por primera vez. Si tenemos N sistemas idénticos que empiezan a funcionar en el instante $t=0$ y medimos el tiempo en el que está funcionando cada uno de los sistema antes de fallar, la media de todas estas medidas constituye el *MTTF*. Si cada sistema i funciona durante un tiempo t_i antes de fallar, el *MTTF* viene dado por la expresión:

$$MTTF = \sum_{i=1}^{N} \frac{t_i}{N}$$

El *MTTF* se puede obtener mediante teoría probabilística calculando el valor medio del tiempo al fallo. De la teoría de probabilidad se sabe que el valor medio de una variable aleatoria X es:

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

donde *f(x)* es la función de densidad de probabilidad. Por lo tanto si llamamos *f(t)* a la función de densidad de fallos y consideramos sólo la parte positiva del intervalo de la integral, tenemos que:

$$MTTF = \int_{-\infty}^{\infty} tf(t)dt = \int_{0}^{\infty} t\frac{dQ(t)}{dt}dt$$

Integrando por partes y considerando que $\frac{dQ(t)}{dt} = -\frac{dR(t)}{dt}$ obtenemos la expresión del *MTTF*:

$$MTTF = \int_{0}^{\infty} t\frac{dQ(t)}{dt}dt = -\int_{0}^{\infty} t\frac{dR(t)}{dt}dt = \left[-tR(t) + \int R(t)dt\right]_{0}^{\infty} = \int_{0}^{\infty} R(t)dt$$

Si la función de fiabilidad obedece a la ley de fallos exponencial, tenemos que el *MTTF* es la inversa de la tasa de fallos del sistema:

$$MTTF = \int_{0}^{\infty} e^{-\lambda}dt = \frac{1}{\lambda}$$

por lo que se puede calcular la probabilidad de que en los sistemas con una tasa de fallos constante no exista ningún fallo antes de que se cumpla el *MTTF* como:

$$R(MTTF) = R\left(\frac{1}{\lambda}\right) = e^{-\lambda(1/\lambda)} = e^{-1} = 0.3678$$

1.5 Tiempo medio de reparación (MTTR)

Las actividades de reparación de un sistema no se modelan tan fácilmente como la evolución del proceso de producción de fallos, ya se basan en la propia experiencia de la reparación de equipos anteriores y además les afectan muchos más factores, como pueden ser la habilidad del operador, el tiempo que tarda el operador en desplazarse, la capacidad de diagnosis del sistema y la disponibilidad de repuestos. El cálculo de este parámetro es por tanto muy subjetivo.

El MTTR hace referencia al tiempo medio necesario para reparar un sistema. Es difícil de estimar por lo que su valor se calcula experimentalmente inyectando una serie de fallos al sistema y calculando el tiempo que se tarda en reparar el error. Si el fallo número *i* de una serie de *N* fallos requiere un tiempo t_i de reparación, el *MTTR* se estimará como

$$MTTR = \frac{\sum_{i=1}^{N} t_i}{N}$$

Normalmente se expresa como una tasa de reparación μ, que es el número medio de reparaciones que ocurren por unidad de tiempo (medido en horas).Por lo tanto obtenemos la relación

$$MTTR = \frac{1}{\mu}$$

La capacidad de mantenimiento (mantenibilidad) que tiene un sistema *M(t)* se define como la probabilidad de que un sistema que ha fallado sea reparado en un tiempo menor o igual que *t*. La

capacidad de mantenimiento de un sistema para un tiempo equivalente al *MTTR* y suponiendo una distribución exponencial para el tiempo de reparación vale:

$$M(MTTR) = M\left(\frac{1}{\mu}\right) = e^{-\mu(1/\mu)} = 1 - e^{-1} = 0.632$$

1.6 Tiempo medio entre fallos (MTBF)

Se define como el tiempo medio que transcurre entre dos fallos consecutivos en un sistema. Para calcularlo se debe de incluir también el tiempo necesario para reparar el sistema y para ponerlo de nuevo en funcionamiento. Si cada uno de los N sistemas está trabajando durante un tiempo T y al número de fallos encontrados en el sistema i le llamamos n_i, el número medio de fallos que ocurren lo calculamos como:

$$n_{avg} = \sum_{i=1}^{N} \frac{n_i}{N}$$

y por tanto el *MTBF* es:

$$MTBF = \frac{T}{n_{avg}}$$

Por lo tanto el *MTBF* es el tiempo total de operación del sistema, T, dividido por el número medio de fallos que ocurren durante T.

Si consideramos que al reparar un sistema, éste queda como nuevo, es decir, en las mismas condiciones que cuando se puso en funcionamiento por primera vez, existe una relación entre el *MTBF*, el *MTTR* y el *MTTF*, como se puede ver en la Figura 5.4.

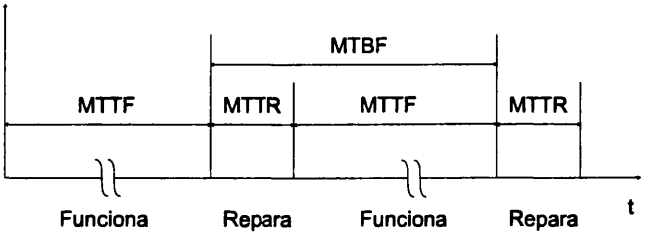


Figura 5.4: Relación entre el MTTF, el MTTR y el MTBF

Normalmente como el MTTF es mucho mayor que el MTTR, se utiliza la aproximación de que el MTTF es equivalente al MTBF.

1.7 Función de disponibilidad

Esta función es importante en el estudio de los sistemas tolerantes a fallos puesto que muchas empresas se preocupan más de que sea alta la probabilidad de que los sistemas estén en funcionamiento cuando sus clientes quieren usarlos (disponibilidad) de que estén mucho tiempo sin

fallar (fiabilidad). Se aplica a sistemas reparables, por lo que aunque un sistema tenga una alta disponibilidad, no significa que es muy fiable, puesto que puede haber sufrido un número considerable de averías. En este caso la tasa de reparación del sistema llega a ser una parte vital del diseño, por lo que se tiene que se intenta reducir el tiempo de detección y recuperación de la avería.

La función de disponibilidad define la probabilidad de que el sistema esté funcionando en un tiempo dado. Intuitivamente se puede calcular como el tiempo total en el que el sistema está operativo dividido por el tiempo total que transcurrió desde que se puso en marcha el sistema, lo que significa que mide básicamente el porcentaje del tiempo en que el sistema está funcionando.

Si la experiencia nos muestra que el número medio de fallos que experimenta un sistema a lo largo de su vida es *N*, el número total de horas en el que estará operativo será *N*MTTF*. Del mismo modo, el número de horas en las que el sistema está en reparación es *N*MTTR*. Por lo tanto, la disponibilidad media o disponibilidad en régimen estacionario (*steady-state*) será:

$$A_{SS} = \frac{N * MTTF}{N * MTTF + N * MTTR} = \frac{MTTF}{MTTF + MTTR}$$

Para un sistema simple con una tasa de fallos constante λ y una tasa de reparación constante μ, la disponibilidad del estado-estable se calcula como

$$A_{SS} = \frac{\mu}{\lambda + \mu}$$

1.8 Cobertura

Es un factor muy importante puesto que afecta en gran medida a la fiabilidad, seguridad y otros atributos del sistema. La cobertura es una medida de la capacidad del sistema para realizar la detección, la localización, el aislamiento, la contención y/o la recuperación de la avería*. Dependiendo del análisis que se quiera realizar se aplicará a un campo o a otro.

Cuando se están evaluando sistemas tolerantes a fallos, la cobertura de fallos se interpreta como una medida de la capacidad que tiene el sistema de recuperarse después de que se haya producido una avería, por lo tanto afecta a todos los niveles: la detección, la localización, el aislamiento y la recuperación del fallo.

La cobertura tiene dos significados principalmente: Uno cuantitativo que se utiliza en el modelado de la fiabilidad de sistemas redundantes, y que define la cobertura como la probabilidad condicional de que el sistema se recupere de forma satisfactoria ante la ocurrencia de algún tipo de fallo [40], y uno cualitativo que especifica la probabilidad de que el sistema detecte una clase particular de fallos, como por ejemplo los tipos de errores contra los que protege un esquema de redundancia en particular, como los códigos Hamming que detectan dos errores y recuperan uno.

---

* La detección de fallos (*fault detection*) es el proceso de reconocer que ha ocurrido un fallo. La localización de fallos (*fault location*) es el proceso de determinar dónde ha tenido lugar el fallo para poder implementar la recuperación adecuada. El aislamiento de fallo (*fault containment*)) es el proceso de aislar el fallo y el de prevenir que se propaguen sus efectos a lo largo del sistema. La recuperación de fallos (*fault recovery*) es el proceso de permanecer activo o el de volver a estar operativo mediante la reconfiguración ante la presencia de fallos.

Para calcular este parámetro hay que tener en cuenta lo que significa para el analista el término recuperación, ya que ésta depende en gran medida de la aplicación; puede significar, por ejemplo, la utilización de una configuración hardware que siempre esté funcionando (sistema telefónico) o por el contrario que nunca hayan pérdidas de datos o datos corruptos (como en los ordenadores que procesan transacciones bancarias). A parte de esto, el problema más grave es que no es fácil calcularla. Lo más fácil es realizar una lista con los posibles fallos que se considera que pueden ocurrir en el sistema y después hacer listas con los fallos que pueden ser detectados, los que pueden ser localizados, los que pueden ser aislados y los que el sistema puede recuperar. El factor de cobertura de detección de fallos se calcula de esta forma como el número de fallos detectado dividido por el número total de fallos.

Existen varios puntos que se deben destacar con respecto a la cobertura [41]:

- La estimación de la cobertura de fallos necesita la definición de los tipos de fallos que pueden ocurrir. Si tenemos una cobertura de 0.9, ésta carece de significado a no ser que se hayan identificado los tipos de fallos que se están considerando.

- Normalmente se asume que las coberturas de fallos son constantes a lo largo del tiempo.

2 Evaluación del software

Debido al alto coste del desarrollo y mantenimiento del software, son necesarios implementar diversos mecanismos para medir la calidad del mismo. Se estima que más del 90% del coste de computación se gasta en el software. Los problemas que presentan estos modelos son que asumen que la tasa de fallos del software es proporcional al número de errores de diseño que contiene, pero no tienen en cuenta que diferentes tipos de errores pueden contribuir de forma diferente a la tasa de fallos total. Si eliminamos un error grave podemos duplicar el tiempo medio hasta la avería, mientras que si eliminamos diez errores de implementación sin importancia, puede que no tengan ningún efecto considerable. Además, ningún modelo asume el hecho de que la tasa de fallos es proporcional a la carga del sistema.

2.1 Modelos en el dominio del tiempo

Estos modelos relacionan la fiabilidad del software con el número de errores presentes en el software en un momento determinado de su desarrollo. La eliminación de los errores de implementación debería incrementar el MTTF y la correlación de la historia de esta eliminación con la evolución temporal del MTTF, puede permitir la predicción de cuándo un valor para el MTTF puede ser alcanzado.

Las desventajas de estos modelos son que la corrección de los errores puede generar nuevos errores y que la infiabilidad del software puede ser causada no sólo por los errores de implementación sino también por los errores de especificación, caracterización y simulación durante el test de una carga típica.

Entre los modelos típicos destaca el de Shooman [51], que para estimar la fiabilidad del software tiene en cuenta el número de errores por instrucción en lenguaje máquina, presentes en un sistema software, después de T meses de depuración. La función tasa de fallos después de T meses de depuración se supone que es proporcional al número de errores que aún quedan en el sistema. La fiabilidad del sistema se calcula por tanto mediante la siguiente ecuación:

$$R(t) = e^{-CE(r,T)}$$

donde $E(r,T)$ es el número de errores que aún quedan en el sistema después de T meses de depuración y C es una constante de proporcionalidad.

El modelo de Jelinsky-Moranda [52] es un caso particular del anterior pero asume que el error que se descubre se elimina inmediatamente, decrementándose en una unidad el número de errores remanentes. Si suponemos que el tiempo de depuración que transcurre entre la detección de dos errores sigue una distribución exponencial, la función de densidad del tiempo en el que se descubre el error número i, medida desde el tiempo en el que se detecto el error i - 1 es:

$$p(t_i) = \lambda(i)\exp(-\lambda(i)t_i)$$

donde $\lambda(i) = f(N - i + 1)$ y N es el número de errores que había inicialmente.

En una extensión de este modelo [53], se supone que la tasa de errores no es sólo proporcional al número de errores sino también al tiempo que se gasta en la depuración, por lo tanto la probabilidad de descubrimiento del error se incrementa con el tiempo. Posteriores modelos [54] suponen que en un determinado periodo de tiempo se puede detectar más de un error y que no se hace ninguna corrección hasta que termina el intervalo de tiempo.

Todos estos modelos intentan predecir la fiabilidad de un sistema software después de un periodo de depuración y prueba. Después del análisis se llega a estimar el tiempo medio entre errores software (MTBSE). A medida que se va depurando el sistema se irá incrementando el MTBSE.

También existen otros modelos [55] en donde se simula la ejecución de un programa mediante una cadena de Markov, que va cambiando de estado a medida que se van ejecutando una serie de programas más pequeños. En [56] supone que el número más probable de errores que se habrán corregido en un tiempo t se basa en el modelado preliminar de la ocurrencia de error y de las tasas de reparación. En [57] se describe un modelo en donde la tasa de detección de errores durante un intervalo se asume que es proporcional al número de errores presentes durante ese intervalo.

2.2 Modelos en el dominio de los datos

El primer tipo de estos modelos se describe en [58]. Se basa en el hecho de que si se puede saber a priori el conjunto de datos sobre los cuales puede operar un programa, se puede realizar una estimación de la fiabilidad del mismo si se ejercita al programa con un subconjunto de esos datos.

2.3 Modelos axiomáticos

En estos modelos, la fiabilidad del software se postula para que obedezca una serie de leyes universales. La validez de estos modelos nunca ha sido probada, simplemente dan una estimación del número de errores presentes en un programa.

El modelo más conocido es el llamado "*Software Science*" [59]. En este modelo se realizan medidas cuantitativas sobre la dificultad del lenguaje de programación elegido, la complejidad del algoritmo, la claridad en la programación, el efecto de la modularización, el esfuerzo del programador y el tiempo de programación utilizado. Con estas medidas se intenta realizar una estimación del número de fallos que puede tener el programa. Más concretamente, se define el número de errores como:

$$B = K\left(\frac{V}{EO}\right)$$

donde K es una constante de proporcionalidad, V es el volumen de implementación de un algoritmo (depende del número de operaciones y de operandos) y EO es el número medio de discriminaciones mentales que hace el programador entre dos errores sucesivos (se estima de forma empírica que está sobre las 3000).

2.4 Otros modelos

Por último destaca el modelo presentado en [60]. Este modelo se basa en el hecho de que la ocurrencia de un error software se produce porque se combinan dos factores: un conjunto de datos de entrada determinados y un conjunto de caminos lógicos que pueden seguir los datos. Estos eventos son aleatorios e independientes del comportamiento anterior del sistema, por lo que se puede suponer que tienen una tasa de fallos constante.

El modelo se basa en las siguientes suposiciones:

1. El sistema inicialmente tiene N errores de diseño que pueden ser corregidos
2. La tasa de fallos software es constante para un número dado de errores de diseño presentes en el programa
3. Cuando se detecta un fallo se pasa a un estado de reparación. Si es debido a un fallo transitorio del hardware, el sistema se pone de nuevo a funcionar después de un instante de tiempo. Si es debido a un error en el software, se produce el mantenimiento que puede eliminar el fallo, introducir nuevos fallos o simplemente puede dejar al sistema en el mismo estado que se encontraba antes del error.

El modelo calcula la disponibilidad del sistema como una función del tiempo, llegando a la conclusión de que la disponibilidad tiende hacia la disponibilidad asintótica (disponibilidad del sistema cuando está libre de errores de diseño), a medida que se van eliminando los errores.

3 Modelado de la fiabilidad

La fiabilidad es uno de los atributos más importantes de un sistema. Casi todos los sistemas incluyen en sus especificaciones ciertos valores para la fiabilidad que se deben cumplir e incluso demostrar. Como hemos visto anteriormente la fiabilidad se puede calcular de forma experimental poniendo un número considerable de sistemas idénticos a funcionar y recogiendo medidas sobre sus tiempos de avería. Claramente existe el problema de la necesidad de tener muchos sistemas iguales y de la gran cantidad de tiempo necesaria para realizar las medidas. Es necesario por tanto utilizar otros métodos para poder estimar la fiabilidad.

Las técnicas más populares de análisis de la fiabilidad son las aproximaciones analíticas, en donde destacan los modelos combinatorios y los modelos de Markov.

3.1 Modelos combinatorios

Estos modelos usan técnicas probabilísticas que enumeran las diferentes formas en las que un sistema puede estar operativo. Para realizar una estimación de la fiabilidad de un sistema, se calcula la fiabilidad de todos sus componentes por separado y se calculan las probabilidades de ocurrencia de eventos que hacen que el sistema se averíe. Se realiza un análisis de los fallos del sistema hasta que este cae.

Para estimar la fiabilidad se divide al sistema en módulos, teniendo cada uno bien una probabilidad de estar trabajando, p_i, o bien una función de probabilidad que depende del tiempo de estar operativo, $R_i(t)$. El objetivo es derivar una probabilidad, p_{sys}, o función $R_{sys}(t)$, de la operación correcta del sistema. Se realizan los siguientes supuestos:

- Los fallos de los módulos son independientes.
- Una vez que el módulo falla, siempre va a generar resultados incorrectos.
- Se considera que el sistema ha fallado si no satisface unos requerimientos definidos previamente que indican el número mínimo de módulos que deben estar en funcionamiento.
- Una vez que el sistema entra en un estado de fallo, subsiguientes fallos no pueden devolverlo a un estado correcto.

Se suelen utilizar dos modelos: el modelo serie y el paralelo (ver Figura 5.5). En un sistema en serie se necesita que todos los elementos funcionen para que el sistema funcione correctamente. En un sistema paralelo, únicamente uno de los elementos tiene que funcionar correctamente para que el sistema realice su función.

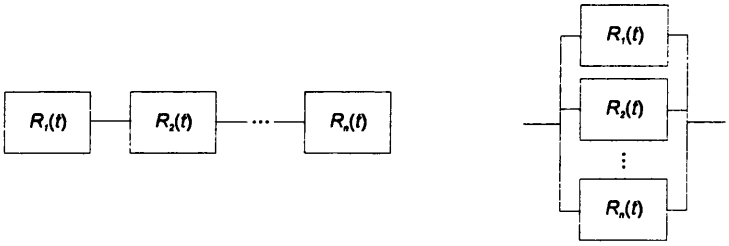


Figura 5.5: Esquema de un modelo serie y uno paralelo

Los sistemas serie no contienen redundancia alguna y por lo tanto la fiabilidad del sistema se calcula como la probabilidad de que funcionen todos los elementos. Si suponemos que $C_{iw}(t)$ representa el hecho de que el componente C_i está funcionando adecuadamente en el tiempo t, $R_i(t)$ la fiabilidad del componente C_i en el tiempo t, y $R_{serie}(t)$ la fiabilidad total del sistema serie, la fiabilidad en cualquier instante t para un sistema con N componentes se calcula como la probabilidad de que todos los componentes estén funcionando adecuadamente, es decir:

$$R_{serie}(t) = P\{C_{1w}(t) \cap C_{2w}(t) \cap \cdots \cap C_{Nw}(t)\}$$

suponiendo que todos los eventos, $C_{iw}(t)$ son independientes, tenemos:

$$R_{serie}(t) = R_1(t)R_2(t)\cdots R_N(t) = \prod_{i=1}^{N} R_i(t)$$

Si todos los componentes del sistema cumplen la ley de fallos exponencial (se considera el tiempo de producción de fallos como una variable aleatoria que sigue una distribución exponencial) y suponemos que cada componente tiene una tasa de fallos constante λ_i, la fiabilidad del sistema viene dada por:

$$R_{serie}(t) = e^{-\lambda_1 t}e^{-\lambda_2 t}\cdots e^{-\lambda_N t} = e^{\sum_{i=1}^{N} \lambda_i t}$$

En el sistema paralelo, por el contrario, sólo se necesita a uno de los N elementos para que funcione. La no-fiabilidad del sistema se puede calcular como la probabilidad de que todos los elementos fallen. Si suponemos que $C_{if}(t)$ representa el hecho de que el componente C_i haya fallado en el instante t, $Q_i(t)$ la no-fiabilidad del componente C_i en el tiempo t, y $Q_{paralelo}(t)$ la no-fiabilidad del sistema paralelo, tenemos:

$$Q_{paralelo}(t) = P\{C_{1f}(t) \cap C_{2f}(t) \cap \cdots \cap C_{Nf}(t)\}$$

suponiendo que todos los eventos, $C_{if}(t)$ son independientes, tenemos:

$$Q_{paralelo}(t) = Q_1(t)Q_2(t)\cdots Q_N(t) = \prod_{i=1}^{N} Q_i(t)$$

La fiabilidad del sistema se puede calcular por tanto como:

$$R_{paraleo}(t) = 1 - Q_{paralelo}(t) = 1 - \prod_{i=1}^{N} Q_i(t) = 1 - \prod_{i=1}^{N} (1 - R_i(t))$$

En estos sistemas se supone que los fallos ocurren de forma independiente, sin embargo, esta suposición no es muy exacta cuando los fallos se producen debido a la influencia de los agentes externos. Por lo tanto se tendrá que tener cierta precaución a la hora de utilizar estos modelos.

Los sistemas M-de-N son una generalización del sistema paralelo ideal. Para que el conjunto funcione, al menos M del total de N elementos replicados deben funcionar. Los sistemas triple modular redundantes son un ejemplo de un sistema 2-de-3. La expresión de la fiabilidad de estos sistemas se calcula como:

$$R_{M-de-N}(t) = \sum_{i=0}^{N-M} \binom{N}{i} R_m{}^{N-i}(t)(1 - R_m(t))^i$$

A veces se utiliza un diagrama de *éxito* para representar los modos de operación de un sistema (ver Figura 5.6), que no es directamente reducible a un sistema serie-paralelo. Cada camino desde la entrada a la salida representa un estado diferente de funcionamiento del sistema.

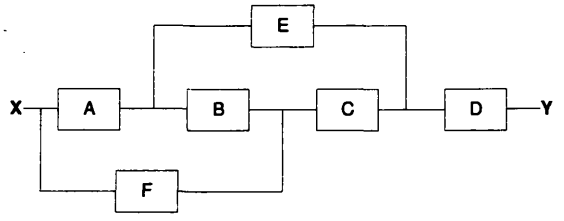


Figura 5.6: Diagrama de éxito del sistema

La fiabilidad del sistema se calcula dividiendo el modelo en dos y sumando las fiabilidales de los dos subsistemas condicionadas a que un elemento funciona o está averiado. Si el cálculo es demasiado complejo, se puede hallar una aproximación de la fiabilidad poniendo todos los posibles caminos en paralelo (a este nuevo esquema se le conoce como un *diagrama de bloques de fiabilidad*).

3.2 Modelos de Markov

La aplicación de modelos combinacionales para realizar la evaluación de un sistema presenta varias desventajas:

1. No es fácil modelar sistemas complejos de forma combinacional.
2. La inclusión de la cobertura en estos modelos es bastante difícil.
3. No contemplan el proceso de reparación de los sistemas.

Por estos motivos es por lo que amenudo se utilizan Modelos de Markov, también llamados Cadenas de Markov para evaluar la fiabilidad y la disponibilidad de los sistemas tolerantes a fallos. En estos modelos se utilizan dos conceptos principalmente: los estados y las transiciones entre estados.

El estado del sistema representa la descripción del sistema en un instante de tiempo determinado. Para hacer un estudio de la fiabilidad, en cada estado tendremos una combinación distinta de los módulos que han fallado y de los que no. Las transiciones entre estados gobiernan los cambios de estado que ocurren en un sistema y se expresan mediante probabilidades, como la probabilidad de fallo, la cobertura de fallos, y la probabilidad de reparación. A medida que pasa el tiempo y se van produciendo los fallos y las recuperaciones, el sistema va evolucionando desde un estado a otro. En los modelos de tiempo discreto las transiciones entre estados ocurren a intervalos fijos y tienen asignada una probabilidad para cada posible transición. En los modelos de tiempo continuo las transiciones entre estados ocurren a intervalos aleatorios y tienen asignada una tasa de transición. En los modelos de fiabilidad las tasas de transición se corresponden con las funciones tasa de fallos y tasa de reparación afectadas por un factor de cobertura.

Los modelos de Markov se basan en que la probabilidad de transición de un estado dado depende únicamente de ese estado. Para los procesos de Markov de tiempo continuo, la cantidad de tiempo que se está en un estado no influye sobre la distribución de probabilidad del próximo estado ni sobre la distribución de probabilidad de tiempo de permanencia en el estado actual. Estas características hacen a los modelos de Markov muy representativos para la suposición que se hace en la teoría de la fiabilidad de que las tasas de fallos son constantes.

Si suponemos un sistema triple modular redundante, como muestra la Figura 5.7, éste empieza en el estado (111) y cuando falla uno de los módulos se pasa al estado (110), (101), o (011) dependiendo del módulo que ha fallado. Los estados que muestra la se pueden dividir en tres grupos: el estado perfecto en donde todos los módulos funcionan correctamente, los estados en donde ha fallado un único módulo y los estados de avería del sistema en donde han fallado los suficientes módulos para hacer que el sistema falle.

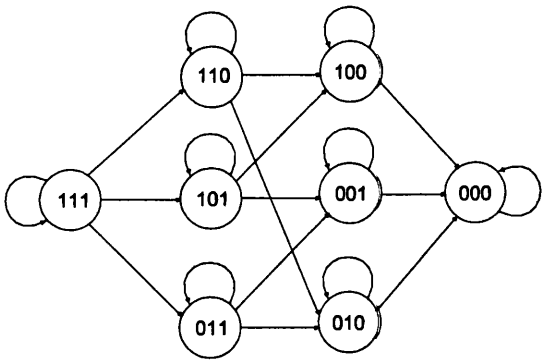


Figura 5.7: Modelo con Cadenas de Markov de un sistema TMR

Si consideramos un modelo de Markov discreto en el tiempo, cada transición entre estados lleva asociada una probabilidad de transición. Estas probabilidades se recogen en una matriz de transiciones A cuyos elementos p_{ij} son las probabilidades de transición desde el estado i al j. El modelo se resuelve mediante un conjunto de ecuaciones lineales que se basan en la matriz de probabilidades de transición. Estas ecuaciones se definen como:

$$\vec{P}(k+1) = \vec{P}(k)A$$

La matriz de probabilidades de transición del paso n (A^n)contiene las probabilidades de transición de un estado a otro justo después de que hayan pasado n intervalos de tiempo.

Para calcular las probabilidades de transición supondremos que cada módulo sigue la ley de fallos exponencial con una tasa de fallos constante λ. La probabilidad de que un módulo haya fallado en algún momento $t + \Delta t$, dado que estaba operativo en el instante t viene dada por:

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

siendo:

$$1 - e^{-\lambda\Delta t} = 1 - \left[1 + (-\lambda\Delta t) + \frac{(-\lambda\Delta t)^2}{2!} + \cdots\right] = (-\lambda\Delta t) - \left(\frac{(-\lambda\Delta t)^2}{2!}\right) - \cdots$$

Para valores de Δt tales que $\lambda\Delta t > 1$ la expresión se reduce a:

$$1 - e^{-\lambda\Delta t} \approx \lambda\Delta t$$

Por lo tanto, usando la ley de fallos exponencial, la probabilidad de que un componente falle en el periodo de tiempo Δt, dado que el componente estaba operacional en el tiempo t, es aproximadamente $\lambda\Delta t$.

En la Cadena de Markov reducida del sistema TMR, ver Figura 5.8, el estado 3 corresponde al estado perfecto, el estado 2 corresponde a la agrupación de los estados en donde se ha averiado un módulo y el estado F corresponde, a la agrupación de los estados con dos módulos averiados y al estado en donde se han averiado los tres módulos. Las probabilidades de transición se derivan de la suma de todas las probabilidades de transición a un estado reducido.

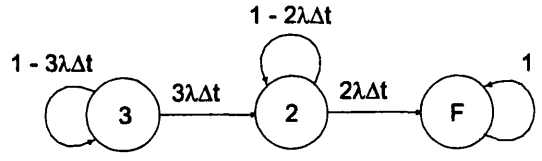


Figura 5.8: Modelo de un sistema TMR mediante una Cadena de Markov reducida

Para extraer la matriz de transición entre estados, nos basamos en la propiedad de las Cadenas de Markov de que la probabilidad de estar en cualquier estado, s, en un tiempo $t + \Delta t$ depende tanto de la probabilidad de que el sistema se encuentre en el instante t en un estado en donde exista una transición al estado s, como de la probabilidad de que ocurra esa transición. Por ejemplo, la probabilidad de estar

en el estado 3 en el sistema TMR, depende de la probabilidad de estar en el estado 3 en el instante t, y de la probabilidad de que el sistema pase desde el estado 3 otra vez al 3. Las probabilidades de permanecer en cada uno de los estados vienen dadas por tanto por:

$$p_3(t+\Delta t) = (1-3\lambda\Delta t)p_3(t)$$
$$p_2(t+\Delta t) = 3\lambda\Delta t p_3(t) + (1-2\lambda\Delta t)p_2(t)$$
$$p_F(t+\Delta t) = 2\lambda\Delta t p_2(t) + p_F(t)$$

Estas ecuaciones se pueden escribir en forma de matriz:

$$\begin{bmatrix} p_3(t+\Delta t) \\ p_2(t+\Delta t) \\ p_F(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1-3\lambda\Delta t & 0 & 0 \\ 3\lambda\Delta t & 1-2\lambda\Delta t & 0 \\ 0 & 2\lambda\Delta t & 1 \end{bmatrix} \begin{bmatrix} p_3(t) \\ p_2(t) \\ p_F(t) \end{bmatrix}$$

de una forma más compacta como:

$$\bar{P}(t+\Delta t) = A\bar{P}(t)$$

siendo $\mathbf{P}(t)$ el vector de probabilidad de estados en el instante t y $\mathbf{A}$ la matriz de transición de estados. $t = 0$, tenemos que $\mathbf{P}(\Delta t) = \mathbf{AP}(t)$, para $t = 2\Delta t$, tenemos que $\mathbf{P}(2\Delta t) = \mathbf{AP}(\Delta t)$, por lo tanto, extendiendo este resultado a $n\Delta t$:

$$\bar{P}(n\Delta t) = A^n\bar{P}(0)$$

En el modelo de Markov para tiempo continuo, las transiciones ya no ocurren a intervalos fijos sino que pueden tener lugar en cualquier instante de tiempo. Este modelo se deriva del anterior utilizando un paso de tiempo del intervalo que tienda a cero. Las ecuaciones anteriores se pueden ver como un conjunto de ecuaciones diferenciales utilizando el límite cuando Δt se aproxima a 0 (ecuaciones de Chapman-Kolmogorov):

$$\frac{dp_3(t)}{dt} = 3\lambda p_3(t)$$
$$\frac{dp_2(t)}{dt} = 3\lambda p_3(t) - 2\lambda p_2(t)$$
$$\frac{dp_F(t)}{dt} = 2\lambda p_2(t)$$

cuya solución es (utilizando la transformada de Laplace):

$$p_3(t) = e^{-3\lambda t}$$
$$p_2(t) = 3e^{-2\lambda t} - 3e^{-3\lambda t}$$
$$p_F(t) = 1 - 3e^{-2\lambda t} + 2e^{-3\lambda t}$$

De esta forma se puede obtener la función de fiabilidad del sistema como uno menos la probabilidad de estar en el estado de avería.

3.3 Modelos con Redes de Petri Estocásticas

Una red de Petri [65][64] es un grafo que contiene dos tipos de vértices: los lugares, representados por círculos y las transiciones representadas por líneas o barras. Los arcos conectan los lugares y las transiciones y pueden ser normales, representados por flechas, o inhibidores, representados por líneas terminadas en un pequeño círculo. En cualquier instante, los lugares de la red pueden contener marcas, representadas por pequeños círculos.

Las transiciones se disparan cuando están conectadas a un lugar con alguna marca mediante un arco normal y no existe ningún arco inhibidor que conecta esa transición con un lugar marcado. Inmediatamente se elimina una marca de cada lugar de origen y se añade a cada lugar de salida. Una Red de Petri Estocástica Extendida asocia a cada transición una distribución de un tiempo de disparo, que añade un retardo al disparo desde el momento en que se habilita la transición y se dispara si al final de ese tiempo la transición sigue habilitada. Además se pueden incluir probabilidades de elección de una transición u otra.

El ejemplo de la Figura 5.9 modela un sistema multiprocesador tolerante a fallos con P procesadores y M módulos de memoria. El sistema funcionará siempre que exista al menos un procesador y un módulo de memoria activos y además existe una posibilidad de reparación que es compartida por todos los componentes.

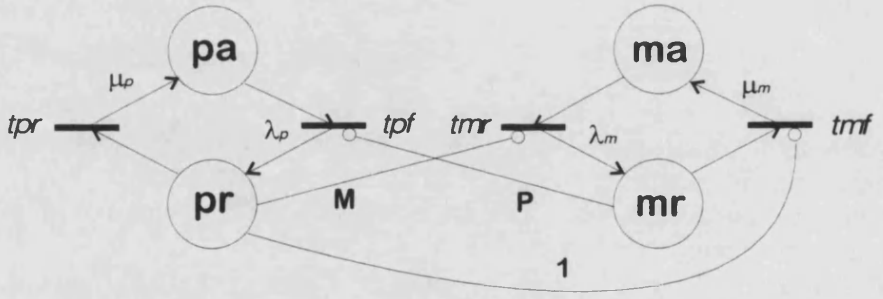


Figura 5.9: Modelo con una Red de Petri Estocástica Generalizada

Existe una marca para cada procesador en el estado de *procesador activo* (*pa*) y para cada elemento de memoria en el estado de *memoria activa* (*ma*). Si falla un procesador se dispara la transición de *fallo del procesador* (*tfp*) y una marca pasa al estado de *procesador en espera de ser reparado* (*pr*). La reparación del procesador se representa por el disparo de la transición *procesador reparado* (*tpr*) pasando una marca de nuevo al estado *pf*. Los arcos inhibidores representan la suposición de que si el sistema está averiado, debido a que todos los procesadores o todas las memorias han fallado, los demás componentes que quedan no pueden fallar puesto que el sistema no está en marcha. Las tasas de fallos y de reparación tanto para los procesadores como para las memorias vienen representadas respectivamente en el modelo por λ_p, μ_p , λ_m y μ_m.

4 Modelado de la seguridad

La cobertura de fallos es uno de los parámetros más importantes a tener en cuenta cuando se realiza la evaluación de un sistema tolerante a fallos, puesto que tiene un gran impacto en la fiabilidad, seguridad y muchos otros atributos del sistema [38].

La incorporación de la cobertura en el análisis de un sistema con Cadenas de Markov, nos permite modelar la seguridad del sistema, ya que esta mide la efectividad de las estrategias de cobertura de

fallos. Ahora cada estado de la cadena que represente un funcionamiento correcto tendrá dos transiciones a dos estados diferentes, uno de los cuales indica que la avería está prevista y el otro que no. En la Figura 5.10 se puede observar un sistema con un solo componente. Ahora tendremos tres estados: el estado de funcionamiento, el estado de fallo seguro y el estado de fallo no seguro.

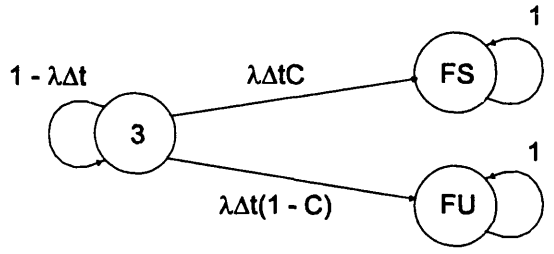


Figura 5.10: Modelo de un sistema con coberturas

La matriz de transiciones del sistema es la siguiente:

$$\begin{bmatrix} p_O(t+\Delta t) \\ p_{FS}(t+\Delta t) \\ p_{FU}(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1-\lambda\Delta t & 0 & 0 \\ \lambda\Delta tC & 1 & 0 \\ \lambda\Delta t(1-C) & 0 & 1 \end{bmatrix} \begin{bmatrix} p_O(t) \\ p_{FS}(t) \\ p_{FU}(t) \end{bmatrix}$$

La seguridad del sistema S(t) en un instante de tiempo viene dada por la probabilidad de estar en el estado de funcionamiento $p_O(t)$ más la probabilidad de estar en el estado seguro $p_{FS}(t)$. La solución temporal a las ecuaciones de la matriz es:

$$p_{FS}(t) = C - Ce^{-\lambda t}$$
$$p_{FU}(t) = (1-C) - (1-C)e^{-\lambda t}$$
$$p_O(t) = e^{-\lambda t}$$

La seguridad del sistema vendrá dada por la ecuación:

$$S(t) = p_O(t) + p_{FS}(t) = C + (1-C)e^{-\lambda t}$$

En el instante inicial, la seguridad del sistema es 1 y a medida que el tiempo se acerca al infinito, la seguridad se acerca a un valor constante definido como la seguridad en el estado estable del sistema. Este valor constante coincide con la cobertura de fallos que tiene el sistema:

$$S(\infty) = C$$

5 Modelado de la disponibilidad

Si suponemos que el sistema se puede reparar, éste puede pasar de un estado de avería o de un estado de funcionamiento en modo degradado a un estado de total funcionamiento. En los modelos con Cadenas de Markov de un sistema con reparación deberemos de incorporar una probabilidad de retorno de un estado a otro. Si suponemos que el sistema tiene una tasa de reparación constante en el tiempo llamada μ, que indica el número de reparaciones que se espera que tengan lugar en un cierto

periodo de tiempo, la probabilidad de que ocurra una reparación en el periodo de tiempo Δt, dado que el sistema había fallado en el instante t, será $\mu\Delta t$.

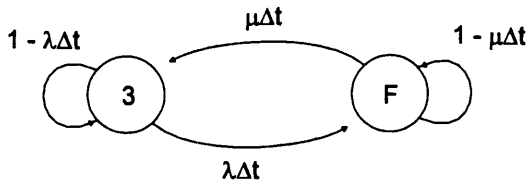


Figura 5.11: Modelo de un sistema reparable

Si consideramos un sistema reparable simple (ver Figura 5.11), las ecuaciones para este sistema serán:

$$\begin{bmatrix} p_O(t+\Delta t) \\ p_F(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1-\lambda\Delta t & \mu\Delta t \\ \lambda\Delta t & 1-\mu\Delta t \end{bmatrix} \begin{bmatrix} p_O(t) \\ p_F(t) \end{bmatrix}$$

Para hallar la disponibilidad del sistema, tendremos que calcular la probabilidad de que el sistema se encuentre en el estado de funcionamiento normal. Las ecuaciones diferenciales que describen el modelo de este sistema son las siguientes:

$$\begin{aligned} \frac{dp_F(t)}{dt} &= \lambda p_O(t) - \mu p_F(t) \\ \frac{dp_O(t)}{dt} &= \lambda p_O(t) + \mu 2 p_F(t) \end{aligned} \qquad [1]$$

Suponiendo que inicialmente el sistema se encuentra en el estado O, las soluciones de las ecuaciones anteriores son:

$$p_O(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)t}$$

$$p_F(t) = \frac{\mu}{\lambda+\mu} - \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)t}$$

Cuando el tiempo tiende a infinito (régimen estacionario), la probabilidad de estar en el estado de funcionamiento se calcula igualando a cero las derivadas de la ecuación [1]:

$$\left.\begin{aligned} 0 &= \lambda p_O(t) - \mu p_F(t) \\ 0 &= \lambda p_O(t) + \mu 2 p_F(t) \\ p_O(t) &+ \mu p_F(t) = 1 \end{aligned}\right\} \quad p_O(\infty) = \frac{\mu}{\lambda+\mu}$$

$$p_O(\infty) = \frac{\mu}{\lambda+\mu}$$

6 Modelos de cobertura de fallos

La mayoría de las averías que se producen en los sistemas de alta fiabilidad, no se deben a que no existe suficiente redundancia en el sistema para enmascarar los fallos, sino se deben principalmente a la carencia de cobertura de recuperación de los errores que se producen. Realizar modelos utilizando una cobertura constante a lo largo del tiempo supone una seria limitación, puesto que este parámetro varía con el tiempo.

El factor de cobertura influye notablemente sobre los parámetros característicos de fiabilidad del sistema. Si consideramos un sistema con n procesadores y tasas de reparación, como el que está modelado en la Figura 5.12, se puede hacer un estudio del MTTF del sistema en función de las tasas de fallos, la tasa de reparación, la cobertura y el número de procesadores del sistema. Para los sistemas no reparables (μ=0) el MTTF del sistema se incrementa al aumentar el factor de cobertura, sin embargo cuando introducimos una tasa de reparación μ para los fallos detectados, se puede llegar incluso a decrementar el MTTF [43].

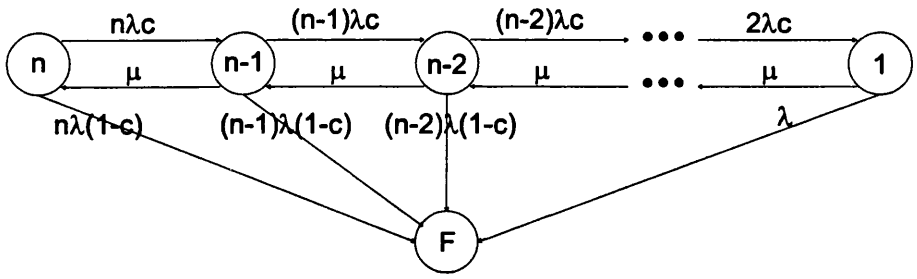


Figura 5.12: Diagrama de transición de estados de un sistema reparable

Otros modelos de cobertura incluyen una tasa de recuperación para los fallos no cubiertos en el sistema γ. Sin embargo los estudios muestran que el valor de la cobertura es el factor que realmente influye en la disponibilidad del sistema, teniendo poca importancia el número de procesadores que tenga el mismo y el valor descrito anteriormente.

Se pueden considerar también modelos de cobertura en donde este parámetro depende del estado del sistema. Cuando en un sistema se produce un fallo, siempre se gasta cierta cantidad de tiempo en la detección y recuperación del error. Si consideramos que este retraso sigue una distribución exponencial de parámetro δ, la cobertura de fallos viene dada por la siguiente expresión [44], donde i es el número de procesadores activos actual:

$$c_i = \frac{\delta}{\delta + (i-1)\lambda}$$

En donde la aparición de un segundo error mientras se está procesando el primero se traduce en un fallo de cobertura. Este tipo de modelos demuestra que la fiabilidad de un sistema puede decrecer a medida que se aumenta el número de procesadores debido a que aumenta la probabilidad de producción de errores coincidentes.

Descomposición comportamental

Al intentar modelar un sistema teniendo en cuenta la cobertura de fallos, resulta adecuado simplificar el número de estados teniendo en cuenta una cobertura que varía con el tiempo. Mediante este análisis se puede incluir la estimación de la fiabilidad del sistema teniendo en cuenta los fallos que ocurren en un periodo de tiempo pequeño (fallos cercano-coincidentes en el tiempo). Si tengo un mecanismo de detección de errores, reconfiguración y recuperación, es fácil entender que la ocurrencia de un nuevo fallo en el sistema mientras se está en medio de este proceso, puede mermar la fiabilidad del mismo, por lo tanto se debe incluir un factor de cobertura que dependa del tiempo para poder atender estas situaciones y que modele de forma más precisa el sistema.

El modelo de descomposición comportamental trata de representar al sistema mediante un diagrama de estados, en donde los diferentes estados sólo difieren entre sí en el número de elementos averiados. Las probabilidades de transición se calculan utilizando un determinado modelo de cobertura. Para calcular este modelo se suelen utilizar modelos semimarkovianos que permiten funciones de transición dependientes del tiempo.

Con el uso de esta técnica, claramente se obtienen dos importantes ventajas:

1. El número de estados del modelo se reduce drásticamente

2. Como el modelo se divide en otros tantos para poder realizar el cálculo de la cobertura de fallos del sistema, se resuelve el problema del tratamiento de modelos en donde las tasas de recuperación se miden en horas y los parámetros de cobertura se miden en milisegundos. En estos sistemas se puede llegar a tener errores graves debido al efecto del redondeo [63].

El problema que se plantea al desarrollar modelos complicados para la cobertura de fallos, es que cuando se incluyen en el modelo del sistema global, el modelo resultante es difícil de resolver, ya que es muy grande y contiene constantes temporales cuyos valores difieren entre si en gran medida (los tiempos de recuperación se expresan en milisegundos, mientras que los tiempos de fallos se expresan en miles de horas).

El modelo de cobertura CAST [45] combina en un único modelo la recuperación transitoria con la recuperación permanente. Los errores permanentes se producen con una tasa λ y los transitorios con una tasa τ y se detectan ambos con una probabilidad u. En el caso de que no se pudiera detectar el error, el sistema pasa a un estado de fallo. Después de detectar el fallo, se pasa a un estado en donde se intenta una recuperación transitoria del sistema por si el error detectado es de tipo transitorio. Con probabilidad l el sistema se recupera y en caso contrario se pasa a otro estado en donde se intenta localizar el fallo, con una probabilidad v y reconfigurar el sistema con una probabilidad w. En la Figura 5.13 se puede ver un gráfico detallado de este modelo.

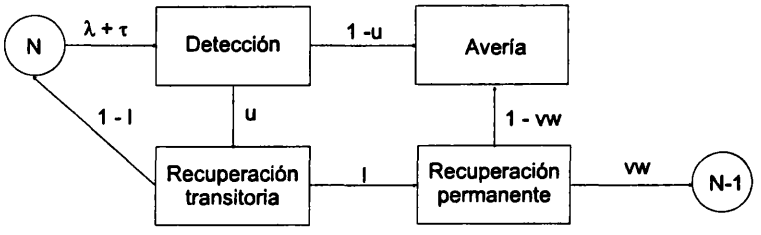


Figura 5.13: Modelo de recuperación CAST

Se pueden construir modelos más complejos en donde las etiquetas de los arcos representan las tasas de cambios entre estados en vez de indicar probabilidades. En el modelo CARE III [48][66] se entra en el estado activo (A) cuando se activa el fallo. Si el fallo es transitorio o intermitente, puede ir desde el estado activo al estado benigno (B), volviendo a pasar a A cuando cesa el fallo. En este estado el fallo no puede ser detectado ni producir un funcionamiento erróneo.

En el estado activo, sin embargo, el fallo puede ser detectado por los circuitos de autocomprobación con una tasa δ, antes de que afecte al servicio que proporciona el sistema, pasando al estado de detectado (D), o puede que produzca un error con una tasa ρ pasando al estado activo con error (E).

Dependiendo de la cobertura de detección de errores, el error que se ha producido será detectado con una probabilidad c y una tasa ε antes de que quede afectado el servicio suministrado por el sistema o por el contrario producirá la avería del sistema pasando al estado de fallo (F). Una vez detectado el error el sistema se podrá degradar con una probabilidad p poniendo al elemento averiado de nuevo en funcionamiento. En la Figura 5.14 se puede ver la representación de este modelo. La flecha punteada indica que la transición se produce de forma inmediata en el momento en el que se entra en el estado, pero con una probabilidad p.

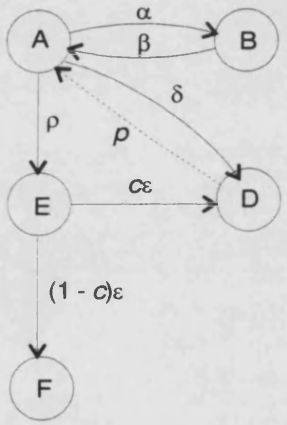


Figura 5.14: Modelo de error simple CARE III

En este modelo, los errores intermitentes hacen al sistema pasar rápidamente entre los estados A y B, siendo α la tasa con la que el error efectivo pasa a estado latente y β la tasa contraria.

Los modelos de fiabilidad de un sistema se pueden calcular realizándolos de forma jerárquica e incluyendo dentro de ellos los modelos vistos anteriormente. Para ello se calculará la probabilidad de pasar al estado en el que el sistema se recupera debido a la acción de un fallo permanente, c, mediante el análisis del modelo detallado y este parámetro se incluirá en el modelo global. De la misma manera, la probabilidad de avería del sistema vendrá dada por $(1\text{-}c)$. En el modelo de la Figura 5.12 se puede incluir la c obtenida del análisis de los modelos de cobertura anteriores.

El problema que se plantea al añadir parámetros de cobertura a los modelos es que la Cadena de Markov contiene tasas de transición muy rápidas (en el modelo de cobertura) y tasas de transición muy lentas (para el modelo global). Para obtener la solución al modelo se emplea la técnica de la *descomposición comportamental* [44].

En los modelos vistos anteriormente las tasas de transición eran independientes del tiempo global y del tiempo en el que se estaba en un estado. Se pueden utilizar modelos estocásticos más complejos para

representar el comportamiento ante el error. En los modelos semi-markovianos, las tasas de transición de los estados dependen del estado actual y del próximo estado y del tiempo que se gasta en el estado actual. Sin embargo, para combinar tiempos locales y globales se suelen utilizar modelos basados en Redes de Petri Estocásticas Extendidas (ESPN) [46].

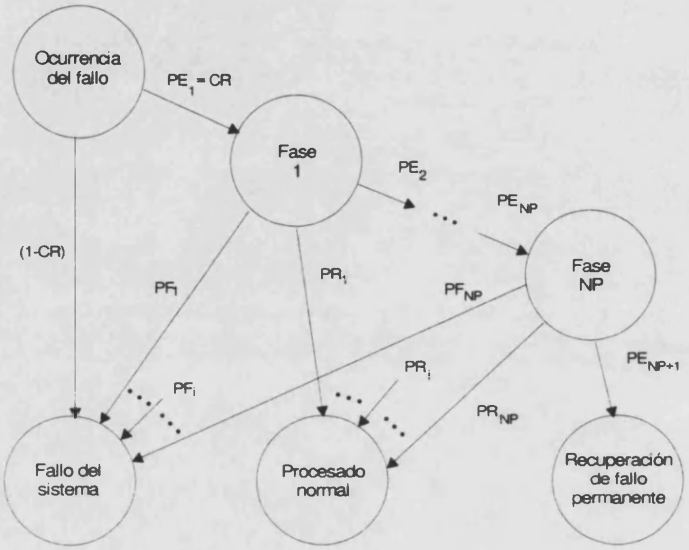


Figura 5.15: Modelo de recuperación de errores transitorios ARIES

El modelo de cobertura ARIES [47] se utiliza para contemplar los errores transitorios. En este modelo, como se puede ver en la Figura 5.15 se contemplan tres posibles salidas: fallo del sistema, procesamiento normal y recuperación de fallos permanente. En este modelo, 1 - CR es la probabilidad de que un error produzca un fallo inmediato en el sistema, PE_i es la probabilidad de que el sistema entre en la *i-ésima* fase de recuperación y PR_i es la probabilidad de que el sistema se recupere en la *i-ésima* fase de recuperación (después de no haber sido capaz de recuperarse en alguna de las fases anteriores). Si llamamos r a la probabilidad de recuperación transitoria del sistema (probabilidad de alcanzar el estado normal de procesado) :

$$r = PR_1 + PR_2 + \cdots + PR_{NP}$$

y c a la probabilidad de recuperación satisfactoria de un error permanente:

$$c = PE_{NP+1} * (\text{Cobertura del procedimiento de recuperacion permanente})$$

La probabilidad de un fallo de cobertura se obtiene por tanto por la expresión 1 - r - c.

En este análisis se supone que el sistema es capaz de recuperarse satisfactoriamente en una fase de recuperación transitoria con una probabilidad fija. Sin embargo, esta recuperación no se contempla para los errores transitorios que duren un tiempo considerable. La recuperación transitoria es satisfactoria si el error transitorio cesa en el momento que empieza la fase de recuperación y el sistema se puede recuperar en esta fase. En este modelo se supone que el tiempo que dura el error se comporta como una variable aleatoria con distribución exponencial de media D, y las fases de recuperación duran un tiempo fijo T_i. Si llamamos E_i a la probabilidad de recuperación de la fase i, podemos obtener las expresiones para la cobertura:

PR_i = Pr[Entrar en fase i] * Pr[Fase i con exito] * Pr[Error transitorio acabe antes de comenzar la fase i] =

$$= PE_{i-1} * E_i * (1 - e^{-\frac{(T_1+T_2+\cdots+T_{i-1})}{D}})$$

con

$$PE_i = PE_{i-1} - PF_i - PR_i$$

Podemos tener un modelo más general considerando la vida del error transitorio como una variable aleatoria de función no exponencial con una función de distribución $F_D(t)$. En este caso la expresión anterior resulta:

$$PR_i = PE_{i-1} * E_i * F_D(T_1 + T_2 + \cdots + T_{i-1})$$

Stiffler [48] propone también un modelo de error transitorio en donde el tiempo de vida del error está distribuido exponencialmente con parámetro α. Es más limitado que el modelo ARIES puesto que la duración de cada fase de recuperación transitoria es independiente y con una distribución idéntica. Sin embargo el número de ellas es variable.

El modelo de Stiffler se puede generalizar para los modelos de fallos permanentes o intermitentes, dependiendo de los valores de α y de β. Aplicando este modelo se puede obtener r (probabilidad de recuperación transitoria), c (probabilidad de recuperación permanente) y s (probabilidad de único punto de fallo).

Los modelos vistos hasta ahora no tienen en cuenta el hecho que durante la recuperación se puede producir un nuevo fallo. Podemos considerar, por ejemplo en los sistemas en tiempo real, que existe un límite temporal en las acciones de recuperación para considerarlas satisfactorias. En los sistemas de alta fiabilidad se puede considerar también los efectos de un segundo error (error *cercano-coincidente* en el tiempo) [49] que puede ocurrir durante el intento de recuperación de un error simple. Estos errores se dividen en dos clases: los de la clase dependiente cuya ocurrencia interfiere con la recuperación del primer error y los de la clase independiente cuya ocurrencia no afecta a la recuperación del primer error. Normalmente cuando se produce el error cercano-coincidente se producirá una avería del sistema.

Como conclusión se ve que el análisis de la fiabilidad de un sistema tolerante a fallos está fuertemente influenciado por la predicción precisa de la cobertura (probabilidad de que el sistema se pueda recuperar cuando existe un error). Los modelos simples son útiles para un análisis inicial de un prototipo de sistema, ya que muchos detalles del proceso de recuperación aún no son conocidos. El comportamiento no exponencial del modelo de manejo de errores se puede tratar mediante los procesos semi-markovianos, procesos de Markov no homogéneos o mediante Redes de Petri estocásticas extendidas [64].

7 Estudio de la influencia de la cobertura en la fiabilidad de un sistema

Es de vital importancia realizar un cálculo preciso de la cobertura de recuperación del sistema informático que se quiere evaluar, puesto que dependiendo de este factor varían considerablemente las características de la garantía de funcionamiento del sistema.

A continuación se presenta el estudio de la fiabilidad de un sistema tolerante a fallos con redundancia híbrida. El sistema está compuesto por un módulo principal y número variable de módulos de repuesto, todos iguales. Cada módulo i tiene una fiabilidad R_i y además en cada uno de ellos existen

implementados una serie de mecanismos de tolerancia a fallos que son capaces de detectar con una probabilidad c_i cuando el módulo se ha averiado. Inicialmente funciona un único módulo y los demás están ejecutando programas de autotest. Cuando el módulo principal falla, el sistema se reconfigura y uno de los repuestos pasa a ser el módulo activo. Este proceso se repite hasta que se agotan los módulos de repuesto, produciéndose en este caso la avería del sistema.

Si consideramos un sistema dúplex en donde sólo hay un repuesto, podemos calcular la fiabilidad del sistema R_{sys} como:

$$R_{sys} = R_1 + cR_2(1 - R_1)$$

El sistema estará en funcionamiento en el caso de que funcione el módulo R_1 o en el caso de que se haya podido detectar el error en el módulo R_1 y el módulo R_2 esté funcionando. Es interesante observar que si la cobertura de fallos es del 100 %, este sistema se corresponde con un sistema de dos módulos en paralelo y que si por el contrario la cobertura de fallos es nula, la expresión se reduce a la fiabilidad de un único módulo.

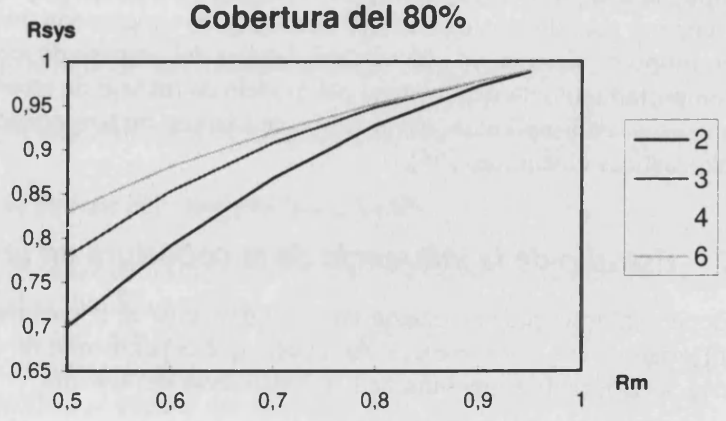
Si generalizamos la expresión anterior para un sistema con n módulos idénticos, y por lo tanto con la misma fiabilidad R_m, la fiabilidad del conjunto R_{sys} se obtiene como:

$$R_{sys} = R_m \sum_{i=0}^{n-1} c^i (1 - R_m)^i \qquad [2]$$

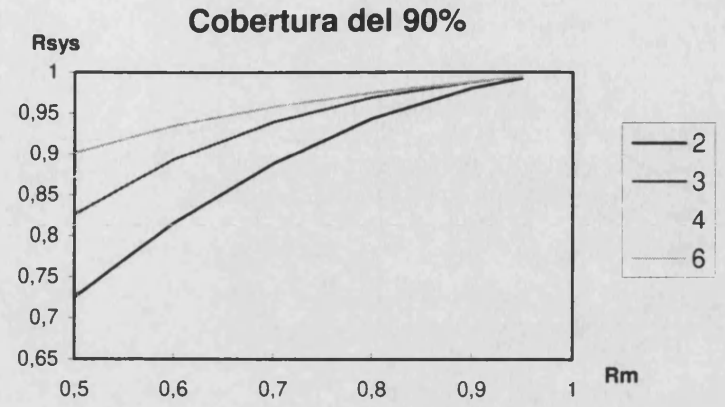
Es interesante realizar un análisis de la fiabilidad de un sistema con repuestos, variando el número de repuestos, la fiabilidad de los mismos y el factor de cobertura, para poder observar cómo influyen estos parámetros en la fiabilidad del sistema global. A continuación se muestran una serie de gráficas que contienen los resultados obtenidos.

En cada gráfica se han superpuesto 4 funciones que representan la fiabilidad global del sistema R_{sys} en función de las fiabilidades individuales de cada módulo R_m. Cada función representa un sistema con un número de módulos distinto. La cobertura de detección de errores de los módulos varía en cada una de las gráficas, siendo respectivamente del 80%, 90%, 95% y 99%, que son los valores habituales que se encuentran en los sistemas comerciales:

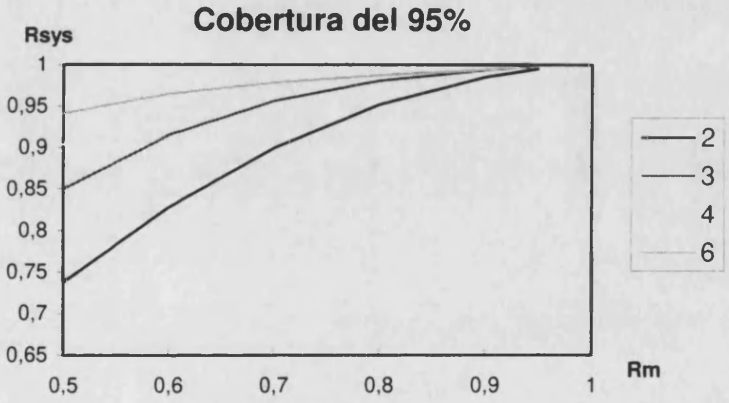
	Número de repuestos			
R_i	2	3	4	6
0,95	0,988	0,989	0,989	0,989
0,9	0,972	0,977	0,978	0,978
0,8	0,928	0,948	0,951	0,952
0,7	0,868	0,908	0,917	0,920
0,6	0,792	0,853	0,873	0,881
0,5	0,7	0,78	0,812	0,829



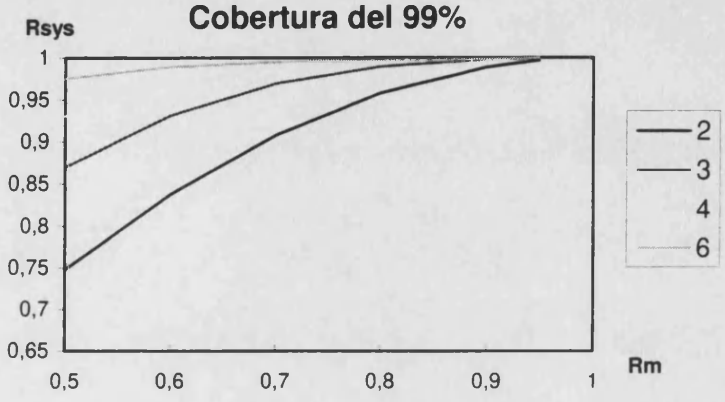
	Número de repuestos			
R_i	2	3	4	6
0,95	0,992	0,994	0,994	0,994
0,9	0,981	0,988	0,988	0,989
0,8	0,944	0,969	0,974	0,975
0,7	0,889	0,940	0,953	0,958
0,6	0,816	0,893	0,921	0,935
0,5	0,725	0,826	0,871	0,901



	Número de repuestos			
R_i	2	3	4	6
0,95	0,995	0,997	0,997	0,997
0,9	0,985	0,993	0,994	0,994
0,8	0,952	0,980	0,986	0,987
0,7	0,899	0,956	0,972	0,978
0,6	0,828	0,914	0,947	0,964
0,5	0,737	0,850	0,903	0,941

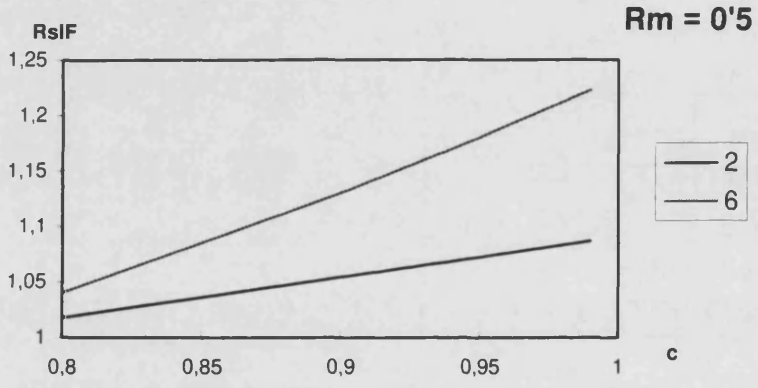


	Número de repuestos			
R_i	2	3	4	6
0,95	0,997	0,999	0,999	0,999
0,9	0,989	0,997	0,998	0,998
0,8	0,958	0,989	0,995	0,997
0,7	0,907	0,969	0,987	0,995
0,6	0,837	0,931	0,968	0,989
0,5	0,747	0,870	0,930	0,975



Del análisis de los datos anteriores podemos obtener las siguientes conclusiones:

- Si sólo tenemos dos módulos, el fallo del segundo no es importante a menos que reemplace al primer módulo.
- A medida que se va depurando el sistema y que se va incrementando el factor de la cobertura de fallos, es interesante utilizar un mayor número de repuestos que intentar aumentar la fiabilidad de los mismos.
- A medida que va aumentando la fiabilidad de los componentes, decrecen: la importancia que tiene la fiabilidad del componente en la fiabilidad global del sistema, la importancia del número de repuestos, y la influencia de la cobertura de fallos de los mismos.
- Cuando solamente tenemos un repuesto, importa poco la cobertura de fallos del sistema, puesto que no se incrementa de forma notable la fiabilidad del mismo. Si calculamos el máximo factor de incremento de la fiabilidad con respecto a la cobertura (RsIF), definido como el factor que se obtiene de dividir la fiabilidad del sistema para $c \geq 0'75$ con respecto a la fiabilidad del sistema para $c = 0'75$, vemos que es casi constante. En la gráfica se compara RsIF de un sistema con un sólo repuesto a un sistema con 5 repuestos.



Podemos calcular el *MTTF* de un sistema con repuestos integrando la ecuación [2]:

$$MTFF(\text{n modulos}) = \int_0^{\infty} R_m \sum_{i=0}^{n-1} c^i (1 - R_m)^i \, dt$$

que para una función exponencial de la fiabilidad se puede escribir de nuevo como:

$$MTFF(\text{n modulos}) = MTFF(\text{n - 1 modulos}) + \int_0^{\infty} R_m(t) c^{n-1} (1 - R_m(t))^{n-1} \, dt$$

$$= MTFF(\text{n - 1 modulos}) + \int_0^{\infty} e^{-\lambda} c^{n-1} (1 - e^{-\lambda})^{n-1} \, dt$$

$$= MTFF(\text{n - 1 modulos}) + \frac{c^{n-1}}{n\lambda} = \frac{1}{\lambda c} \sum_{i=1}^{n} \frac{c^i}{i}$$

Como se puede apreciar en la fórmula anterior, la contribución del repuesto número *n* al *MTTF* es c^{n-1}/n veces la de un único módulo. Por tanto, si *c* no está muy próximo a 1, la contribución es inapreciable.

Cuando estamos analizando los sistemas redundantes, es importante hacer notar que hay que elegir con cuidado el parámetro sobre el cual vamos a calcular la fiabilidad del sistema, ya que a veces puede llevarnos a sacar conclusiones erróneas. Por ejemplo si comparamos un TMR con un sistema simple y tenemos en cuenta el *MTTF* vemos que este parámetro es menor para el sistema replicado, por lo tanto el TMR es aparentemente peor que el sistema simple.

$$R_{simple} = e^{-\lambda t} \qquad MTTF_{simple} = \frac{1}{\lambda}$$

$$R_{TMR} = 3e^{-2\lambda} - 2e^{-3\lambda} \qquad MTTF_{TMR} = \frac{3}{2\lambda} - \frac{2}{3\lambda} = \frac{5}{6\lambda}$$

El *MTTF* representa el área que tiene la curva de la fiabilidad dentro de la gráfica, que como se puede apreciar en la Figura 5.16 es mayor para el sistema simple que para el TMR.

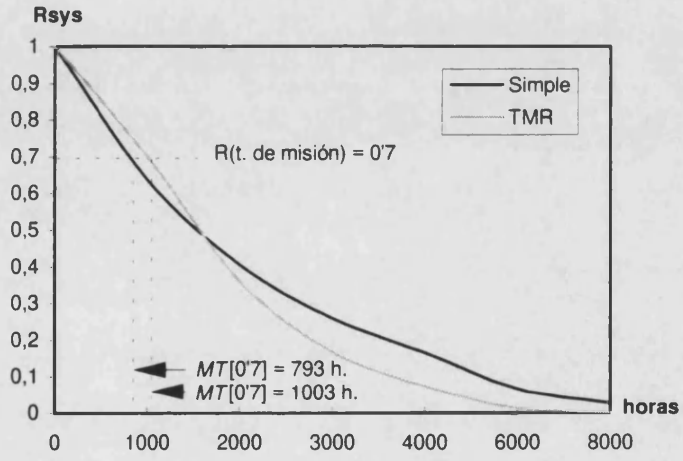


Figura 5.16: Comparación de la fiabilidad entre dos sistemas

Para comparar sistemas es mejor tener en cuenta únicamente una porción de su periodo de operación. Es mejor utilizar el incremento en el tiempo de misión entre dos sistemas para una fiabilidad dada de antemano, que se define como la proporción entre los tiempos de misión respectivos para los dos sistemas. En la Figura 5.16 se aprecia que para una fiabilidad mayor del 0'7, el sistema TMR puede operar 1003/793 veces más tiempo que el sistema simple.

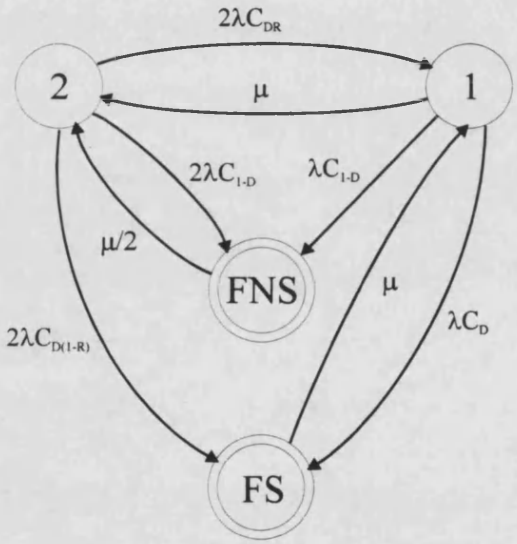
8 Modelado mediante Cadenas de Markov del Sistema FASST

La garantía de funcionamiento del sistema FASST se obtiene mediante el uso de una memoria estable combinado con la existencia de módulos de proceso *fail-stop*. Es muy importante, por tanto, a la hora de validar la garantía de funcionamiento de un sistema de este tipo, realizar un modelo que considere

la existencia de averías debidas a que se han sobrepasado los límites temporales de parada después de la avería que tienen los módulos. En el momento que ocurre una situación de este tipo, pese a que el sistema puede seguir funcionando, se dice que se ha contaminado con datos no válidos, que quedan de forma latente en la memoria y pueden producir una avería posterior. A partir de ahora consideraremos a estas averías como violaciones de las características de parada-ante-fallo del módulo.

Los modelos anteriores no consideran esta posibilidad, por lo que se propone un nuevo modelo macroscópico para la garantía de funcionamiento de los sistemas multiprocesadores con degradación basados en la existencia de módulos *fail-stop*. Este modelo se aplica directamente al Sistema FASST y se utiliza para calcular tanto la fiabilidad, como la seguridad, como la disponibilidad del sistema.

El modelo de la Figura 5.17 simboliza un sistema con degradación con dos procesadores. Como no se ha tenido en cuenta la influencia de un segundo fallo coincidente en el tiempo con el primero, el orden de magnitud de los tiempos de latencia de las tasas de recuperación de errores transitorios y de reconfiguración del sistema es de millones de veces menor que la tasa de fallos del sistema. Es por ello por lo que no se van a tener en cuenta estas tasas en el modelo macroscópico, sino que lo que realmente va a servir es el valor de las coberturas de detección y de reconfiguración de los errores.



	Significado	Estados	Valor
C_D	Cob. de detección de fallos	3, 4, 5, 6	69.7 %
C_{1-D}	Cob. de no detección de fall.	2, 7, 10	8.3 %
C_R	Cob. de recuperación	2, 4, 5, 6, 7	72.7 %
C_{DR}	Cob. Recuperación segura	4, 5, 6	67.8 %
$C_{D(1-R)}$	Cob. no recuperación segura	3	1.9 %
λ	Tasa de fallos del sistema		
μ	Tasa de reparaciones		

Figura 5.17: Modelo macroscópico del sistema FASST

Se han definido cuatro estados. En el estado 2 el sistema está funcionando correctamente con los dos módulos de proceso en estado operativo. Cuando se produce un fallo, el sistema se puede degradar pasando al estado 1 en donde sólo hay un módulo funcionando y no se ha producido ninguna violación de las características de *fail-stop* del módulo que ha fallado. Se ha considerado que aunque el sistema se degrade y siga funcionando, si no se ha detectado el error en el módulo con fallo el sistema evoluciona hacia el estado FNS (fallo no seguro), puesto que no se pueden prever las consecuencias de este tipo de error. Si por el contrario cuando se produce el error fallan los mecanismos de reconfiguración del sistema, el modelo evoluciona desde el estado 2 hacia el estado FS (fallo seguro) si el error había sido detectado o hacia el estado FNS si el error no se detectó, puesto que el sistema se para mucho después de que se produzca el error. Si se produce un error cuando el sistema está en el estado 1, es decir cuando sólo hay un módulo funcionando, se pasa al estado de FS si se ha detectado el error o al estado FNS si no se ha detectado.

Para el cálculo de la fiabilidad se pueden considerar dos alternativas. En la primera se supone que el sistema no considera la posibilidad de extracción y de inserción en vivo del módulo que ha fallado y por lo tanto en el momento que fallan los dos módulos, el sistema falla. El modelo equivalente se obtiene igualando a cero todas las tasas de reparación existentes.

La segunda alternativa sí que contempla el caso de la extracción y de la posterior inserción en vivo del módulo que ha fallado una vez se haya reparado. Si el sistema está funcionando, se puede extraer el módulo y repararlo, por tanto existe una transición con tasa de reparaciones μ que reconfigura el sistema para que vuelva a funcionar con los dos módulos. Este es el caso del Sistema FASST, puesto que el bus Futurebus+ goza de esta característica.

El modelo de seguridad y de disponibilidad se diferencia del anterior en que ya no tenemos estados absorbentes en la cadena de Markov. El análisis pasa por calcular la suma de las probabilidades de estar en estados con al menos algún procesador funcionando en el estado estable, para el caso de la disponibilidad, y por el cálculo de la probabilidad en el estado estable de encontrarse en alguno de los estados seguros para calcular la seguridad de funcionamiento. En ambos casos ya no tenemos una función dependiente del tiempo, sino lo que nos interesa es realizar el cálculo cuando el tiempo tiende a infinito. Los estados absorbentes se han eliminado puesto que se han tenido en cuenta las tasas de puesta en marcha del sistema cuando éste sufre una avería. En el caso de averiarse en un caso seguro, bastará con arreglar un módulo para poner de nuevo el sistema en funcionamiento. Si por el contrario el sistema se avería en un caso no seguro, el problema es más grave y se deben reparar los dos módulos para dejar al sistema funcionando otra vez. Se ha considerado que se tarda justo el doble de tiempo en arreglar los dos módulos con fallo no seguro que el tiempo que se tarda en arreglar un solo módulo que deja de funcionar en el estado seguro.

Para simplificar la sintaxis del modelo se han definido nuevas coberturas de detección y de recuperación de fallos y sus valores se van a estimar mediante un método experimental basado en la inyección física de fallos.

Las tasas de fallos de los módulos de proceso del sistema se han calculado en una sección anterior basándose en el estándar MIL-HDBL-217F. Faltará calcular la tasa de reparación de los módulos cuyo valor se elige de forma aproximada, según la experiencia que se tenga sobre el funcionamiento del prototipo.

9 Conclusiones

En este capítulo se ha realizado un análisis de las técnicas que permiten validar la garantía de funcionamiento de un sistema tolerante a fallos. El objetivo ha sido el de revisar los modelos de cobertura de fallos existentes para poder terminar con un modelo macroscópico de cobertura del sistema FASST.

Estos modelos se caracterizan por un conjunto de tasas de transición entre estados, que dependen de la cobertura de detección de errores y de recuperación y también de la tasa de producción de errores. Se ha resuelto un modelo de sistema ejemplo para observar la importancia que tiene el factor de cobertura en los parámetros característicos del sistema. También se ha calculado la tasa de producción de fallos del sistema FASST aplicando el estándar americano MIL-HDBL-217F.

Capítulo 6

Análisis experimental de la garantía de funcionamiento

1 Introducción

Una vez realizado el diseño de un sistema tolerante a fallos, se debe realizar una validación del mismo con respecto a las especificaciones de la garantía de funcionamiento, mediante el estudio del efecto que tienen los fallos y las averías en el sistema. Debido a la naturaleza destructiva de las averías y a las grandes latencias de producción de errores, se hace difícil identificar cuáles son las causas de las averías cuando el sistema está funcionando a pleno rendimiento.

Para poder estudiar por tanto la garantía de funcionamiento de un sistema, es preferible implementar un experimento que me de información sobre las averías potenciales que puede sufrir el sistema, antes que tener que esperar a que estas se produzcan de forma natural. Antes que nada se debe estudiar el comportamiento del sistema, y sus mecanismos de detección de errores y de recuperación y se deben de implementar instrumentos que me permitan inyectar fallos de forma artificial, crear errores o averías y monitorizar sus efectos. En [67] y en [68] se puede ver un resumen sobre las técnicas existentes para la inyección de fallos.

La validación experimental como se realiza inyectando fallos de forma artificial, tiene el problema de que no suministra información real sobre algunos parámetros muy influyentes en la garantía de funcionamiento del sistema, como pueden ser el tiempo que transcurre entre dos fallos consecutivos, y el tiempo medio de reparación del sistema.

La inyección de fallos se relaciona estrechamente, dentro del proceso de validación, con la eliminación de fallos basada en la verificación y con la predicción de fallos basada en la evaluación. Con respecto a la eliminación de fallos, se realiza un análisis cualitativo, tratando de ver la idoneidad de los procedimientos de verificación y de los mecanismos de tolerancia a fallos con respecto a los fallos que se desean considerar. En el caso de la predicción de fallos, la inyección de fallos permite caracterizar

la cobertura de los fallos y estimar los valores de los parámetros característicos de los mecanismos de tolerancia a fallos. Los resultados negativos obtenidos de la evaluación servirán para retroceder en el diseño y mejorar los mecanismos de tolerancia a fallos.

En [81] se describe una metodología que formaliza el proceso de inyección de fallos. Cualquier experimento de inyección viene caracterizado por una tupla formada por cuatro conjuntos, llamados respectivamente FARM. El dominio de entrada se corresponde con un conjunto de fallos F y un conjunto de activaciones A, utilizados para probar el sistema y el dominio de salida se corresponde con un conjunto de lecturas R y un conjunto de medidas M derivadas de las lecturas. Esta metodología se aplica en los niveles de inyección vistos anteriormente: el nivel axiomático o analítico (diagramas de bloques de fiabilidad, árboles de fallos, Cadenas de Markov y Redes de Petri), el nivel empírico, que incorpora descripciones más detalladas sobre el comportamiento y la estructura del sistema, necesitando por tanto la realización de una simulación para su cálculo y los modelos físicos que se corresponden con los prototipos tanto hardware como software. A continuación se puede ver una descripción más detallada de los conjuntos FARM:

- **El conjunto F:** En los modelos axiomáticos, el conjunto F se describe mediante procesos estocásticos, cuyos parámetros son distribuciones probabilísticas. En los modelos empíricos se utilizan distribuciones más reales en los parámetros del conjunto, realizando modelos de simulación de fallos jerárquicos que permiten evaluar el sistema a cualquier nivel. Los modelos físicos son necesarios para evaluar la implementación real del sistema. Los estudios en los prototipos software se limitan a alterar los programas mediante variaciones de líneas de código en el conjunto F. En los estudios hardware se aplican métodos de inyección de fallos para construir el conjunto F.

  Si el objetivo de la evaluación es la eliminación de fallos, el conjunto F estará formado por un pequeño número de fallos específicos que se identifican de acuerdo con las especificaciones de diseño. Los experimentos con estos fallos, además deben ser reproducibles para poder comprobar la funcionalidad de los posibles cambios en el diseño. Con respecto a la predicción de fallos, el conjunto F se corresponde con una distribución estadística de entre todos los posibles fallos.

- **El conjunto A:** En los modelos axiomáticos, el conjunto A es descrito mediante procesos estocásticos, al igual que se hacía con el conjunto F. Sin embargo, debido al mayor orden de magnitud de los parámetros de este conjunto, los procesos se estabilizan más rápido e incluso a veces se llega a omitir la descripción del conjunto A. En los modelos empíricos el conjunto A describe el comportamiento del sistema ya que los parámetros elementales se pueden definir de forma más apropiada. En los prototipos software y hardware el conjunto A consiste en un conjunto de patrones de test obtenidos mediante inyección de fallos.

  En la eliminación de fallos se deben distinguir dos casos dependiendo de los objetivos del test. Si el objetivo es la activación del fallo para observar el error, el conjunto A está formado implícitamente por los procedimientos de test, mientras que si el objetivo es el procesado del error, el conjunto A será un programa de test que maximiza el número de fallos producidos. Para la predicción de fallos el conjunto se corresponde con una simulación del sistema bajo test.

- **Los conjuntos R y M:** En los modelos axiomáticos el conjunto M se corresponde con las medidas de fiabilidad que se pueden calcular mediante un procesado analítico del modelo. En los modelos empíricos y físicos las medidas de M se obtienen aplicando varias series de inyección de fallos, llamadas experimentos.

En el caso de la eliminación de fallos, el conjunto M es una variable binaria que indica el éxito en la predicción del comportamiento del sistema. Si las predicciones son erróneas es aconsejable recoger cierto número de datos de la salida *r* con propósitos de diagnóstico. En la predicción de fallos el conjunto M se corresponde con medidas probabilísticas de la ocurrencia de estados característicos y de la duración y separación entre estados.

Para cada experimento se selecciona un fallo *f* de F y se describe una trayectoria de activación *a* de A. Las reacciones del sistema se describen en *r* caracterizando al experimento. Por lo tanto se puede definir un experimento como una tripleta <*f*,*a*,*r*> donde los resultados de cada experimento *r* forman un conjunto de resultados R para la secuencia de tests específica y se pueden usar para elaborar una medida en M.

Al pin donde se van a inyectar los fallos se llama *punto de inyección*. Para cada experimento, el fallo *f* inyectado se corresponde con un conjunto de fallos elementales $f_1, f_2, \ldots, f_m$, donde *m* indica la multiplicidad del fallo. Cada fallo elemental f_i en un punto de inyección se caracteriza por los siguientes atributos:

- La localización del fallo a través del elemento que ha fallado.
- La naturaleza (modelo) del fallo, es decir, si es un cortocircuito, circuito abierto, pegado a 1 o pegado a 0 y si es permanente o transitorio.
- Tiempo en el que se aplica el fallo con respecto al comienzo del experimento.
- La duración de la aplicación del fallo.

Para cada experimento, la salida *r* corresponde con un conjunto de salidas elementales $r_1, r_2, \ldots, r_n$, que corresponden con las reacciones del sistema a los fallos inyectados. Cada salida r_i se puede descomponer en tres tipos:

- Binarias: Predicado que indica si se produce o no una respuesta a un determinado evento.
- Contador: Indican medidas temporales del experimento.
- Adquisición de datos extensiva de la evolución del comportamiento del sistema.

2 Manifestación de los fallos

Para cubrir el hueco entre el comportamiento del sistema y los modelos a nivel lógico de fallos, se deben realizar experimentos. Hay que inyectar fallos y ver si se detectan y cuál es su latencia, dependiendo de la duración, donde se inyectan y del tipo de ciclo del procesador. También es importante analizar cuál es el mecanismo de detección que los detecta.

La inyección de fallos se puede utilizar para validar los mecanismos de manejo de fallos y para producir un modelo de manifestación de fallos a nivel de sistema. Se pueden detectar dos tipos de cobertura de detección de errores:

- Cobertura de detección de fallos latentes/activos (fallos detectados / fallos latentes (no detectados))
- Cobertura de detección de errores (errores detectados por la aplicación y el S.O. / errores detectados).

Los tipos de fallos insertados suelen ser de pegado-a, compensado de dos bits (para no ser detectados por la paridad), byte a cero/uno (para emular un fallo de una selección de chip, condición de carrera o un fallo de un transceiver).

Las manifestaciones a nivel de sistema se refieren a cómo evoluciona un sistema ante la presencia de un fallo. Entre los eventos más usuales se suelen contemplar los siguientes:

- Parada de tareas: El S.O. detecta una instrucción ilegal, direcciones o datos incorrectos u otra condición que produce una excepción
- Salida inválida: Detectada por los comparadores
- Respuesta demasiado tarde: Salta un temporizador
- Fallo del sistema: Cuando el fallo inyectado produce una reinicialización del sistema
- Parada del sistema sin reinicialización

Una vez determinadas las consecuencias del error, se debe también analizar la latencia del error, que se puede hacer en función de la manifestación del sistema que primero detectó el error.

3 Técnicas estadísticas

Existen infinidad de métodos y técnicas para realizar un análisis sobre los datos recogidos acerca de la garantía de funcionamiento de un sistema tolerante a fallos. El uso de sencillas técnicas estadísticas, que permitan calcular parámetros simples, tales como el porcentaje, la frecuencia o la probabilidad de producción y de detección de fallos suministran una idea general del comportamiento del sistema y a menudo, simplemente con estos datos se pueden conocer cuales son los cuellos de botella en la fiabilidad del equipo.

Sin embargo, si se quiere evaluar y modelar la función de fiabilidad del sistema es necesario encontrar una función de distribución que la represente. Por simplicidad se asume que esta función es de tipo exponencial, sin embargo existen estudios basados en medidas sobre sistema reales [121][122] que encontraron que la distribución de Weibull con una tasa de fallos decreciente es representativa del tiempo entre fallos (MTBF) en estos sistemas.

En el área del análisis experimental de la garantía de funcionamiento de un sistema informático, se está prestando mayor atención a la combinación entre el modelado analítico y el análisis experimental y también en combinar la evaluación con el diseño del sistema. Las técnicas de modelado analítico se están aplicando a sistemas reales para evaluar sus características de rendimiento y de fiabilidad. Los resultados del análisis experimental se usan para validar los modelos analíticos y revelar prácticas que el modelado analítico debe tener en cuenta para desarrollar modelos más representativos.

Una vez recogidos los datos del experimento, debo de aplicar una serie de técnicas estadísticas para analizarlos y poder sacar las conclusiones adecuadas. El método más simple es estudiarlos a través de medias y probabilidades, sin embargo para poder predecir el comportamiento de sistemas completos, estas observaciones se deben determinar de una forma más precisa a través de una función del tiempo. Para ello se debe encontrar una distribución matemática que se acople y represente a los datos observados. En [96] se realiza un análisis de la validez de estos métodos para los sistemas con necesidades altas de cobertura de fallos.

3.1 Ajuste del muestreo a una distribución conocida

Cuando se caracterizan los fallos de un sistema o componente mediante una distribución en particular, se deben determinar (estimar), a través de los datos del experimento, los valores de los parámetros de la distribución.

Existen dos tipos de estimación: por punto, en donde se obtiene un valor estimado concreto y por intervalo, en donde el parámetro a estimar pertenece a un cierto intervalo con una cierta probabilidad. Estos métodos se emplean para estimar la cobertura de detección de errores o el tiempo medio entre fallos.

Estimadores máximo-verosímiles

Este tipo de estimación proviene de [69]. Si $\vec{x}_n$ es una muestra de los datos observados y $\vec{\theta}$ un vector de parámetros desconocidos, si llamamos $P(\vec{x}_n|\vec{\theta})$ a la probabilidad de observar $\vec{x}_n$ dados los parámetros del vector $\vec{\theta}$, el estimador máximo-verosimil de $\vec{\theta}$, denotado por $\vec{\theta}_{MV}$, es el valor de $\vec{\theta}$ para el cual $P(\vec{x}_n|\vec{\theta})$ es máximo. Es decir:

$$\forall\vec{\theta} \quad P(\vec{x}_n|\vec{\theta}_{ML}) \geq P(\vec{x}_n|\vec{\theta})$$

Si se supone que el tiempo al fallo sigue una distribución exponencial y se obtiene experimentalmente una medida de los tiempos de fallo de un componente, $\vec{\tau} = (\tau_1, \tau_2, \ldots, \tau_N)$. Para hallar el estimador máximo-verosimil de λ en la distribución exponencial tenemos:

$$P(\vec{\tau}|\lambda) = (\lambda e^{-\lambda\tau_1})(\lambda e^{-\lambda\tau_2})\cdots(\lambda e^{-\lambda\tau_N}) = e^{-\lambda\sum_{i=1}^{N}\tau_i + N\ln\lambda}$$

Esta función debe ser un máximo cuando λ valga λ_{ML}. Por lo tanto para maximizar la función anterior se puede hallar el mínimo de la función:

$$f(\lambda) = \lambda\sum_{i=1}^{N}\tau_i - N\ln\lambda$$

que diferenciando con respecto a λ y haciendo la derivada igual a 0 se obtiene el siguiente valor para λ:

$$\lambda_{ML} = \frac{N}{\sum_{i=1}^{N}\tau_i}$$

que es la inversa de la media de todos los tiempos hasta la avería obtenidos experimentalmente.

Análisis de regresión lineal

Si se considera que los datos de la muestra siguen una distribución de Weibull, se puede realizar un análisis gráfico de la función de distribución para ajustar los datos. Esta técnica se basa en la transformación de la función de distribución de Weibull en una función lineal de $\ln(t)$:

$$\ln\left[\ln\frac{1}{1-F(t)}\right] = \alpha \ln t + \alpha \ln \lambda$$

Si los datos de la muestra siguen esta distribución, el gráfico que formarán se aproximará a una línea recta, que se puede calcular aplicando el método de los mínimos cuadrados. La pendiente de la curva es un estimador de α y el valor por el que cruza la recta al eje de abcisas dividido por la pendiente es un estimador de $\ln(\lambda)$. El valor de la función $F(t)$ se estima como:

$$F(t_j) = \frac{j - 0'5}{N}$$

Intervalos de confianza

La estimación de los valores hecha mediante los métodos anteriores raramente se aproxima a los valores reales. Debido a ello es preferible estimar un rango de valores dentro de los cuales se encuentra el valor real del parámetro bajo consideración con una probabilidad bastante alta [71]. Si un rango de valores constituye un intervalo de confianza del 90 % para un parámetro, significa que si repito el muestreo, el 90 % de los intervalos de confianza así construidos deben contener los valores de los parámetros reales.

3.2 Caracterización de la distribución

A través de la estimación anterior puedo averiguar la media y la varianza de un determinado parámetro. Sin embargo, si conozco su distribución puedo disponer de más información para realizar modelos o sacar conclusiones. En [70] se puede ver un análisis para determinar la cobertura de fallos de un sistema a través del calculo de los momentos de la distribución experimental.

Distribución empírica

La manera más fácil de obtener una distribución experimental de una muestra X es dibujar un histograma de las observaciones. El rango del espacio muestral lo puedo dividir en k zonas del mismo tamaño, separadas mediante $x_0, x_1, \ldots, x_k$, para una muestra de tamaño n. Si en cada zona tengo y_i instancias, el tamaño de la muestra se calcula como:

$$n = \sum_{i=1}^{k} y_i$$

siendo y_i / n una estimación de la probabilidad de que X tome una valor en la zona i. El histograma es la función de distribución de probabilidad empírica de X. La función de distribución se calcula como:

$$F_k(x) = \begin{cases} 0, & x < x_0 \\ \sum_{l=1}^{i} \frac{y_l}{n}, & x_{i-1} \le x \\ 1, & x_k \le x \end{cases}$$

El problema reside en la elección del tamaño de la zona, puesto que si este es demasiado pequeño pueden haber tantas variaciones que puede ser imposible identificar la distribución, y si por el

contrario es demasiado grande, se pueden perder muchos detalles de la distribución. Lo normal es utilizar sobre las decenas de zonas, dependiendo del tamaño de la muestra.

4 Inyección de fallos en la fase de diseño

Para realizar una evaluación de la garantía de funcionamiento de un sistema informático cuando únicamente está hecho el diseño, debido a la ausencia del prototipo, es necesario realizar una inyección de fallos simulada sobre un modelo simplificado del propio prototipo. Al obtener los resultados se podrá tener una estimación de la efectividad de los mecanismos de tolerancia a fallos que se han incorporado en el sistema, permitiendo en el caso de que no sean satisfactorios retroceder en el diseño para mejorar los mecanismos de tolerancia a fallos.

Los objetivos que se persiguen en la simulación son el determinar los cuellos de botella en la garantía de funcionamiento del sistema, analizar como se propagan los fallos dentro del sistema, determinar la cobertura de los mecanismos de detección y de reconfiguración de errores, medir la latencia en la detección y recuperación de errores, determinar la efectividad de los esquemas de reconfiguración y analizar la pérdida de rendimiento que sufre el sistema como consecuencia del proceso de producción de fallos.

La ventaja de la simulación frente a la realización de modelos analíticos estriba en que permite la realización de modelos del sistema mucho más complejos y por lo tanto más representativos, aunque tiene el problema de que si el modelo es muy complejo se necesita mucho tiempo para terminar la simulación.

Figura 6.1: Niveles en la inyección de fallos

La simulación a nivel eléctrico emula fallos transitorios mediante la provocación de cambios de voltaje y de intensidad en los circuitos integrados, que producen errores en los valores lógicos a nivel de puerta. Estos errores se propagan a otras unidades funcionales dentro del circuito y por último a las patas de salida de los circuitos integrados. La simulación puede resultar costosa en el tiempo ya que hasta que se manifiesta el fallo en el dispositivo se tienen que atravesar muchos niveles. Sin embargo hay veces que es necesaria hacerla, debido principalmente a tres causas:

- Se utiliza para estudiar el impacto que producen las causas físicas en el proceso de la producción del fallo y de la avería.
- Hay veces que los modelos de pegado-a no son representativos de los fallos físicos.
- Permite analizar circuitos con partes analógicas.

Para realizar la simulación se suele utilizar un modelo mixto [72] en donde las partes sobre las cuales actúa el proceso de inyección del fallo se simulan a nivel eléctrico y el resto se simula a nivel lógico. Recientemente se ha desarrollado una técnica más eficiente [73], en donde la representación del circuito al cual se le están inyectando fallos se evalúa tanto a nivel lógico como eléctrico. Cuando se inyecta el fallo se emplea el nivel eléctrico, para interpretar de forma analógica el comportamiento ante el fallo. Cuando se estabiliza el voltaje pasando a ser una señal discreta se emplea el modelo lógico hasta que vuelve a ocurrir un nuevo transitorio en donde se vuelve a cambiar el modelo.

La inyección de fallos simulada a nivel lógico persigue los mismos objetivos que la anterior, pero difiere en que ahora se utilizan modelos de fallos más sencillos que son abstracciones del comportamiento de los modelos físicos a nivel de puerta lógica. De este modo se puede realizar un estudio más sencillo de los sistemas VLSI. Los modelos de fallos utilizados son los de *pegado-a* y los de inversión de línea.

En la simulación lógica, se introduce un fallo en el modelo de un dispositivo en concreto, se ejercita el circuito mediante un conjunto de valores para las entradas (llamado vector de test) y se recoge el resultado obtenido. A continuación se compara con el resultado obtenido al ejercitar un *dispositivo de oro*, similar al anterior pero libre de errores con el mismo vector. Si el resultado obtenido es el mismo en ambos casos indica que el vector de test no ha detectado el fallo, en caso contrario el fallo sí que se ha detectado. El proceso se repite de forma análoga con todas las combinaciones de valores de entrada para todos los posibles fallos que se puedan simular en el circuito hasta que se detecta el fallo. Al termino de la simulación se sabrá la cobertura de detección de fallos que tienen un conjunto de patrones de test para un dispositivo dado. Esta técnica se utiliza para detectar los defectos de fabricación de un integrado. Los vectores se generan mediante un ordenador utilizando un generador de patrones de test automático (ATPG) [74].

Si nos preocupa realizar un análisis global de la garantía de funcionamiento del sistema, tendremos que preocuparnos por la sensibilidad y la propagación de los fallos en los circuitos VLSI. Además tendremos que evaluar simultáneamente el comportamiento ante fallo de todos los componentes en conjunto. Para simular los fallos de *pegado-a* se puede poner un nodo del circuito a un valor determinado durante todo el periodo que dure la simulación, mientras que para los fallos transitorios sólo se realizará durante un breve instante de tiempo. El fallo transitorio se puede simular bien generando un pulso, bien cambiando el contenido de un registro. El problema reside en que mediante esta técnica no se refleja fielmente el comportamiento del sistema ya que los transitorios se pueden propagar por múltiples caminos y resultar en más de un error en los registros.

Existe una técnica desarrollada en [75] que permite utilizar modelos de fallos más realistas, basada en la utilización de un diccionario de comportamientos del fallo. Dado un circuito determinado se realiza una simulación de fallos del mismo a nivel eléctrico y se analiza su comportamiento a nivel lógico. Este resultado se almacena en el diccionario, recogiendo el vector de entrada, el tiempo en el que se produce la inyección y la localización del fallo. Después se pueden inyectar fallos a nivel lógico en partes próximas al circuito y analizar las consecuencias mediante el diccionario. En [76] se utilizan diccionarios de fallos para inyectar errores en el software que se corresponden con un fallo hardware. El método se utiliza para analizar el impacto de los fallos transitorios en una red de comunicaciones.

Por último, la simulación de la inyección de fallos a nivel funcional se utiliza para estudiar las características de fiabilidad de sistemas grandes o incluso de redes de ordenadores, en vez de estudiar

los componentes que las componen por separado. Se estudia tanto el hardware como el software y la interacción que existe entre los mismos. Existen una serie de factores a tener en cuenta a la hora de realizar estos modelos:

- La carencia de modelos de fallos a nivel de sistema bien definidos: Los componentes son muy diversos, realizan múltiples funciones y están estrechamente relacionados unos con otros, por lo que se hace muy difícil establecer un único modelo de fallos que se pueda aplicar consistentemente a todos los componentes. Debido a ello se utilizan diferentes tipos de modelos de fallos, dependiendo del componente que se quiera estudiar, derivados de la simulación a nivel lógico. Por ejemplo modelos típicos son el cambio en un bit de memoria o la inserción de un error en un registro de la CPU.

- El gran esfuerzo y tiempo requerido para desarrollar un modelo de simulación funcional: Ello es debido principalmente al tiempo que se gasta en la descripción de las funcionalidades de los componentes del sistema y al tiempo que se tarda en preparar el experimento de la inyección, iniciar las reparaciones, sincronizar los eventos y recoger las estadísticas. Se debe de utilizar un método estructurado en donde se recojan las experiencias de simulaciones anteriores, pudiendo construir una librería con objetos software que describan los componentes.

- La influencia del software en la fiabilidad del sistema [119]: A medida que la tecnología avanza, se van reduciendo las tasas de fallos de los componentes, por lo que los esquemas de detección y reparación del sistema dependen en gran medida de la aplicación, que es la que marca los tiempos de latencia en la detección del error y los tiempos de propagación del mismo. Es por tanto muy importante incorporar en las herramientas de simulación al software para realizar los estudios de fiabilidad.

- La explosión del tiempo de simulación: Debido a la baja probabilidad de fallo del sistema se necesitan realizar simulaciones muy costosas temporalmente para obtener resultados válidos desde el punto de vista estadístico. La solución adoptada es la división del modelo en otros modelos más simples que se analizan por separado y la posterior recombinación de los mismos para obtener un modelo del sistema simplificado. Si todos los modelos son modelos de simulación, se dice que se ha realizado una *simulación jerárquica*, mientras que si por el contrario el modelo del sistema es un modelo analítico, se dice que se ha tomado una aproximación mediante una *simulación híbrida*.

Los simuladores funcionales tienen la ventaja frente a los modelos analíticos que permiten el modelado del comportamiento del sistema, es decir que no necesitan predefinir los efectos de los fallos mediante probabilidades. Para realizar un análisis únicamente es necesario especificar las tasas de producción de fallos (que pueden seguir cualquier distribución) y los tipos de fallos que se van a inyectar. Los demás elementos del sistema ya están simulados y como resultado se pueden identificar los mecanismos de detección de fallos, obtener la probabilidades de mal funcionamiento y cuantificar el efecto de los fallos, además de poder conocer con exactitud los parámetros adecuados a emplear en los modelos analíticos. Como el software que se utiliza es imagen del real, también se pueden detectar fallos de diseño en el mismo.

Actualmente existen muchas herramientas de simulación funcional que se pueden utilizar para inyectar fallos. Entre ellas se pueden citar: NEST [77], DEPEND [78], REACT [79] y MEFISTO [80].

5 Inyección de fallos en la fase de prototipación

En la fase de prototipación, el sistema se está ejercitando mediante una carga controlada. Se puede realizar una inyección de fallos real y emplear utilidades de monitorización para evaluar el comportamiento del sistema ante la ocurrencia de fallos, incluyendo la cobertura de detección y la capacidad de recuperación de los mecanismos de tolerancia a fallos. La inyección de fallos en el sistema real suministra información sobre el proceso de generación de la avería, pasando por todos los estados desde la generación del fallo, hasta la recuperación del sistema. Este método proporciona resultados más exactos que la simulación, puesto que ya no se está realizando ninguna suposición sobre como se va a implementar el diseño. Además, también se pueden inyectar fallos sobre un equipo real, obteniéndose información mas fideligna sobre el proceso de producción de averías.

Un entorno de inyección típico, como muestra la Figura 6.2, necesario para realizar los experimentos consta de los siguientes componentes:

- El sistema al que se le van a inyectar los fallos, que puede ser un ordenador, un circuito VLSI o incluso una red, y que además está ejecutando una carga determinada.
- Un controlador que gobierna el proceso de la inyección. Suele ser un programa que a veces reside en otra máquina aparte.
- El propio inyector de fallos que se comunica con el controlador para obtener información sobre el tipo de fallos a inyectar, el lugar donde se va a producir y en qué instante de tiempo.
- Un monitor que se encarga de detectar las ejecuciones normales o anormales de la carga del sistema y que inicia la toma de datos cuando lo considera necesario.
- El sistema recolector y analizador de datos que recoge los datos en paralelo con el desarrollo del experimento y que luego los analiza utilizando las herramientas necesarias.

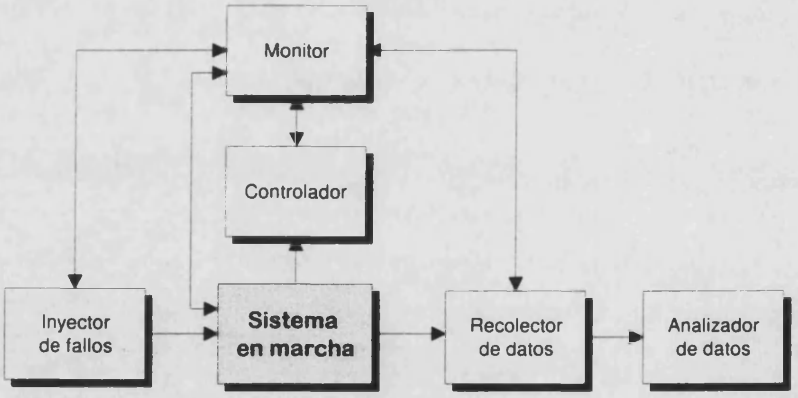


Figura 6.2: Componentes de un experimento de inyección

Los fallos pueden inyectarse al nivel hardware (fallos lógicos o eléctricos) o al nivel software (corrupción del código o de los datos) ó se puede emplear un método que consiste en la radiación de iones.

5.1 Inyección de fallos hardware

Este método utiliza un dispositivo adicional, llamado inyector de fallos, para inducir fallos de forma artificial en el hardware del sistema. El método es apropiado cuando se necesita realizar un análisis del comportamiento ante el fallo con mucha resolución temporal.

Se pueden emplear dos técnicas, la primera utiliza pruebas que se conectan a los puntos de inyección, que alteran la intensidad de corriente que pasa por las líneas, modificando el valor lógico de las señales. Los fallos que se inducen suelen ser de *pegado-a*, aunque también se pueden simular fallos por puenteado al utilizar varias puntas de inyección. La segunda técnica necesita la incorporación de un hardware adicional al sistema, que habitualmente es un zócalo conectado entre el circuito impreso y el integrado donde se quieren inyectar los fallos. El zócalo tiene la misión de aislar ciertas señales de salida del circuito al que sustituye y en su lugar insertar los valores lógicos deseados. Este método, debido a que se incluye un elemento inteligente, permite la inyección de fallos más complejos, como los derivados de funciones lógicas, fallos por puenteado o incluso retardos en las señales.

Los puntos de inyección se limitan a aquellas posiciones en el sistema que son accesibles físicamente con las puntas de prueba. Normalmente se inyecta en las patas de los circuitos integrados, aunque ahora, con la inclusión de tecnología de montaje superficial, a veces parece ser más recomendable utilizar las vías del circuito impreso como puntos de acceso, debido a la imposibilidad de hacerlo en los pines. También es usual utilizar buses para insertar las sondas de inyección.

Los ventaja que tienen estos sistemas, aparte de su alta resolución, es que permiten fácilmente controlar la duración de los fallos que insertan. Los inyectores hardware permiten inyectar fallos permanentes, transitorios o intermitentes, dependiendo de la duración de la inyección y de su periodicidad.

Múltiples experimentos de inyección se ha realizado sobre prototipos de sistemas tolerantes a fallos. En [95] se realiza un análisis de los tiempos de latencia y de la cobertura de detección de fallos en un sistema basado en el procesador Motorola 68000. Aunque las latencias son similares tanto para los fallos transitorios como para los permanentes, la detección de fallos sí que es mucho mayor para los fallos permanentes. Dependiendo del punto de inserción del fallos también varía considerablemente la probabilidad de detección del error. En [93] se describe el proceso de inyección de fallos sobre un sistema basado también en el procesador Motorola 68000. Los resultados muestran que los mecanismos de detección de fallos implementados mediante técnicas software detectan mayor número de fallos que los mecanismos hardware. En [94] se realiza un análisis del comportamiento ante fallo de los sistemas con redundancia dinámica.

Como en secciones anteriores, proponemos para la lectura las referencias a dos entornos de inyección de fallos física: FTMP [83] y MESSALINE [81].

5.2 Inyección de fallos software

Este método tiene la ventaja de que no necesita de hardware adicional para realizar la inyección por lo tanto es una metodología muy barata y fácil de controlar. Mediante la inyección software se pueden inyectar fallos tanto en los componentes hardware accesibles mediante la ejecución de instrucciones máquina como en el software, con la ventaja adicional de que es el único método que permite la emulación de defectos en el software mediante la alteración de una parte del código.

La inyección se logra alterando el contenido de algún registro o de la memoria, basándose en modelos de fallos específicos, que simulan la ocurrencia de algún tipo de fallo hardware o software. Los fallos en el hardware pueden producir errores que afectan la ejecución correcta del software (errores software inducidos por el hardware) produciendo la ejecución de instrucciones incorrectas, el acceso a datos incorrectos o la salida de resultados incorrectos. Los fallos software son defectos en el diseño o la implementación del código y pueden provocar la desviación del sistema a un estado inesperado.

Los modelos de fallos utilizados más comúnmente pueden ser la utilización de una variable sin inicializar, la definición o el uso incorrecto de un módulo del programa, o la eliminación y alteración

de sentencias de código. Este hecho produce la modificación de los datos produciendo los errores software. Estos errores también pueden se inyectados directamente cambiando el segmento de datos. En la Tabla 6.1 se recogen las técnicas que más usualmente se emplean para realizar la inyección.

Tipo	Método
Fallo software	Modificación del segmento de texto del programa.
Error software	Modificación del segmento de datos del programa.
Fallo de la memoria	Intercambio de varios bits de memoria.
Fallo del procesador	Usar una interrupción para modificar la zona de memoria donde se almacenan los registros de la CPU.
Fallo del bus	Utilizar interrupciones antes y después de la instrucción para modificar el código o los datos usados por la instrucción y para restaurarlos después.
Fallo de la red	Modificar o eliminar algún paquete.

Tabla 6.1: Técnicas utilizadas para la inyección de fallos software

Normalmente se inyectan fallos hardware de naturaleza transitoria. Para emular fallos permanentes si se trata de un fallo de pegado-a en una posición de memoria, cada vez que haya una escritura sobre esa posición se modificará el bit en cuestión, o si por el contrario el fallo se produce en una línea de dirección, se deberán de modificar los registros de la CPU antes de realizar el acceso.

La ventaja de este método frente a la inyección hardware es que es fácil dirigir la inyección sobre una parte de la carga del sistema determinada, por ejemplo para analizar los procesos de un usuario en concreto o las tareas del sistema operativo. Sin embargo, como inconvenientes tiene que no se pueden inyectar fallos en posiciones no accesibles por el software, que el mismo proceso de inyección altera la carga del sistema, y que el método es demasiado lento para poder analizar las posibles propagaciones de los errores. Como solución a este último punto se puede incorporar un monitor hardware para analizar los efectos de los fallos inyectados. En [97] se realiza una comparación de las técnicas de inyección de fallos física frente a la inyección de fallos software. En [82] se propone un nuevo método para inyectar fallos basado en un modelo de fallos que influye sobre el lenguaje de transferencia de registros del procesador.

Como en apartados anteriores se incluye una referencia a las herramientas de inyección software más importantes que existen para su análisis: FIAT [84], FERRARI [85], HYBRID [86], DEFINE [87] y SFI [88].

5.3 Inyección de fallos mediante radiaciones

Este método consiste en el bombardeo con iones pesados de *Californio*^{252} a los circuitos integrados. Como efecto se produce la generación de fallos transitorios en posiciones aleatorias del integrado, que pueden afectar a uno o varios bits [89][90]. Con los métodos anteriores no se podía lograr este efecto.

6 Inyección de fallos en la fase de operación

En la fase operacional cuando el sistema está funcionando normalmente y ejecutando una carga real, en vez de inyectar fallos se puede registrar como eventualmente se van produciendo los errores y las averías. Esta es la forma más natural de analizar el comportamiento del sistema, ya que recoge datos reales que me ayudan a comprender como evoluciona el sistema de forma real.

En esta fase es donde realmente se puede evaluar la garantía de funcionamiento del sistema, ya que se están realizando medidas sobre el sistema real, trabajando en condiciones reales con una carga real. El análisis de las medidas de ocurrencia de errores y averías suministra información sobre su

comportamiento real, identifica los cuellos de botella del sistema, cuantifica las medidas de la fiabilidad y verifica las suposiciones realizadas en los modelos analíticos.

La Figura 6.3 muestra los pasos realizados en el estudio de las medidas realizadas en un sistema. El primer paso consiste en la extracción de toda la información necesaria del sistema, la clasificación de los errores y las averías y el agrupamiento e identificación de los mismos para evitar procesar el mismo fallo varias veces [91]. En el paso dos se deben identificar los modelos apropiados y hacer una selección de los parámetros que se van a estimar de los datos recogidos. El siguiente paso resuelve estos modelos para obtener las medidas de la garantía de funcionamiento del sistema (fiabilidad, disponibilidad, tasas de recuperación) y el último paso interpreta los modelos y las medidas obtenidas de los datos, para sacar conclusiones sobre las características del sistema y su impacto en la garantía de funcionamiento del mismo. En esta fase se pueden identificar los cuellos de botella en la fiabilidad y estudiar la dependencia existente entre la producción de errores y la carga del sistema.

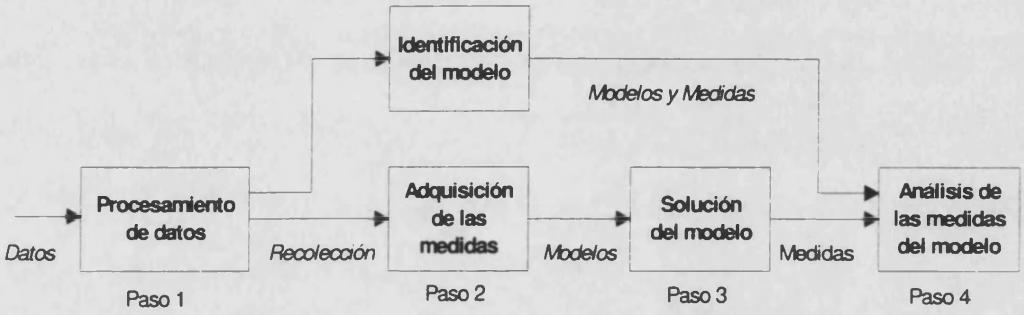


Figura 6.3: Proceso de análisis de un sistema

El problema de este último método es que las condiciones de operación del sistema pueden variar de forma drástica, invalidando la validez estadística de los resultados y además el análisis está limitado a errores detectados, no teniéndose en cuenta para nada los errores que no se detectan. Aparte de todo esto, los datos deben de recogerse durante mucho tiempo, ya que los errores y averías ocurren muy infrecuentemente.

7 Descripción del inyector de fallos utilizado en el experimento

La validación del sistema FASST se va a realizar por el método de la inyección física de fallos a nivel de pin [93]. Para ello se ha utilizado un dispositivo, llamado inyector de fallos, que permite alterar a voluntad el nivel lógico de una señal en cualquier parte del sistema. La condición indispensable es que el nodo donde se inyecta el fallo sea alcanzable, es decir que se pueda acceder para conectar la sonda de inyección.

El inyector utilizado [92] se ha diseñado en la Universidad Politécnica de Valencia en el Grupo de Sistemas Tolerantes a Fallos. El inyector se conecta por un lado a un ordenador personal, a través de una placa específica compuesta por varios puertos paralelos de Entrada/Salida para bus ISA, y por otro lado al prototipo del sistema tolerante a fallos a validar. (En la Figura 6.4 llamado CUT, ó circuito bajo pruebas).

Este inyector, como se puede ver en la Figura 6.4, consta de un conjunto de bloques funcionales que se describen a continuación:

Módulo de sincronización y disparo: Esta parte se encarga de disparar la inyección cuando se detecta un evento determinado. La palabra de disparo se programa de forma automática accediendo al registro valor de la máscara de disparo que está físicamente implementado en la tarjeta del PC. Para poder discriminar un conjunto de señales del disparo, existen una serie de jumpers, que de forma manual, permiten habilitar a la señal para que se tenga en cuenta en la palabra de disparo.

Aunque se supone que el proceso de generación de fallos en un circuito electrónico es aleatorio, es conveniente realizar la inyección siguiendo una estrategia, mediante la programación de las palabras de disparo, para poder tener más información sobre cuáles han sido los efectos de los fallos en función del ciclo de bus que se esté realizando en ese momento. Por ejemplo si se desea calcular la latencia de detección de fallos en el bus de datos, si el ciclo de bus es una escritura, el procesador no se dará cuenta de que se ha producido un fallo hasta que se vuelva a leer el dato incorrecto y por lo tanto la latencia será mucho mayor que en los ciclos de lectura. Dependiendo del tipo de análisis a realizar y de los mecanismos de detección de fallos implementados en el sistema, sería bueno hacer esta distinción en el cálculo de los parámetros de la garantía de funcionamiento. También sería interesante distinguir, o llevar cuenta de los fallos provocados en varias regiones del espacio de direccionamiento del procesador. No afectarán igual los fallos producidos en la memoria, por ejemplo, que en los registros de los controladores de Entrada/Salida.

Figura 6.4: Diagrama de bloques del inyector

Otro aspecto importante a tener en cuenta a la hora de elegir la palabra de disparo es el de maximizar el número de errores efectivos producidos por la inyección. El error es efectivo cuando la inyección produce un cambio en el estado de la señal. Por ejemplo no constituiría un error efectivo el hecho de inyectar un cero en una línea con un pegado a cero. Por tanto es deseable disparar la inyección siempre que la línea a inyectar tenga un valor contrario del que se quiera forzar. Si no se realiza de este modo, la probabilidad de consecución de errores efectivos es muy baja. En una línea de reloj con un ciclo del 50 % será aproximadamente del 50 %. En la Figura 6.5 se muestra con detalle el ejemplo anterior. La inyección A produce un error efectivo, puesto que al inyectar el cero se produce un flanco. Sin

embargo, la segunda inyección no deriva en un error efectivo puesto que al forzar el cero ya no se van a producir más flancos en la señal.

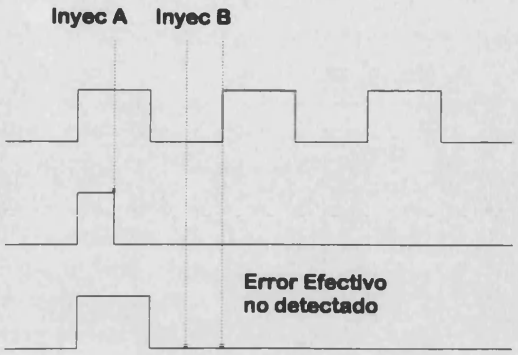


Figura 6.5: Influencia del disparo en el error efectivo

El módulo temporizador: Este módulo se encarga de generar la señal que indica la duración de la inyección. Está formado básicamente por un integrado con 3 contadores de 16 bits a 10 MHz, que se pueden programar para producir diversas formas de onda. La combinación del modo de funcionamiento de los contadores con la señal de habilitación y la señal de salida de los mismos, permite definir los tres tipos distintos de fallos que se pueden inyectar: fallos permanentes, transitorios y fallos intermitentes. En la Tabla 6.2 se recoge la duración y la frecuencia máxima de inyección de fallos.

Tipo de fallo	Duración	Periodo
Fallo Permanente	200 ns - 107'37 seg	0
Fallo Transitorio	100 ns - 3'28 mseg	0
Fallo intermitente	100 ns - 3'28 mseg	500 useg - 1'64 seg

Tabla 6.2: Valores extremos de duración y frecuencia del fallo

Los fallos permanentes simulan la producción de un fallo de tipo pegado-a en una línea. Los transitorios (que son los que ocurren la mayor parte de las veces) son fallos que se producen de forma esporádica con una duración limitada, normalmente de unos pocos ciclos de bus. Los intermitentes son fallos transitorios que se repiten de forma cíclica en el tiempo.

En el momento en que el módulo activa la señal de detección de error, indicando que el error ha sido detectado, los contadores desactivan su salida para terminar con la inyección del fallo. De esta forma no se desgasta de forma inútil al integrado.

Los relés sirven para controlar sobre qué línea se va a inyectar el fallo. La activación del relé se realiza por software escribiendo en uno de los puertos de E/S de la placa de PC. Existen 40 relés, por tanto se podrá inyectar en 40 sitios diferentes sin variar la topología de conexión de las sondas de inyección. Las salidas de todos los relés están conectadas a la misma línea de inyección, por lo tanto sólo se podrá inyectar cada vez en un único punto de inyección.

Módulo inyector: Este módulo se encarga de producir la inyección física del fallo. Le llegan tres entradas, una indicando si el fallo a inyectar va a ser de pegado-a-0 ó de pegado-a-1, otra indicando la duración de la inyección del fallo y una tercera para inicializar el registro que almacena la señal de error efectivo. Tiene también una salida que se conecta con los relés y es donde se realiza la inyección.

El módulo incluye también un circuito detector de flancos, tanto de subida como de bajada, utilizado para señalizar la inserción de un error efectivo.

Módulo contador: Su función es la de llevar cuenta de los tiempos de latencia de los circuitos de detección de errores del sistema que se está analizando. En el momento en que el fallo inyectado se convierte en un error efectivo, el contador empieza a contar pulsos de reloj, hasta que le llega la señal de error detectado. En ese momento se para, almacenando el valor de la latencia de la detección del error. El contador funciona a una frecuencia máxima de 40 Mhz, por lo que tiene una resolución de 25 ns. Está compuesto por seis integrados conformando un contador de 24 bits. La latencia máxima que podrá medir será por tanto de 419 ms aproximadamente. La señal de desbordamiento del contador va conectada al analizador lógico para poder comprobar que la medida tomada es correcta.

Analizador lógico: El analizador lógico está compuesto por un puerto paralelo en la placa del PC, donde le llegan las señales más significativas del proceso de la inyección. La Tabla 6.3 recoge una lista de las señales, con su nombre y función

AI	Registro que indica que la inyección se ha activado
EE	Señal que registra si el fallo ha producido un error efectivo
FN	El circuito detector de flancos ha detectado un flanco negativo
FP	El circuito detector de flancos ha detectado un flanco positivo
/AI	Señal inversa de la señal de inyección activada (AI)
DE	Registro que indica la detección del error
/G	Señal que indica que está teniendo lugar la inyección

Tabla 6.3: Señales utilizadas por el analizador lógico

Módulo generador de las señales de reloj: El sistema de inyección funciona con dos frecuencias base de reloj. La primera, de 10 Mhz, se utiliza para temporizar los circuitos generadores del tiempo de inyección y la segunda, de 40 MHz para sincronizar los contadores del tiempo de latencia de detección del error.

Estas frecuencias son configurables, por medio de una serie de jumpers, pudiendo dividir la frecuencia por dos o por cuatro, con propósitos de depuración del hardware del inyector o simplemente para utilizarlo en sistemas lentos con grandes tiempos de latencia. Además también es posible mediante un jumper sustituir el reloj de 40 Mhz por otro de 4 Mhz.

8 Conclusiones

En este capítulo se presenta una introducción a la validación experimental de los sistemas tolerantes a fallos. Se introduce la inyección de fallos física, entre otras, como técnica para introducir fallos en un sistema y poder medir la bondad de los mecanismos de detección y de recuperación del sistema.

Se presenta el diseño de un inyector de fallos, modificado para realizar la inyección en altas frecuencias. Se ha realizado un diagrama de bloques del mismo y se ha explicado su funcionamiento, indicándose también sus características principales, como el conjunto de fallos que puede inyectar.

Capítulo 7

Validación del sistema FASST

1 Introducción

Se han perseguido dos objetivos a la hora de inyectar los fallos sobre el prototipo de sistema multiprocesador construido. El primero de ellos es el de calcular de forma experimental la cobertura de detección y de recuperación de fallos del sistema y el segundo el cálculo del tiempo de latencia de la detección de fallos. El primero se utilizará para dar valores a los parámetros en los modelos estocásticos de la fiabilidad, seguridad y disponibilidad del sistema y el segundo será útil para poder realizar un análisis sobre la violación de las propiedades que tiene el módulo de proceso de ser *fail-stop*.

Aunque el sistema inyector es completamente autónomo, es decir que no precisa de ningún otro equipo similar para realizar los experimentos de inyección, se ha utilizado un analizador lógico (Tektronix DAS 9200) para tomar las medidas de los resultados. El uso del analizador se debe principalmente a dos causas:

- Debido a la alta frecuencia de funcionamiento del sistema multiprocesador, las sondas, tanto las de inyección como las de toma de resultados (señal de error detectado), introducían una capacidad parásita muy grande. Esta capacidad se traduce en el retardo de las señales y en la generación de picos, siendo bastante difícil realizar el ajuste, por medio de filtros y divisores de tensión, del sistema de adquisición de datos. Las consecuencias del problema eran la presencia de errores previos al proceso de inyección que impedían el disparo del mismo.

- El sistema FASST incorpora varios mecanismos de detección de errores que funcionan en paralelo. Si se utiliza el inyector únicamente se puede capturar un evento por experimento, por lo que se tendrían que repetir las mismas experiencias de inyección, una para cada mecanismo de detección de fallos, para poder evaluarlas. Aparte del tiempo adicional que esto supone, también hay que tener en cuenta la dificultad que existiría en la repetición exacta de las condiciones previas a la realización del experimento.

La inyección se realiza mediante un proceso en donde cooperan tres elementos: el sistema tolerante a fallos objeto de la evaluación, el inyector de fallos y un el analizador lógico. Cada sistema incorpora una máquina de estados que dirige el proceso de la inyección y que se sincroniza con las demás. En la Figura 7.1 se puede ver un esquema que recoge el diagrama de bloques de los tres sistemas interconectados entre sí. Dentro del módulo de inyección están el hardware del inyector y un ordenador personal donde se ha colocado la placa de PC que es donde se ejecuta el software que implementa la máquina de estados del proceso de inyección.

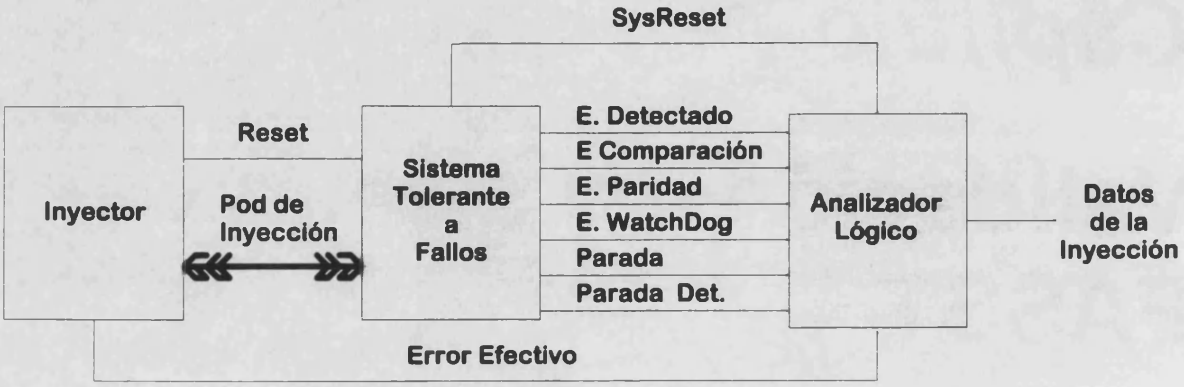


Figura 7.1: Diagrama de bloques del entorno de inyección

El número de experimentos realizados ha sido muy numeroso, bien debido a la necesidad de calibrar los mecanismos de adquisición de datos del inyector, bien porque el sistema es muy complejo y existen numerosos lugares donde es interesante realizar la inyección. Como se comenta anteriormente, la calibración no ha sido fácil, ya que debido a la "alta" frecuencia de funcionamiento del prototipo, se introducía mucho ruido sobre la línea de detección de fallos. Este problema impedía la correcta inyección del fallo puesto que el inyector está diseñado para que en el momento en que se detecte el error, automáticamente se cese de inyectar el fallo con el objeto de dañar lo menor posible al sistema que se está validando. Por este motivo, si en el momento de inyectar el fallo, la señal de error detectado está activa, la inyección no tiene lugar.

Antes de empezar a inyectar se decidió realizar un número de experimentos elevado. Existen trabajos que tratan de averiguar cual es el número de inyecciones que se deben de realizar en un sistema para obtener medidas con un intervalo de confianza correcto. Sin embargo, se ha preferido pecar por exceso que por defecto, procurando realizar una inyección exhaustiva con el objeto de obtener una muestra muy completa de cómo afectan los fallos al sistema, teniendo en cuenta la duración de los mismos, el lugar de inserción y el tipo de fallo. Al tener la muestra muy grande no importa eliminar algunas experiencias en donde queda la duda de si ha habido algún problema en el proceso de la inyección o en la toma de medidas.

La fotografía de la Figura 7.2 muestra el conjunto de todas las máquinas necesarias para realizar la inyección (el prototipo de Sistema FASST, sobre él el inyector y a la derecha el analizador lógico).

Figura 7.2: Fotografía del entorno de inyección y del prototipo

2 Descripción de la secuencia de inyección

A continuación se describe la secuencia de test utilizada para realizar la inyección de fallos. Esta secuencia está caracterizada por un dominio de entrada y un dominio de salida. El dominio de entrada se corresponde con el conjunto de los fallos a inyectar (Conjunto F) y los datos usados en el sistema en el momento de la activación (Conjunto A). El dominio de salida se corresponde con un conjunto de lecturas que caracterizan el comportamiento del sistema después de sufrir la inyección del fallo (Conjunto R) y un conjunto de medidas derivadas del procesamiento y el análisis de los conjuntos anteriores (Conjunto M).

Los conjuntos *FARM* [81] constituyen los atributos que se utilizan para caracterizar el experimento de la inyección. En la práctica, la secuencia de test de la inyección consiste en una serie de experimentos, donde cada experimento individual se corresponde con un punto en el espacio $\{F \mathrm{x} A \mathrm{x} R\}$.

2.1 El conjunto F

Este conjunto se corresponde con los fallos a inyectar. La siguiente tabla resume las características y número de fallos inyectados, teniendo en cuenta que el modelo de fallos elegido ha sido el de fallos de pegado-a, siendo el 50 % de pegado-a-0 y el otro 50 % de pegado-a-1.

		Lugar de la inyección			
		Proc. Maestro	Proc. Esclavo	Host Bus	CSR Bus
Tipo de fallo	Permanentes	2640	2640	750	630
	Transitorios	2640	2640	750	630
	Intermitentes	2640	2640	750	630
	Suma:	7920	7920	2250	1890
				Número total de fallos inyectados:	**19980**

La siguiente tabla resume las características temporales de los fallos. Cuando se especifica un rango de valores, la elección de la duración del fallo se realiza de forma aleatoria entre este rango, teniendo todos los valores la misma probabilidad de ser escogidos.

Tipo de fallo	Duración	Periodo
Fallo Permanente	1'2 mseg	0
Fallo Transitorio	100 ns - 4 useg	0
Fallo intermitente	100 ns - 2 useg	1 - 65 mseg

Los mecanismos de recuperación de errores transitorios se han diseñado para que se recuperen los fallos que duren un tiempo inferior a 32 ciclos de reloj. Como la inyección se ha realizado dividiendo la frecuencia del módulo de proceso por la mitad es de suponer que el tiempo máximo que puede durar un fallo son 2μs, por tanto, a priori, es interesante utilizar un intervalo de tiempo que ronde este valor. Si se repite un fallo transitorio antes de que se active la señal de #LOCK el sistema se detendrá, puesto que se supone que se trata de un fallo intermitente. La señal de #LOCK se activa como mínimo cada vez que se produce una interrupción. Existe una interrupción periódica en el sistema que se encarga de enviar los mensajes de "*estoy vivo*" cada 14'6 mseg, por lo que se ha considerado oportuno elegir un valor que oscile sobre este tiempo.

Teniendo en cuenta el lugar físico donde se van a inyectar los fallos, se han insertado un total de 30 fallos por pin, bien sea en el procesador, en el host bus o en el CSR bus. En la Tabla 7.1 se pueden observar el conjunto de pines donde se ha realizado la inyección:

Procesador Maestro	*Direcciones* *Datos* *Control*	A[31..2] D[31..0], DP[3..0] BE#[3..0], BS16#, BS8#, AHOLD, RESET, HOLD, INTR, BREQ#, BLAST#, HLDA, BRDY#, PCHK#, CLK, BOFF#, NMI, RDY#, ADS#, LOCK#, D/C#, M/IO#, W/R#
Procesador esclavo	*Direcciones* *Datos* *Control*	A[31..2] D[31..0], DP[3..0] BE#[3..0], BS16#, BS8#, AHOLD, RESET, HOLD, INTR, BREQ#, BLAST#, HLDA, BRDY#, PCHK#, CLK, BOFF#, NMI, RDY#, ADS#, LOCK#, D/C#, M/IO#, W/R#
Host Bus	*Direcciones* *Datos*	HA[16..0] HD[14..0]
CSR Bus	*Direcciones* *Datos*	CA[12..0] CD[7..0]

Tabla 7.1: Lugares de la inyección de fallos

2.2 El conjunto A

Este conjunto especifica los datos que se esta utilizando el sistema cuando se va a realizar la activación de la experiencia. En nuestro caso se trata de un programa muy simple que se ejecuta concurrentemente en ambos módulos de proceso.

La aplicación de usuario utilizada durante el experimento consiste en el dibujo por la pantalla de una serie de asteriscos, uno detrás de otro formando una fila. Al llegar al término de la fila, se borra y se comienza de nuevo a dibujar asteriscos por la parte izquierda de la pantalla en la misma fila. En paralelo se van produciendo interrupciones periódicas, cuyas rutinas de tratamiento se encargan del envío de mensajes de estoy vivo a todos los módulos del sistema mediante el uso de las interrupciones dirigidas que implementa Futurebus+.

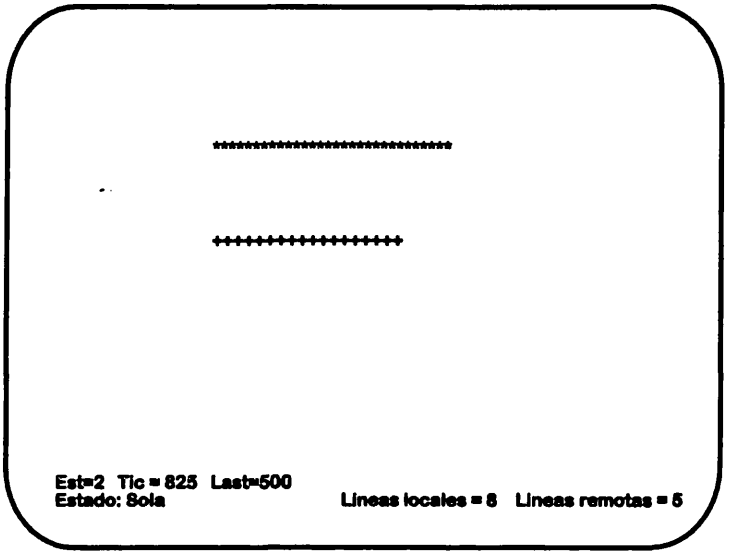


Figura 7.3: Ejemplo de ejecución de la aplicación

La aplicación además suministra información de diagnóstico en la parte inferior de la pantalla, habiendo una variable que indica el estado de los módulos del sistema. En el estado normal (`Estado: OK`), los dos módulos están funcionando correctamente y cada uno está sacando por su terminal las líneas de asteriscos. En el momento que falla uno de los módulos, éste deja de enviar los mensajes de "estoy vivo", y eventualmente el otro módulo detecta el fallo y comienza a escribir una línea de signos de sumar, indicando que el sistema se ha reconfigurado y se ha recuperado (`Estado: SOLA`). En el momento en que se vuelven a recibir los mensajes la aplicación del otro módulo ya no se ejecuta, y por tanto ya no se escriben signos de sumar. En la parte inferior de la pantalla se muestra un contador de las líneas locales y de las líneas remotas que se han escrito. Debido a que no funciona la placa de memoria, no se han podido almacenar en memoria compartida (se supone que en memoria estable) los valores de estas variables, para que sean únicas en el sistema. La idea inicial era almacenar un punto de recuperación cada vez que se escribe el último asterisco de la línea. Si un módulo falla, el otro lo que hace es leer el contador de líneas remotas de la memoria compartida (vuelta atrás) y continua con los signos de sumar escribiendo la línea donde se produjo la avería.

Las variables `Tic` y `Last` indican respectivamente el número de interrupciones periódicas que se han ejecutado desde que se reinicializó el sistema y el valor de Tic en el momento en que se recibió el último mensaje de "estoy vivo". La detección de la avería del otro módulo y el paso al estado de `SOLO` se realiza en el momento en que la diferencia entre `Tic` y `Last` sobrepasa un determinado umbral.

Para poder simular las averías de los módulos a voluntad y poder depurar el software, se ha programado la interrupción del interruptor externo de las placas para que cese el envío de mensajes de estoy vivo. Además se pasará al estado de `FALLO`. Si se vuelve a accionar el interruptor, se reanudará de nuevo el envío de mensajes.

Simplemente mediante la inspección visual del sistema se puede averiguar su estado. Si está encendido el led verde indica que el módulo está funcionando correctamente y que recibe los mensajes del otro. Si se enciende el amarillo, se ha detectado el fallo del otro módulo. Si se enciende el rojo es que el módulo esta en estado de FALLO y no está enviando mensajes.

2.3 El Conjunto R

En el analizador lógico se ha programado un autómata compuesto por estados y transiciones que se encarga de realizar la adquisición de datos. En la Figura 7.4 se puede observar el diagrama del autómata junto con la definición del significado de los estados. Se produce una transición entre estados cuando el analizador detecta una combinación específica de señales, definidas mediante palabras de disparo. Las señales de la palabra de disparo significan respectivamente: Error Efectivo, Inyección Activa, Reset, Error de Paridad, Error del Temporizador de Guardia, Parada, Error de Comparación y Recuperación. El funcionamiento del autómata es el siguiente: en cada estado se ha programado una serie de palabras de disparo distintas que producen cambios de estado. El analizador está continuamente monitorizando las señales de las sondas en espera de detectar alguna de estas palabras de disparo. En el momento que se detecta una de ellas se ejecutan una serie de acciones, que en nuestro caso son el disparo de la transición pasando hacia otro estado y la captura del estado de las señales junto con la información del temporizador (captura de datos dirigida por eventos). El proceso se repite hasta que se llena la memoria del analizador (16K eventos) o hasta que se interrumpe el proceso de forma manual. El analizador lógico inicialmente se encuentra en el estado 0 esperando a que se active la señal de reset.

Nº	Estado	Acciones
0	Inicialización	Espera la primera activación del Reset
1	Reset	Se almacena el tiempo de activación
2	Inyección activada	Se almacena el tiempo de inicio
3	Error efectivo	Cálculo de la latencia del error efectivo
4	Error detectado por el comparador	Cálculo de la latencia de detección del comparador
5	Sistema parado	Cálculo del tiempo de latencia de la detección del fallo. Captura de las señales EPar, EWD y Ecom
6	Error detectado por el otro módulo	Cálculo de la latencia de reconfiguración del sistema
7	Terminado el experimento	Espera la activación del Reset

	Palabra de disparo de las transiciones							
	EEf	/AIny	/RST	/Par	/WD	Hold	/Cmp	/Rec
#1	X	X	1	X	X	X	X	X
#2	X	0	X	X	X	X	X	X
#3	X	X	0	X	X	X	X	X
#4	X	X	X	X	X	X	0	X
#5	X	X	X	X	X	1	X	X
#6	X	X	X	X	X	X	X	0
#7	1	X	X	X	X	X	X	X

Figura 7.4: Diagrama del autómata de registro de eventos

El proceso de la inyección tiene lugar de la forma siguiente:

- En el ordenador personal donde se conecta el inyector se ejecuta un programa que dirige los experimentos. El programa consta de un bucle principal que se repite una vez para cada fallo que queramos inyectar. Las funciones que realiza son el control del inyector, eligiendo el tipo de fallo que se va a inyectar, su duración y el lugar de la inyección, y la reinicialización del prototipo del sistema tolerante a fallos. Esta última tarea es fundamental, puesto que todas las experiencias de inyección se deben realizar bajo las mismas condiciones iniciales. El proceso es sencillo: en cada iteración del bucle se resetea el sistema, se inyecta el fallo y se espera un tiempo prudencial para que se detecte el fallo.

  A continuación se puede observar el bucle principal del programa.

```
main() {

    inicializar_contr();
    inicializar_ports();
    desactivar_reles();

    inicializar_temporizadores()

    /*************************************/
    /*      Inicialización de secuencia     */
    /*           de inyección               */
    /*************************************/
    for(npin=0;npin<40;npin++)              /* Elige el pin a inyectar */
    for (tfallo=0;tfallo<2;tfallo++)        /* Elige entre pegado_a_0 o pegado_a_1 */
    for (j=0;j<15;j++) {
          nexp++;
          hacerreset(500);
          borrar_disparo();
          activar_rele(npin);
          poner_timer(0,MODO_3,600);        /* Duracion pulso = (nc1*nc0*100 ns) */
          poner_timer(1,MODO_1,200);
          d = 120000;
          cel(tfallo); /* Elige el tipo de fallo 0 o 1 */

    /*************************************/
    /*      Inyeccion de un fallo          */
    /*************************************/

          rescon();
          clred();         /* Desactiva error detectado */
          if (eel()==1) printf("Error en el latch de error efectivo\n");
          if (ltched()==1) printf("Error en el latch de error detectado\n");
          cin(1);                  /* Habilito el timer para inyectar  el fallo */
          delay(1000);             /* Tiempo que dura la inyeccion */

          /* Se desactiva la inyeccion */
          poner_timer(0,MODO_2,2000);
          printf("%d\t%d\t%s\t%d\t%ld\n",nexp,npin,pin_csr[npin],tfallo,d*100);
          desactivar_reles();
          cin(0);
          delay(2000);     /* Espera a reconfiguracion del sistema */
          clred();
          rescon();
    }
    hacerreset(500);
}
```

- Este mismo programa genera un fichero que recoge los datos de las experiencias de inyección. A continuación se puede ver una pequeña parte de él. Existen 5 campos, indicando sucesivamente el número de la experiencia de inyección, el número de pin donde

se inyecta el fallo, el nombre del pin, el tipo de fallo (si es de pegado-a-0 ó de pegado-a-1) y por último el tiempo que dura la activación del fallo.

```
Inyeccion de fallos permanentes en pin interior
Tiempo de inyección: 12000000 ns
#       #pin    nombre  p_a_    iny
1       1       HD2     0       12000000
2       1       HD2     0       12000000
210     8       HA4     1       12000000
211     9       HA5     0       12000000
```

- Una vez realizada la inyección se deben tomar los resultados y recoger las consecuencias que produce la activación del fallo en el sistema. Como ya se ha comentado anteriormente se utiliza el analizador lógico de Tektronik para este propósito. Las sondas del analizador se encargan de recoger el estado de la señal de RESET del Futurebus+, las señales del analizador lógico que indican el comienzo de la inyección y que el fallo inyectado es efectivo, la señal de error de comparación que manda el comparador al votador final, la señal de error de paridad de los procesadores, la señal de error del temporizador de guardia y por último la señal que enciende el led amarillo del otro módulo del sistema indicando una reconfiguración.

- En el analizador lógico se empiezan a capturan los tiempos de latencia en base al autómata dirigido por eventos descrito previamente (ver Figura 7.4). Una vez que se detecta la activación de la señal de RESET, se almacena el tiempo absoluto del momento en que ha ocurrido y se pasa a un nuevo estado quedando en espera de recoger nuevos eventos dependiendo del comportamiento del sistema. La resolución que se ha utilizado en las medidas permite distinguir tiempos de latencia de hasta 2'5 nanosegundos de duración. En el caso más favorable se detectará secuencialmente la activación de las siguientes señales: comienzo de la inyección, error efectivo, error de comparación, parada del módulo, señal que enciende el led amarillo en el otro módulo y por último la de RESET indicando que ha terminado el experimento. Mediante esta secuencia ejemplo se sabe que el fallo:

  Ha sido efectivo
  Ha sido detectado por los comparadores (cálculo de la latencia)
  Ha producido la detención del módulo (cálculo de la latencia)
  Ha reconfigurado el sistema (cálculo de la latencia)

- El fichero es sacado del analizador lógico por el puerto serie para su posterior procesamiento. A continuación se puede ver el formato del fichero tal y como se recibe inicialmente.

```
12 Jan 1997 10:59                                        DAS 9200 92A96-1 State
                                                                      Page 2

Sequence   Control  Group_A  Group_B        Time stamp
-------------------------------------------------------------------------------
      93   01       01101    1              37.592,331,993 s
      94   01       11101    1              37.592,331,995 s
      95   01       11101    1              37.592,331,998 s
      96   01       11101    1              40.093,665,710 s
      97   01       11101    1              40.093,665,713 s
      98   01       11101    1              40.093,665,715 s
      99   00       11101    1              40.093,665,718 s
     100   00       11101    1              40.093,665,760 s
     101   10       11101    1              40.093,665,763 s
     102   10       11101    1              40.093,665,765 s
     103   10       11101    1              40.093,665,768 s
     104   01       11101    1              42.439,844,180 s
     105   01       01101    1              42.439,844,183 s
```

```
106  01      01101   1          42.439,844,185 s
107  01      01101   1          42.439,844,188 s
108  01      01101   1          42.442,547,740 s
109  01      01101   1          42.442,547,743 s
110  01      11101   1          42.442,547,745 s
```

- El análisis consiste en procesar la información que muestra el fichero. Para ello se han realizado una serie de filtros que se encargan respectivamente de:

a) Formatear el fichero para separar los campos y eliminar información redundante.

```
93  0    1    0    1    1    0    1    1    37.592,331,993
94  0    1    1    1    1    0    1    1    37.592,331,995
95  0    1    1    1    1    0    1    1    37.592,331,998
96  0    1    1    1    1    0    1    1    40.093,665,710
97  0    1    1    1    1    0    1    1    40.093,665,713
98  0    1    1    1    1    0    1    1    40.093,665,715
99  0    0    1    1    1    0    1    1    40.093,665,718
100 0    0    1    1    1    0    1    1    40.093,665,760
101 1    0    1    1    1    0    1    1    40.093,665,763
102 1    0    1    1    1    0    1    1    40.093,665,765
103 1    0    1    1    1    0    1    1    40.093,665,768
104 0    1    1    1    1    0    1    1    42.439,844,180
105 0    1    0    1    1    0    1    1    42.439,844,183
106 0    1    0    1    1    0    1    1    42.439,844,185
107 0    1    0    1    1    0    1    1    42.439,844,188
108 0    1    0    1    1    0    1    1    42.442,547,740
109 0    1    0    1    1    0    1    1    42.442,547,743
110 0    1    1    1    1    0    1    1    42.442,547,745
```

b) Eliminar las líneas iguales donde no se produce ningún cambio.

c) Eliminar los posibles pulsos no deseados quitando de nuevo las líneas en donde una señal cambia de estado dos veces en menos de 90 nanosegundos.

```
94  0    1    1    1    1    0    1    1    37.592,331,995
99  0    0    1    1    1    0    1    1    40.093,665,718
101 1    0    1    1    1    0    1    1    40.093,665,763
104 0    1    1    1    1    0    1    1    42.439,844,180
105 0    1    0    1    1    0    1    1    42.439,844,183
110 0    1    1    1    1    0    1    1    42.442,547,745
```

d) Contar las experiencias realizadas por medio de la monitorización de la señal de reset.

e) Discernir sobre cual ha sido el efecto del fallo.

f) Calcular los tiempos de latencia del error de comparación, de la parada del sistema y de reconfiguración y recuperación del sistema.

g) Sacar los resultados en un formato tabulado listo para integrar en una base de datos.

```
nexp    evento tlatee  tlatcmp tlatpar tlatrec /pchk   /wd     /comp   codigo  t_fal   t_pin
0       0      0       0       0       0       0       0       0       0       1       3
1       9      42      0       0       0       1       1       1       1       1       3
2       25     42      0       0       0       1       1       1       1       1       3
3       43     40      0       0       0       1       1       1       1       1       3
4       58     -1      0       0       0       1       1       1       1       1       3
5       77     45      0       0       0       1       1       1       1       1       3
6       94     45      0       0       0       1       1       1       1       1       3
7       110    42      0       0       0       1       1       1       1       1       3
8       125    42      0       0       0       1       1       1       1       1       3
```

h) Unir estos resultados con los datos de la experiencia de inyección y exportar el conjunto a la base de datos o a un programa estadístico y poder comenzar con el análisis.

#	#pin	nombre	p_a_	iny	tlatee	tlatcmp	tlatpar	tlatrec	/pchk	/wd	/comp	codigo	t_fal	t_pin
1	5	HA16	0	12000000	42	0	0	0	1	1	1	1	1	3
2	5	HA16	0	12000000	42	0	0	0	1	1	1	1	1	3
3	5	HA16	0	12000000	40	0	0	0	1	1	1	1	1	3
4	5	HA16	0	12000000	-1	0	0	0	1	1	1	1	1	3
5	5	HA16	0	12000000	45	0	0	0	1	1	1	1	1	3
6	5	HA16	0	12000000	45	0	0	0	1	1	1	1	1	3
7	5	HA16	0	12000000	42	0	0	0	1	1	1	1	1	3

En este fichero se pueden observar los siguientes campos:

Número experiencia	Lleva cuenta de las experiencias realizadas.
Número pin	Código del pin donde se realiza la inyección.
Nombre pin	Nombre del pin donde se realiza la inyección.
Tipo fallo	Fallo de pegado-a-0 ó de pegado-a-1.
Duración fallo	Tiempo de duración de cada inyección.
Tiempo entre fallos	Tiempo entre dos fallos intermitentes.
Tiempo latencia Error Efectivo	Tiempo que tarda en detectarse que el fallo inyectado está produciendo un error.
Tiempo latencia Detección comparador	Tiempo que transcurre desde que se activa el error efectivo y se produce un error de comparación.
Tiempo latencia parada módulo	Tiempo que transcurre desde que se activa el error efectivo hasta que se activa la señal HOLD.
Tiempo latencia recuperación	Tiempo que transcurre desde que se activa el error efectivo hasta que el otro módulo procesador pasa al estado de SOLO.
Código del suceso	Código del tipo de evento registrado.
Código del tipo de fallo	Distingue entre fallos permanentes, transitorios o intermitentes.
Código del lugar de inyección	Distingue entre procesador maestro, procesador esclavo, Host Bus o CSR Bus.

2.4 El conjunto M

Este conjunto está formado por los resultados y conclusiones obtenidas fruto del análisis de los datos. Para realizar el procesamiento de los datos se ha utilizado una hoja de cálculo (Microsoft Excel) y el paquete estadístico SPSS. Ambos programas toman como datos el último fichero generado del *Conjunto R*, que describe el comportamiento del sistema.

El primer paso en el procesamiento de los datos va a tener el objetivo de eliminar los casos en donde se haya detectado algún error producido por el propio inyector. Estos errores se pueden detectar puesto que para adquirir los datos se ha utilizado un patrón fijo de acciones, delimitándose de este modo las posibilidades que se pueden encontrar en los cambios de estado de las señales. Piénsese como ejemplo el caso en el que se detecta un error efectivo antes de activar la inyección, o el caso en el que esté activa la línea de error de comparación mientras el sistema está en estado de reset. Este último no supone un error en el módulo, sino un problema en la sonda encargada de medir el estado de la línea. La Tabla 7.2 describe el conjunto de situaciones válidas que se pueden producir en el experimento. Los casos que no se ajusten a ninguno de esta tabla no se tendrán en cuenta para el análisis.

Cód	/Comp	/Par	/WD	$T_{latcomp}$	$T_{latstop}$	T_{rec}	Evento
1	1	1	1	0	0	0	Este evento supone la existencia de un error efectivo, pero no propagado al sistema. El resultado por tanto es que el funcionamiento del sistema no se modifica
2	1	1	1	0	0	TR	Error no detectado. Se ha producido un fallo en la cobertura de detección. El sistema eventualmente se reconfigura
3	0 1 1	1 0 1	1 1 0	- - TC	TP	0	Fallo del sistema. Se detecta el error pero fallan los mecanismos de recuperación del mismo y por tanto no se reconfigura el sistema
4	0	1	1	-	TP	TR	Error de comparación
5	1	0	1	-	TP	TR	Error de paridad
6	1	1	0	TC	TP	TR	Error del temporizador de guardia
7	1	1	0	0	TP	TR	El módulo se detiene pero sin detectarse la causa de la parada. El sistema se reconfigura normalmente.
8	1	1	1	TC	0	0	Fallo transitorio recuperado
9	1	1	1	TC	0	TR	Se produce una degradación del sistema debido a que se tarda demasiado tiempo en recuperar el fallo transitorio.
10	1	1	0	0	TP	0	El módulo se detiene pero sin detectarse la causa de la parada. También falla el proceso de reconfiguración del sistema.

Si aparece TC, TP ó TR indica que el valor de este parámetro es distinto de cero
Si aparece un guión indica que no importa su valor

Tabla 7.2: Eventos recogidos de la experiencia de inyección

La siguiente tabla muestra el número de experiencias inválidas obtenidas después de este primer análisis. Entre paréntesis se incluye el porcentaje sobre el total de las experiencias.

		Número de experiencias inválidas			
		Proc. Maestro	Proc. Esclavo	Host Bus	CSR Bus
Tipo de fallo	Permanentes	1295 (49 %)	1396 (53 %)	386 (51 %)	276 (44 %)
	Transitorios	1128 (43 %)	1171 (44 %)	322 (43 %)	215 (34 %)
	Intermitentes	1132 (43 %)	1920 (73 %)	335 (45 %)	229 (36 %)
	Suma:	3555 (45 %)	4487 (57 %)	1043 (46 %)	720 (38 %)
				Total:	9805 (49 %)

Como se puede observar el número de casos desechados casi ronda el 50 %, debido precisamente a los problemas de ruido del inyector y también a problemas de conexión de las sondas del analizador lógico. Por ejemplo, en el caso de los errores intermitentes inyectados en el procesador esclavo, se detectó una mala conexión de la señal de error efectivo procedente del inyector, de ahí que el porcentaje de experiencias inválidas sea mucho mayor en este caso.

El siguiente filtro que se le pasa a los datos consiste en eliminar los experimentos que no han producido un error efectivo. El objetivo de la inyección del fallo es producir un error en el sistema, justo en el lugar de la inserción del fallo. Debido a que el momento de disparo de la inyección se elige de forma aleatoria, no se puede garantizar que se vaya a producir un error efectivo. No se producirá el error cuando se fuerce la línea a pasar a un estado en el que ya estaba. Por ejemplo cuando inyecto un cero en una línea que está a nivel bajo, o un uno en una línea que está a nivel alto. La siguiente tabla muestra los porcentajes de errores no efectivos, obtenidos de las experiencias válidas.

		Número de errores no efectivos			
		Proc. Maestro	Proc. Esclavo	Host Bus	CSR Bus
Tipo de fallo	Permanentes	138 (10 %)	121 (10%)	1 (0 %)	41 (12 %)
	Transitorios	133 (9 %)	239 (16 %)	5 (1 %)	20 (5 %)
	Intermitentes	108 (7 %)	77 (11 %)	2 (0 %)	10 (2 %)
	Suma:	379 (9%)	437 (13 %)	8 (1 %)	71 (6 %)
				Total:	895 (9 %)

En la tabla anterior sí que se aprecia la diferencia existente entre inyectar fallos en el bus del procesador, o inyectar fallos en el Host Bus y en el bus del CSR. Las diferencias se producen por dos causas: la existencia de señales con un nivel lógico que no varía en el tiempo con facilidad (los errores efectivos son menores) y el porcentaje de utilización del bus en el tiempo que hace que las líneas estén en alta impedancia (mayor número de errores efectivos, puesto que al inyectar el fallo, siempre cambia el nivel).

A continuación se incluyen los parámetros obtenidos después de realizar el análisis con los datos. Se han separado en otras secciones para ganar en claridad debido a su extensión, sin embargo hay que hacer notar de forma explícita que todos los resultados obtenidos son los que forman el *Conjunto R*.

3 Cálculo de las coberturas de detección y de recuperación

La Figura 7.5 muestra un diagrama del modelo de cobertura de un sistema multiprocesador con degradación. Este modelo no sólo contempla la ocurrencia de fallos transitorios y permanentes, sino que también es apropiado para los módulos de proceso *fail-stop*, puesto que tiene en cuenta la evolución del sistema ante violaciones de parada después de la avería. Cuando se produce un error efectivo, dependiendo de si se detecta o no, el sistema evoluciona hacia el estado de error efectivo detectado o al de error efectivo no detectado. Es importante resaltar también que el modelo está pensado también para la inyección de fallos, puesto que contempla el caso de los errores efectivos no propagados, muy abundantes en cualquier experiencia de inyección debido a la redundancia intrínseca del sistema. Las transiciones del modelo se basan en la probabilidad de pasar de un estado a otro, que depende de la cobertura de detección de fallos de los mecanismos hardware incorporados dentro del módulo. Dentro del modelo se pueden distinguir 5 salidas:

- **FD**: El sistema detecta el error efectivo que se ha producido
- **FND**: El sistema no detecta el error efectivo que se ha producido
- **FNP**: Indica la existencia de un error efectivo, pero que sin embargo, debido a las características intrínsecas de tolerancia a fallos de cualquier sistema no produce la propagación del error. Piénsese por ejemplo en la inyección de un fallo en una línea en alta impedancia. Como el estado de la línea está flotando, al forzar un pegado-a, seguro que el sistema de inyección detecta que el fallo es efectivo. Sin embargo, las consecuencias de ese error son nulas. Esta situación se debe a la existencia de redundancia no intencionada en el sistema, y puede servirnos para calcular el factor de cobertura de la tolerancia a fallos intrínseca del sistema.
- **R**: Indica la recuperación de un fallo temporal
- **C**: Indica la degradación segura del sistema. En esta salida lo que indica es que el sistema multiprocesador se reconfigura debido, bien a una parada de uno de los módulos, bien a que el proceso de recuperación de fallos transitorios se prolonga demasiado y no se puede

enviar el mensaje de "*estoy vivo*" a tiempo. En cualquier caso el módulo, o se para o no tienen ningún error, por lo tanto no se produce ninguna violación de las características de "parada ante fallo" (*fail-silent*) del módulo.

- **CV**: Cuando se alcanza este estado es debido a que el sistema se ha degradado, ya que una de las placas ha dejado de funcionar. La diferencia con el estado anterior estriba en que en este caso, el módulo averiado no ha podido detectar el error y por lo tanto no se tiene la certeza de que el módulo se ha parado a tiempo, sin contaminar el sistema. Este caso representa por tanto la reconfiguración no segura del sistema.

- **NR**: Este estado supone la producción de un error que no se ha podido recuperar. El sistema se para en un estado seguro puesto que el módulo averiado se ha detenido. La causa de la detección puede no ser conocida.

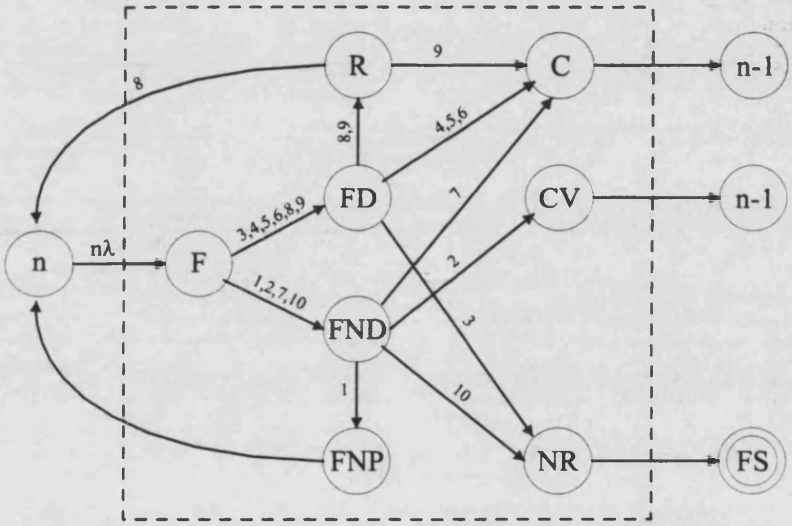


Figura 7.5: Modelo microscópico del sistema

Cada transición está etiquetada con un número que se corresponde con un comportamiento diferente del sistema. El significado de cada número se detalla en la Tabla 7.2. Por ejemplo, si el sistema está funcionando con los dos módulos activos, y se produce una situación 2, el fallo no es detectado, pero sin embargo el sistema se reconfigura eventualmente, evolucionando desde el estado n al F, FND, CV y terminando en el estado en donde el sistema se ha degradado funcionando ahora un solo módulo.

Una vez descrito el modelo queda analizar los datos recogidos de la experiencia para poder asignar las probabilidades a las transiciones, y por tanto poder resolver el modelo de forma analítica o mediante simulación.

La Tabla 7.3 recoge los valores de las tasas de transición entre estados del modelo de la Figura 7.5. Estos valores indican la probabilidad de que se de un caso en concreto de los representados en el modelo macroscópico.

Caso	1	2	3	4	5	6	7	8	9	10
Frecuencia	1821	8	176	3439	2598	252	448	222	0	316
Porcentaje	19'6%	0'1%	1'9%	37'1%	28%	2'7%	4'8%	2'4%	0%	3'4%

Tabla 7.3:Tasas de transición del modelo microscópico

Como se puede observar, el mayor número de casos corresponde con los errores detectados por los comparadores y por la paridad. En contraposición no se ha detectado ningún caso 9, como era de esperar, puesto que los mecanismos de recuperación de errores transitorios tienen una latencia de cientos de nanosegundos de duración, y la reconfiguración del sistema se produce cuando expira la cuenta de un temporizador, con un orden de magnitud de cientos de milisegundos.

El haber realizado una inyección de fallos de diversa duración sobre varias zonas del módulo de proceso, permite hacer muchos análisis diferentes sobre los datos. A continuación se presentan dos tablas de contingencia que se utilizan para contar el número de casos que tienen las diferentes combinaciones de valores de dos variables, mostrando además las pruebas y estadísticos bivariados, que expresan en que medida están relacionadas las dos variables. La primera de ellas muestra la respuesta que tiene el sistema a los fallos de diversa duración. En cada celda se pueden observar cinco parámetros cuyo significado recoge la tabla siguiente:

Frecuencia observada	Número observado de casos de la celda.
Frecuencia esperada	Número de casos que cabría esperar dentro de la celda si las variables de fila y columna fueran estadísticamente independientes o no relacionadas entre sí.
Porcentaje de fila	Porcentaje respecto a todos los casos de una fila que están en una celda
Porcentaje de columna	Porcentaje respecto a todos los casos de una columna que están en una celda.
Residuo no tipificado	Frecuencia observada menos la frecuencia esperada. Un residuo positivo indica que hay más casos en la celda de los que habría si las variables de fila y columna fueran independientes.

Además, al final de la tabla también se ha considerado oportuno mostrar el estadístico Ji-Cuadrado de Pearson, utilizado para verificar la hipótesis de que las variables de filas y columnas son independientes. No debe utilizarse si cualquiera de las celdas tiene un valor esperado menor que uno o si más de un 20% de las celdas tienen valores esperados menores que 5.

Las siguientes tablas recogen la influencia que tiene el tipo de fallo insertado y el lugar de la inserción en los mecanismos de detección y recuperación de fallos. Como se puede observar el coeficiente de homogeneidad es nulo, por lo tanto no existe relación entre las variables aleatorias.

Freq. Esperada / Freq. Observada / % de fila / % de columna / Res no tipificado	Duración del fallo			
	Perman	Transi	Interm	Fila Total
CASOS 1,000	386	1096	339	1821
	589,9	672,5	558,7	19,6%
	21,2%	60,2%	18,6%	
	12,8%	32,0%	11,9%	
	-203,9	423,5	-219,7	
2,000	5	0	3	8
	2,6	3,0	2,5	,1%
	62,5%	,0%	37,5%	
	,2%	,0%	,1%	
	2,4	-3,0	,5	
3,000	66	64	46	176
	57,0	65,0	54,0	1,9%
	37,5%	36,4%	26,1%	
	2,2%	1,9%	1,6%	
	9,0	-1,0	-8,0	
4,000	1475	691	1273	3439
	1114,0	1270,0	1055,0	37,1%
	42,9%	20,1%	37,0%	
	49,1%	20,2%	44,7%	
	361,0	-579,0	218,0	
5,000	793	976	829	2598
	841,6	959,4	797,0	28,0%
	30,5%	37,6%	31,9%	
	26,4%	28,5%	29,1%	
	-48,6	16,6	32,0	
6,000	44	163	45	252
	81,6	93,1	77,3	2,7%
	17,5%	64,7%	17,9%	
	1,5%	4,8%	1,6%	
	-37,6	69,9	-32,3	
7,000	136	127	185	448
	145,1	165,4	137,4	4,8%
	30,4%	28,3%	41,3%	
	4,5%	3,7%	6,5%	
	-9,1	-38,4	47,6	
8,000	1	221	0	222
	71,9	82,0	68,1	2,4%
	,5%	99,5%	,0%	
	,0%	6,4%	,0%	
	-70,9	139,0	-68,1	
10,000	100	89	127	316
	102,4	116,7	96,9	3,4%
	31,6%	28,2%	40,2%	
	3,3%	2,6%	4,5%	
	-2,4	-27,7	30,1	
Columna Total	3006	3427	2847	9280
	32,4%	36,9%	30,7%	100,0%

Test Ji-Cuadrado	Valor	DF	Homogeneidad
Pearson	1360,93893	16	,00000

Freq. Esperada / Freq. Observada / % de fila / % de columna / Res no tipificado	Lugar de la inyección				
CASOS	ProcA	ProcB	Host B	CSR B	Fila Total
1,000	421	608	76	716	1821
	782,2	587,7	235,5	215,7	19,6%
	23,1%	33,4%	4,2%	39,3%	
	10,6%	20,3%	6,3%	65,2%	
	-361,2	20,3	-159,5	500,3	
2,000	0	0	1	7	8
	3,4	2,6	1,0	,9	,1%
	,0%	,0%	12,5%	87,5%	
	,0%	,0%	,1%	,6%	
	-3,4	-2,6	,0	6,1	
3,000	166	7	1	2	176
	75,6	56,8	22,8	20,8	1,9%
	94,3%	4,0%	,6%	1,1%	
	4,2%	,2%	,1%	,2%	
	90,4	-49,8	-21,8	-18,8	
4,000	1752	1687	0	0	3439
	1477,1	1109,9	444,7	407,3	37,1%
	50,9%	49,1%	,0%	,0%	
	44,0%	56,3%	,0%	,0%	
	274,9	577,1	-444,7	-407,3	
5,000	1218	480	722	178	2598
	1115,9	838,5	335,9	307,7	28,0%
	46,9%	18,5%	27,8%	6,9%	
	30,6%	16,0%	60,2%	16,2%	
	102,1	-358,5	386,1	-129,7	
6,000	232	20	0	0	252
	108,2	81,3	32,6	29,8	2,7%
	92,1%	7,9%	,0%	,0%	
	5,8%	,7%	,0%	,0%	
	123,8	-61,3	-32,6	-29,8	
7,000	87	67	150	144	448
	192,4	144,6	57,9	53,1	4,8%
	19,4%	15,0%	33,5%	32,1%	
	2,2%	2,2%	12,5%	13,1%	
	-105,4	-77,6	92,1	90,9	
8,000	96	126	0	0	222
	95,4	71,6	28,7	26,3	2,4%
	43,2%	56,8%	,0%	,0%	
	2,4%	4,2%	,0%	,0%	
	,6	54,4	-28,7	-26,3	
10,000	14	0	250	52	316
	135,7	102,0	40,9	37,4	3,4%
	4,4%	,0%	79,1%	16,5%	
	,4%	,0%	20,8%	4,7%	
	-121,7	-102,0	209,1	14,6	
Column Total	3986	2995	1200	1099	9280
	43,0%	32,3%	12,9%	11,8%	100,0%

Test Ji-Cuadrado	Valor	DF	Homogeneidad
Pearson	5569,96045	24	,00000

En la Figura 7.6 se puede ver un grafo de predicados que se correspondiente con el modelo microscópico simplificado del sistema FASST y que resume las características de la garantía de funcionamiento del mismo, teniendo en cuenta también el proceso de la inyección de fallos. En el diagrama se aprecian los estados por los que va evolucionando el sistema fruto del proceso de producción de fallos y de la activación de los algoritmos y mecanismos de tolerancia a fallos en él implementados. Este grafo describe completamente la cobertura de detección de fallos y de recuperación del prototipo del Sistema FASST.

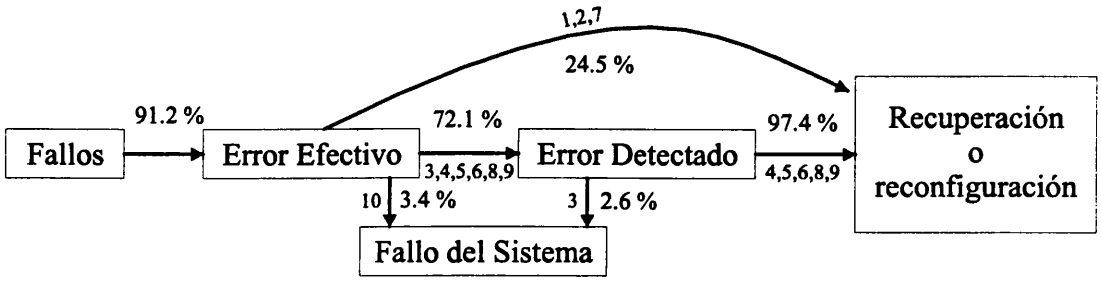


Figura 7.6: Grafo de predicados del sistema

4 *Cálculo de los tiempos de latencia*

Los siguientes gráficos representan las funciones de distribución de la cobertura de fallos en función del tiempo. Se han analizado la cobertura de detección de fallos de los comparadores, de los errores de paridad y de los errores que afectan a la temporización del flujo de instrucciones detectados por el temporizador de guardia a lo largo del tiempo. Por una parte, en la gráfica de la Figura 7.7 se ha representado una comparación entre los tres mecanismos de detección de errores y en la gráfica de la Figura 7.8 se han sumado las tres funciones para poder cuantificar la cobertura total de detección de errores del sistema.

En esta primera gráfica se representan superpuestas tres funciones que representan la evolución a lo largo del tiempo, respectivamente, de la cobertura de detección de errores de los comparadores, de la paridad y del temporizador de guardia. En el eje de abcisas tenemos el tiempo transcurrido desde que se activa el error efectivo hasta que es detectado. En el eje de ordenadas tenemos el porcentaje de casos en los que el error es detectado en un tiempo igual o inferior al dado con respecto a todos los errores inyectados en el experimento.

Como se puede apreciar, los comparadores tienen mayor cobertura de detección de errores que los demás (aproximadamente del 45 %) y un tiempo de latencia en la detección menor, puesto que prácticamente en 1 microsegundo ya han detectado el 90 % de todos los errores que deben detectar. El tiempo de latencia mayor se corresponde con la paridad, puesto que hasta que no pasan 300 nanosegundos no se observa la detección de ningún error. Aunque parezca extraño que la latencia de la paridad sea mayor que la del temporizador de guardia, este efecto se debe a que muchos errores inyectados no afectan a la paridad.

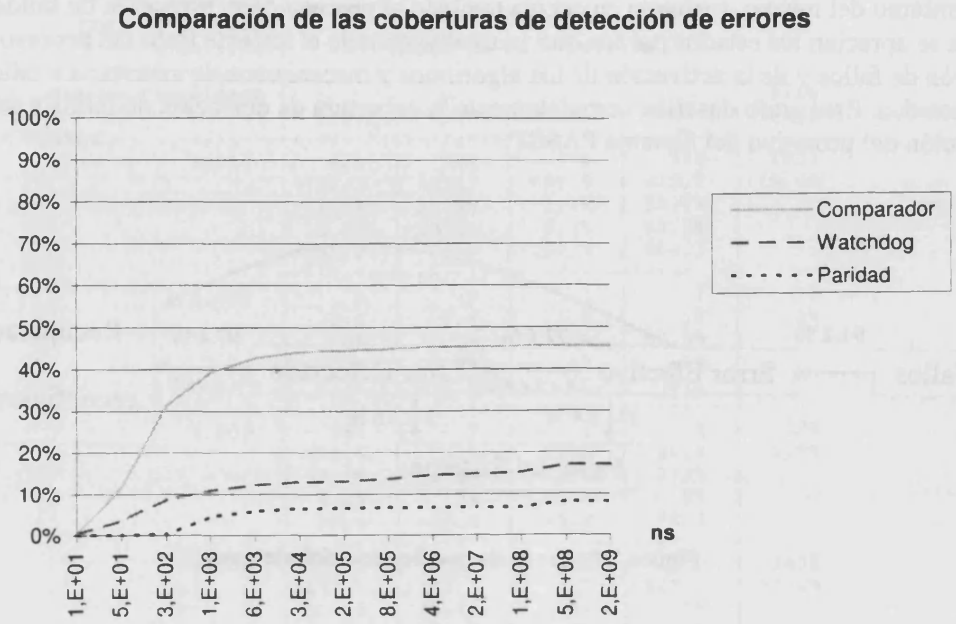


Figura 7.7: Coberturas de detección de errores

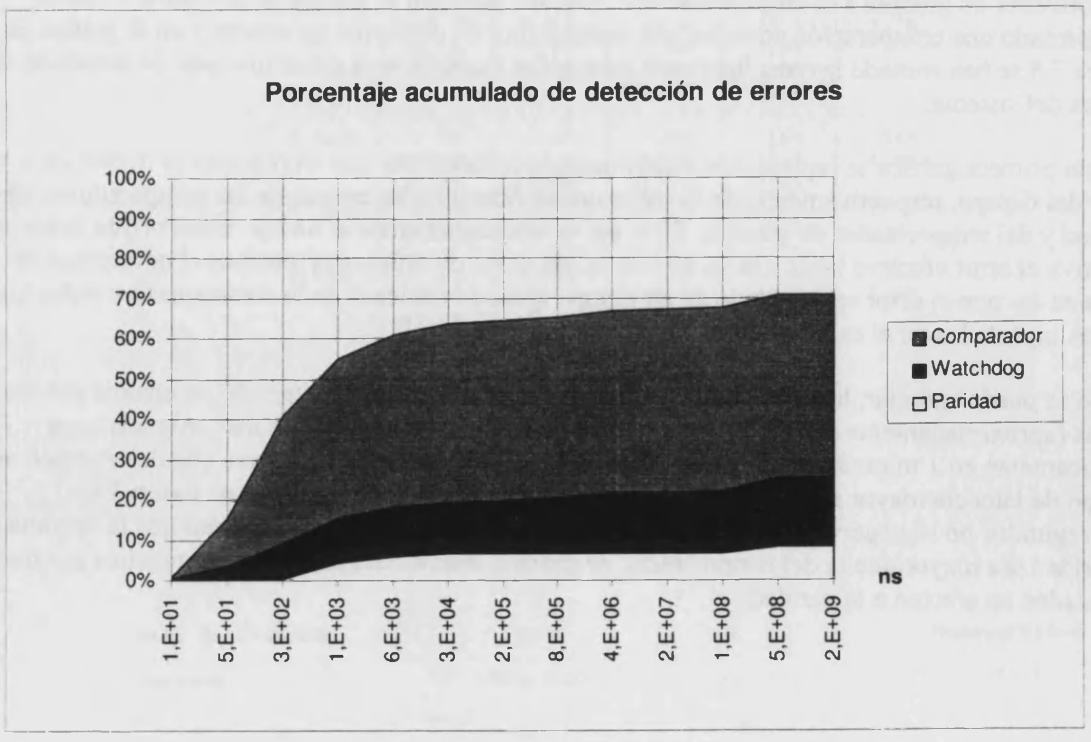


Figura 7.8: Cobertura total de detección de errores

En la gráfica de la Figura 7.8 se representa la evolución a lo largo del tiempo de la cobertura de detección de errores global de los mecanismos de tolerancia a fallos incluidos en los módulos de proceso. Para su representación lo que se ha hecho ha sido sumar los porcentajes de detección de errores de los comparadores, la paridad y el temporizador de guardia. En eje de abcisas representa el tiempo transcurrido desde que se activa el error efectivo hasta que es detectado. En el eje de ordenadas tenemos el porcentaje de errores detectados con respecto a todos los errores inyectados en el experimento. Lo que realmente importa en la gráfica es la función superior, que es la que representa el porcentaje global de la detección.

La gráfica muestra que prácticamente el 90 % de los errores que va a detectar el módulo lo va a hacer con una latencia de 6 microsegundos, que es un tiempo bastante bueno. Cuando ya han transcurrido los 6 us, la probabilidad de tener un error no detectado es bastante grande.

A continuación se ha calculado para cada tiempo de latencia los estadísticos univariados más usuales que describen a estas variables. En la siguiente tabla se indica el significado de cada uno, y se incluye también la abreviatura que utiliza el programa SPSS para referenciarlos.

Media	Mean	Es el promedio aritmético de los valores de la variable aleatoria.
Mediana	Median	Valor que es mayor que la mitad de los datos reales y menor que la otra mitad. Es el percentil 50.
Moda		El valor que ocurre con mayor frecuencia. Si varios valores comparten la mayor frecuencia de aparición, cada uno de ellos es una moda.
Media recortada al 5 %	5% Trim	Media de la muestra eliminando el 5% de los valores extremos
Error típico para la media	Std Err	Medida de cuánto puede variar el valor de la media de una muestra a otra extraídas éstas de la misma distribución. Es la desviación típica de la distribución de todas las posibles medias, si se toman repetidamente muestras del mismo tamaño.
Varianza	Variance	Mide la dispersión alrededor de la media, igual a la suma de los cuadrados de las desviaciones respecto a la media dividida por el número de casos menos 1.
Desviación típica	Std Dev	Mide la dispersión en torno a la media. Equivale a la raíz cuadrada de la varianza.
Mínimo	Min	Valor más pequeño de la muestra.
Máximo	Max	Valor más grande de la muestra.
Amplitud	Range	Diferencia entre los valores más grande y más pequeño.
Amplitud intercuartil	IQR	Diferencia entre el valor mayor y el menor, eliminando el 25% de valores en cada extremo.
Asimetría	Skewness	Medida de la asimetría o sesgo de la distribución. La asimetría positiva indica que los valores más extremos son mayores que la media; la asimetría negativa indica que los valores más extremos son menores que la media.
Error típico de la asimetría	S E Skew	Es el error típico de este estadístico.
Curtosis	Kurtosis	Mide el grado en que una distribución está cargada en sus colas comparada con una distribución normal. La curtosis positiva indica más casos en los extremos de las colas que en una distribución normal con la misma varianza.
Error típico de la curtosis	S E Kurt	Es el error típico de este estadístico.
Intervalo de confianza para la media al 99%	99% CI for Mean	Existe una probabilidad del 99 % de que el intervalo especificado en torno al valor de la media, obtenido mediante una estimación estadística de una muestra, incluya el valor real que se calcularía a partir de la población total (si esto fuera posible).

Como tenemos datos muy dispares debido a los errores en la toma de medidas, el parámetro más significativo para hacernos una idea de los tiempos de latencia de detección medios es la mediana. Se puede observar como la latencia de la comparación es mucho menor que la de la paridad o la del

watch-dog. Otro parámetro que se incluye es el intervalo de confianza, que se ha elegido del 99 % puesto que el tamaño de la muestra es bastante grande.

Tiempo de latencia de detección de errores del comparador:

Mean	19645570	Std Err	1804644	Min	3,0000	Skewness	5,9936
Median	528,0000	Variance	1,41E+16	Max	7,64E+08	S E Skew	,0372
5% Trim	4454,684	Std Dev	1,19E+08	Range	7,64E+08	Kurtosis	34,0816
99% CI for Mean (14995070; 24296071)				IQR	1870,000	S E Kurt	,0743

Tiempo de latencia de detección de errores de la paridad:

Mean	1,22E+08	Std Err	10107643	Min	228,0000	Skewness	1,8425
Median	5242,000	Variance	7,71E+16	Max	7,66E+08	S E Skew	,0890
5% Trim	93483531	Std Dev	2,78E+08	Range	7,66E+08	Kurtosis	1,3988
99% CI for Mean (95933285; 1,48E+08)				IQR	33010,00	S E Kurt	,1777

Tiempo de latencia de detección de errores del temporizador de guardia:

Mean	87637428	Std Err	5896448	Min	23,0000	Skewness	2,4347
Median	1145,000	Variance	5,51E+16	Max	7,67E+08	S E Skew	,0615
5% Trim	55232350	Std Dev	2,35E+08	Range	7,67E+08	Kurtosis	4,0337
99% CI for Mean (72430852; 1,03E+08)				IQR	80634,25	S E Kurt	,1229

Tiempo de latencia global de detección de errores:

Mean	47344785	Std Err	2204610	Min	3,0000	Skewness	3,6344
Median	810,0000	Variance	3,25E+16	Max	7,67E+08	S E Skew	,0300
5% Trim	10605283	Std Dev	1,80E+08	Range	7,67E+08	Kurtosis	11,3008
99% CI for Mean (41664464; 53025107)				IQR	4968,000	S E Kurt	,0599

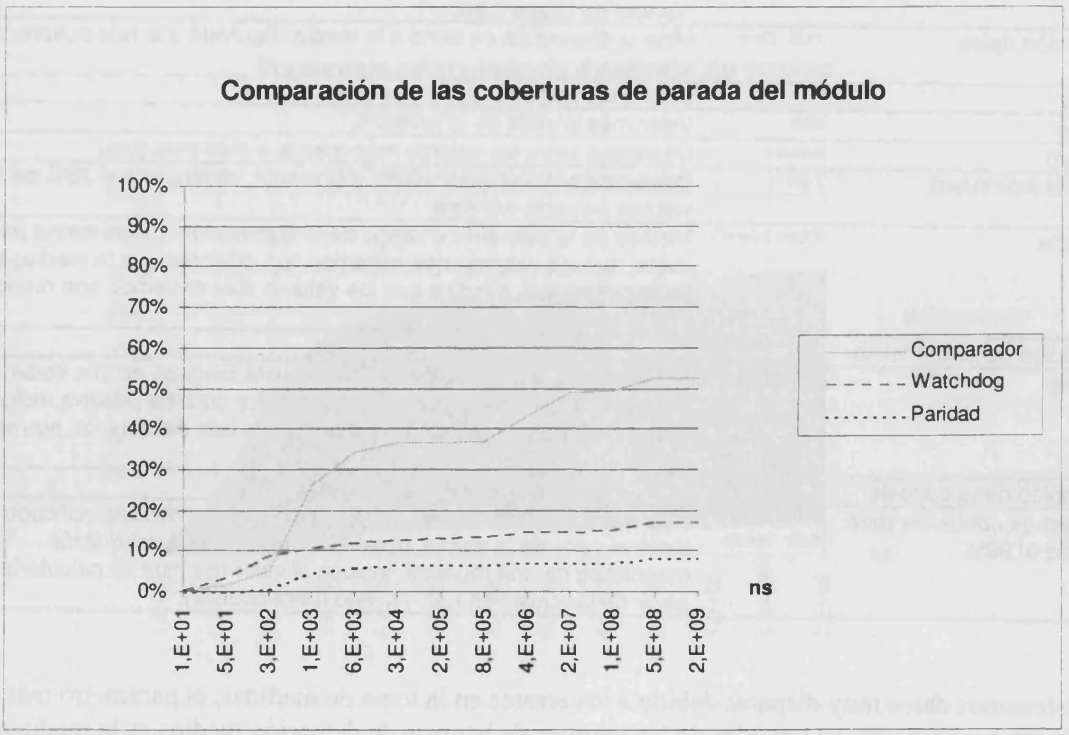


Figura 7.9: Tiempos de parada del módulo de proceso

También se ha realizado un análisis similar al anterior para poder estudiar las características de *parada después de la avería* que tiene el módulo de proceso. Es importante observar los tiempos de latencia de la parada del módulo, puesto que a partir de ellos se debe calcular la frecuencia del establecimiento de puntos de recuperación del sistema.

Se ha analizado el tiempo de parada cuando el error lo detectan los comparadores, la paridad o el temporizador de guardia. En la gráfica de la Figura 7.9 se representan superpuestas las funciones de distribución del tiempo de parada del módulo en función del tiempo. Así se puede realizar una comparación entre los tiempos de parada del módulo de proceso producidos por los tres mecanismos básicos de detección de fallos. En el eje de abcisas tenemos el tiempo transcurrido desde que se activa el error efectivo hasta que el módulo se detiene activando la señal HOLD de los procesadores. En el eje de ordenadas tenemos el porcentaje de veces que cuando se ha inyectado el error efectivo, el módulo se ha parado. La función representa el porcentaje de veces que el módulo se para en un tiempo igual o inferior al especificado en el eje de ordenadas.

Tanto para la paridad como para el temporizador de guardia, los valores de la detección del fallo coinciden con el tiempo de parada, puesto que para estos casos no existe la posibilidad de recuperación ante un fallo transitorio y por lo tanto en el mismo instante que se detecta el error se detiene al módulo. Aunque sean los mismos valores se ha considerado oportuno incluirlos de nuevo para facilitar la comparación de los estadísticos.

En la gráfica de los tiempos de parada se puede apreciar como cuando el error lo detectan los comparadores, se intenta recuperar el módulo realizando un ciclo de *BACKOFF*. Cuando se vuelve a repetir el error es cuando se para el sistema, de ahí que el porcentaje de parada es bastante bajo durante el transcurso de los primeros 300 ns después de la activación del fallo.

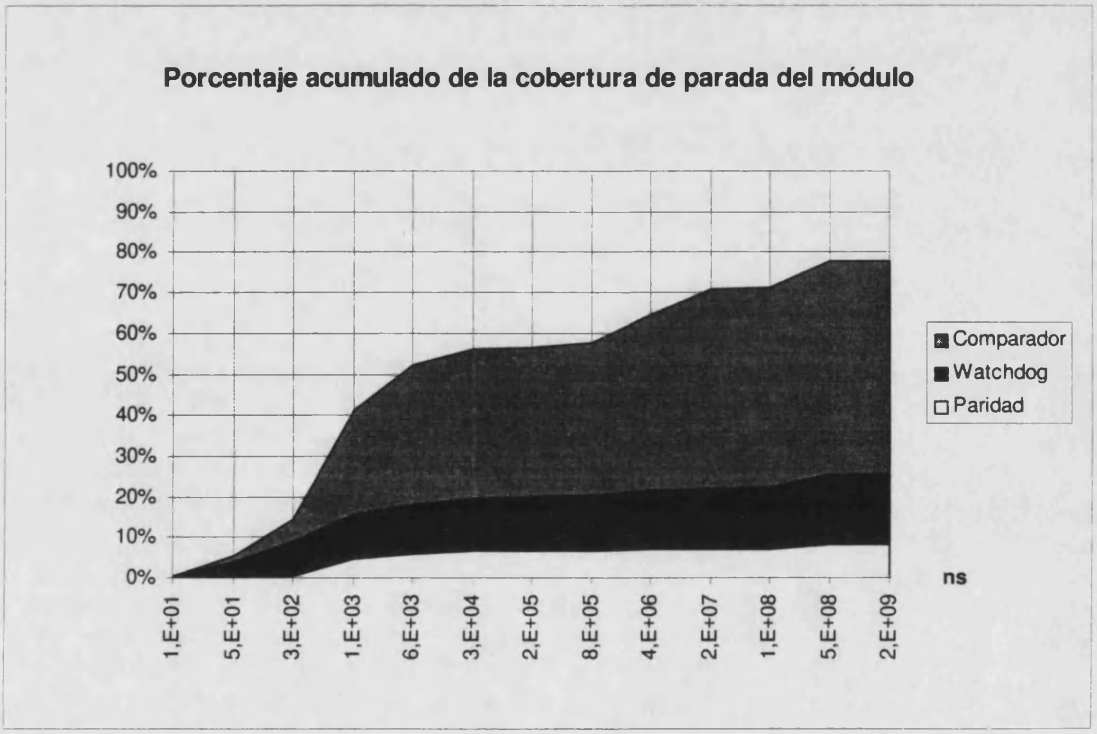


Figura 7.10: Suma de los tiempos de parada del módulo de proceso

En la gráfica de la Figura 7.10 se ha realizado una suma de las funciones de distribución del tiempo de parada para poder calcular el tiempo de parada general. En el eje de abcisas tenemos el tiempo transcurrido desde que se activa el error efectivo hasta que el módulo se detiene activando la señal HOLD de los procesadores. En el eje de ordenadas tenemos el porcentaje de veces que cuando se ha inyectado el error efectivo, el módulo se ha parado. La función representa el porcentaje de veces que el módulo se para en un tiempo igual o inferior al especificado en el eje de ordenadas, sea cual sea el mecanismo que detecta el fallo.

A continuación, como en el caso anterior, se muestran los estadísticos más comunes que definen la variable aleatoria de la latencia de parada del módulo de proceso.

Tiempo de latencia de parada por error de comparación:

Mean	54415320	Std Err	2656228	Min	80,0000	Skewness	3,5328
Median	6697,500	Variance	3,45E+16	Max	1,02E+09	S E Skew	,0350
5% Trim	17780942	Std Dev	1,86E+08	Range	1,02E+09	Kurtosis	10,6120
99% CI for Mean	(47570658; 61259981)			IQR	15215641	S E Kurt	,0700

Tiempo de latencia de parada por error de paridad:

Mean	1,22E+08	Std Err	10107643	Min	228,0000	Skewness	1,8425
Median	5242,000	Variance	7,71E+16	Max	7,66E+08	S E Skew	,0890
5% Trim	93483531	Std Dev	2,78E+08	Range	7,66E+08	Kurtosis	1,3988
99% CI for Mean	(95933285; 1,48E+08)			IQR	33010,00	S E Kurt	,1777

Tiempo de latencia de parada por error del temporizador de guardia:

Mean	87637428	Std Err	5896448	Min	23,0000	Skewness	2,4347
Median	1145,000	Variance	5,51E+16	Max	7,67E+08	S E Skew	,0615
5% Trim	55232350	Std Dev	2,35E+08	Range	7,67E+08	Kurtosis	4,0337
99% CI for Mean	(72430852; 1,03E+08)			IQR	80634,25	S E Kurt	,1229

Tiempo de latencia general de parada

Mean	68757084	Std Err	2465935	Min	23,0000	Skewness	2,9605
Median	5170,000	Variance	4,40E+16	Max	1,02E+09	S E Skew	,0288
5% Trim	33889140	Std Dev	2,10E+08	Range	1,02E+09	Kurtosis	6,8558
99% CI for Mean	(62403577; 75110590)			IQR	9983763	S E Kurt	,0576

Por último también se ha hecho un estudio del tiempo que tarda el sistema en reconfigurarse. El proceso de reconfiguración pasa por la fase de detección del fallo, aislamiento y recuperación del sistema. Para el caso de nuestro prototipo, cuando se detecta que uno de los módulos ha fallado debido a que no llegan mensajes de "estoy vivo", el sistema se reconfigura y se continúa con la aplicación del módulo con fallo. En la misma gráfica (ver Figura 7.11) se ha representado el histograma de frecuencias del tiempo de recuperación y se ha dibujado la función de las frecuencias acumuladas.

En el eje de abcisas se representa el tiempo de latencia de la recuperación del sistema, mientras que en las ordenadas tenemos el porcentaje de experiencias cuyo tiempo de latencia se encuentra en un cierto intervalo de tiempo dado. Como se puede observar, el tiempo de recuperación es prácticamente constante puesto que el envío de mensajes de "estoy vivo" se produce de forma periódica y, por lo tanto, un fallo se distribuye de forma uniforme en el intervalo de tiempo que existe entre el envío de dos mensajes consecutivos.

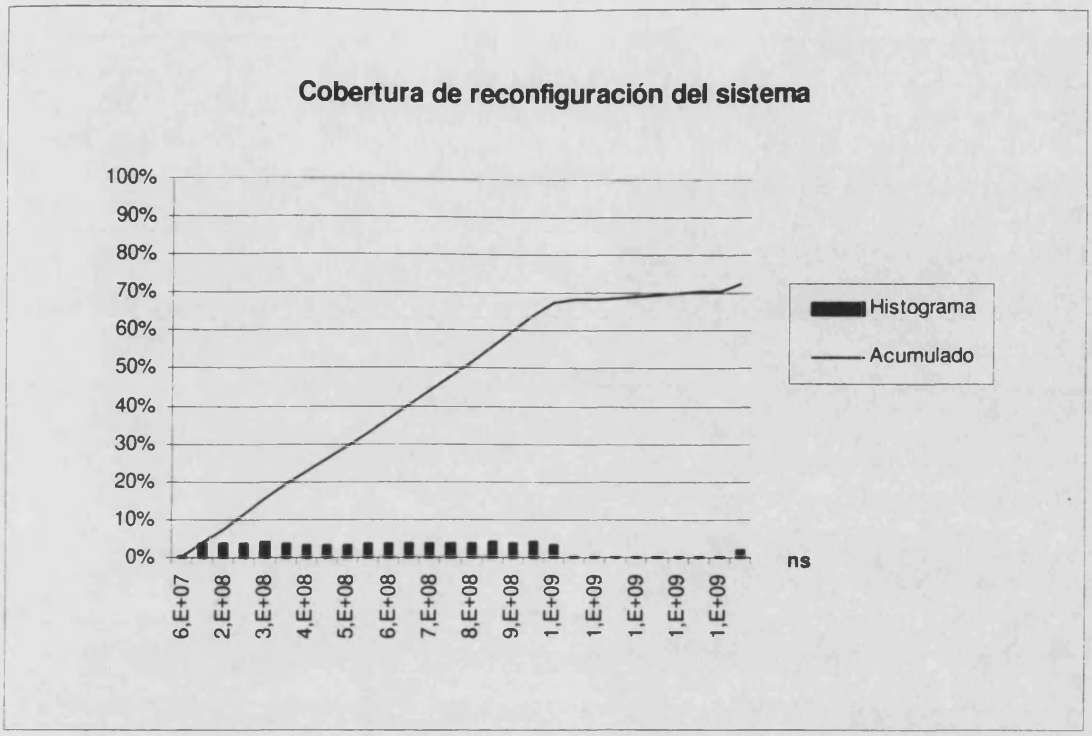


Figura 7.11: Cobertura de reconfiguración en función del tiempo

A continuación se muestran los valores numéricos para el tiempo de reconfiguración del sistema:

Tiempo de reconfiguración global del sistema

Mean	6,42E+08	Std Err	4358529	Min	69594350	Skewness	,6120
Median	6,27E+08	Variance	1,28E+17	Max	1,84E+09	S E Skew	,0298
5% Trim	6,22E+08	Std Dev	3,58E+08	Range	1,77E+09	Kurtosis	,3256
99% CI for Mean	(6,31E+08; 6,53E+08)			IQR	5,48E+08	S E Kurt	,0596

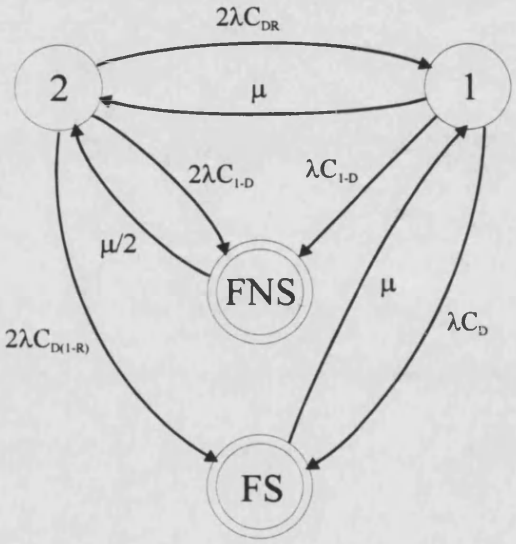
5 *Modelo macroscópico de la Arquitectura FASST*

Una vez obtenidas las coberturas de detección de fallos y de recuperación del prototipo del sistema FASST mediante el análisis de los datos de la inyección, se debe resolver el modelo macroscópico del sistema tolerante a fallos para poder realizar una estimación de los parámetros característicos de la garantía de funcionamiento del sistema. El modelo macroscópico del sistema es el representado en la. Figura 7.12.

El modelo representa un sistema tolerante a fallos con dos módulos, en donde se permite la degradación funcional del sistema y además se tienen en cuenta las violaciones de la propiedad de parada tras la avería de los módulos. En el estado *2*, las dos DPUs (módulos de proceso) están funcionando normalmente, sin errores. Cuando ocurre un fallo (con una tasa de producción de fallos λ), si se detecta el error producido y el sistema se reconfigura normalmente, el modelo evoluciona hacia el estado *1*, en donde está operativo, pero en modo degradado, puesto que ya sólo funciona un módulo. Si no se detecta el error suponemos que el sistema pasa a un estado de fallo no seguro, ya que no se pueden predecir las consecuencias del error y el modelo evoluciona hacia el estado *FNS* (Fallo no seguro). En este estado el sistema podrá estar funcionando en modo degradado o podrá estar

totalmente averiado, pero en cualquier caso los efectos no controlados del error podrían haber contaminado a los demás módulos y haber producido averías catastróficas antes de llegar a la avería. Si por el contrario se detecta el error pero fallan los mecanismos de reconfiguración, el sistema pasa al estado *FS* (Fallo Seguro), puesto que no puede seguir funcionando. El modelo contempla también la posibilidad de reparación de los módulos. En el momento en que falla un módulo se puede realizar una extracción en vivo para repararlo y volverlo a incorporar después una vez se haya solucionado el problema. Este proceso de reparación tiene una duración de $1/\mu$ horas.

Para calcular la función de fiabilidad, de seguridad o de disponibilidad del sistema, se debe resolver el modelo de forma analítica y calcular las probabilidades a lo largo del tiempo de estar en cualquier estado. Estos valores serán función de las tasas de transición del modelo.



	Significado	Estados	Valor
C_D	Cob. de detección de fallos	3, 4, 5, 6	69.7 %
C_{1-D}	Cob. de no detección de fall.	2, 7, 10	8.3 %
C_R	Cob. de recuperación	2, 4, 5, 6, 7	72.7 %
C_{DR}	Cob. recuperación segura	4, 5, 6	67.8 %
$C_{D(1-R)}$	Cob. no recuperación segura	3	1.9 %
λ	Tasa de fallos del sistema		
μ	Tasa de reparaciones		

Figura 7.12: Modelo macroscópico del sistema FASST

Ahora es cuando se encuentra la justificación del proceso de la inyección de fallos. El trabajo anterior sirve para dar valores a las coberturas de detección de errores y de recuperación y reconfiguración del sistema. Podemos agrupar los datos de la inyección de la Tabla 7.3 para calcular las probabilidades de detección y de reconfiguración del sistema. Hay que tener en cuenta para el cálculo del modelo que se trata de dos sucesos no independientes, por tanto $P(AB) = P(A)P(B|A)$. En la siguiente tabla se muestra una relación de los estados recogidos con sus características:

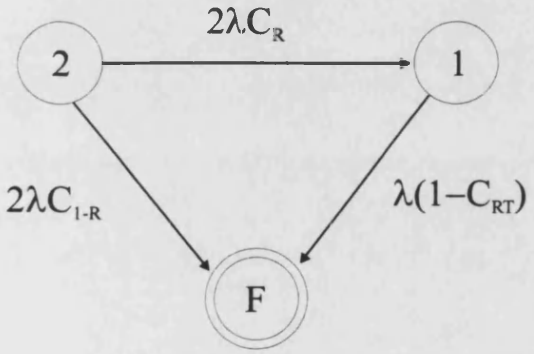
Estado	Detección	No detección	Recuperación	No Recuperación
2		X	X	
3	X			X
4	X		X	
5	X		X	
6	X		X	
7		X	X	
10		X		X

No se han considerado en la tabla los estados 1 y 8 puesto que no suponen ningún cambio de estado en el modelo, ya que se deben a la producción de fallos transitorios que el sistema recupera cuya acción no se contempla en el modelo macroscópico.

5.1 Cálculo de la función de fiabilidad

Para realizar los cálculos de los parámetros que definen la garantía de funcionamiento del sistema, se ha simplificado el modelo macroscópico dividiéndolo en un modelo de fiabilidad, y en un modelo de seguridad de funcionamiento y de disponibilidad.

El modelo de fiabilidad calcula el tiempo que tarda el sistema en averiarse desde un instante inicial donde funcionaba perfectamente. En este modelo no se contemplan las suposiciones de seguridad de funcionamiento, por lo tanto se va a considerar que si se produce una recuperación del sistema, aunque no se haya detectado ningún error, el sistema va a funcionar en modo degradado. (En el modelo macroscópico se pasaba a un estado de fallo no seguro). En la Figura 7.13 se puede ver el modelo de fiabilidad del sistema. Inicialmente el sistema esta funcionando correctamente. Cuando un módulo se avería, si el sistema se puede reconfigurar, se pasa al estado 1, en donde funciona en modo degradado. Si por el contrario no se configura, el sistema pasa al estado con fallo. Hay que hacer notar que C_R no es igual a C_{1-R} puesto que no se debe contemplar en el modelo ni los fallos transitorios ni los fallos cubiertos por la redundancia intrínseca del sistema, que hacen que eventualmente se vuelva al estado 2. Cualquier fallo que tenga lugar en el estado 1, a excepción de los que se acaban de comentar, llevarán al sistema al estado de fallo.



	Significado	Estados	Valor
C_R	Cob. de recuperación	2,4,5,6,7,9	72.7 %
$C_{(1-R)}$	% de no recuperación	3,10	5.3 %
C_{RT}	Cob. recuperación error trans.	1,8	22.0 %
λ	Tasa de fallos del sistema		260 10^{-6}

Figura 7.13: Modelo para la fiabilidad del sistema FASST

La función de fiabilidad del sistema se obtendrá calculando el tiempo que se tarda en alcanzar un estado de fallo. Para ello basta con calcular las probabilidades transitorias de estar bien en el estado 2 bien en el estado 1, teniendo en cuenta que este modelo no tiene tasas de reparación. Se ha resuelto el modelo utilizando el programa SHARPE [42] y se ha supuesto que la temperatura de funcionamiento del módulo es de aproximadamente 65 grados, con lo que la tasa de producción de fallos es de 1 fallo cada 3846 horas de funcionamiento del sistema ($\lambda = 260\ 10^{-6}$ fallos / hora). A continuación se presenta el listado del programa y los resultados obtenidos:

```
markov fasst(flam,fmu)
2 1 2*0.727*flam
2 0 2*0.053*flam
1 0 0.78*flam
1 2 fmu
end

2 1
end

bind
lam 260*10^-6
mu 0
end

cdf(fasst,2;lam,mu)
cdf(fasst,1;lam,mu)
cdf(fasst,0;lam,mu)

expr
mean(fasst,0;lam,mu)
expr prob(fasst,0;lam,mu)
end
```

Las funciones de probabilidad indican la probabilidad de estar en un estado en un tiempo determinado

$$p_2(t) = e^{-0'0004056t}$$
$$p_1(t) = 1'8641e^{-0'0002028t} - 1'8641e^{-0'0004056t}$$
$$p_F(t) = 1 - 1'8641e^{-0'0002028t} + 0'8641e^{-0'0004056t}$$

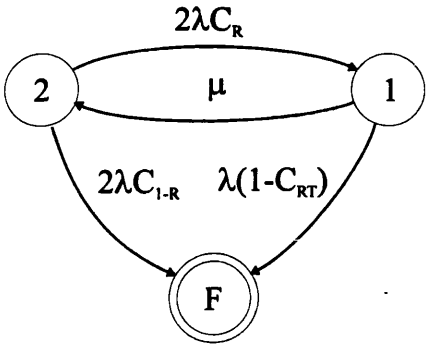
Por tanto la función de fiabilidad del sistema FASST viene dada por:

$$\boxed{R(t) = p_1(t) + p_2(t) = 1'8641e^{-0'0002028t} - 0'8641e^{-0'0004056t}}$$

Si calculamos la media de $p_F(t)$ (función de distribución del tiempo que se tarda en alcanzar el estado de fallo) podemos estimar el Tiempo Medio al Primer Fallo.

MTTF = 7061 horas (aprox. 42 semanas)

A continuación podemos calcular la fiabilidad del sistema pero considerando la posibilidad de la reparación en vivo del módulo que ha fallado. La Figura 7.14 muestra este modelo.



	Significado	Estados	Valor
C_R	Cob. de recuperación	2,4,5,6,7,9	72.7 %
$C_{(1-R)}$	% de no recuperación	3,10	5.3 %
C_{RT}	Cob. recuperación error trans.	1,8	22.0 %
λ	Tasa de fallos del sistema		$260\ 10^{-6}$
μ	Tasa de reparaciones		$13.8\ 10^{-3}$

Figura 7.14: Modelo para la fiabilidad con inserción en vivo

En este caso tendremos que considerar tasas de reparación de los módulos de proceso cuando se averían. El módulo se extrae en vivo del sistema, se repara y a continuación se realiza la inserción en vivo. La tasa de reparación es un parámetro difícil de calcular, puesto que depende de muchos factores, como por ejemplo la existencia de módulos de repuesto, el tipo de fallo producido o la habilidad del técnico que tiene que reparar la placa. Para resolver el modelo vamos a suponer una tasa de reparación de 3 días para cada módulo* (μ = 0'0138 reparaciones / hora). Resolviendo el modelo, podemos calcular de nuevo la función de fiabilidad y el factor de incremento del tiempo medio hasta la avería con respecto al sistema sin reparación:

$$p_2(t) = 0'9739e^{-0'00003217t} + 0'02635e^{0'014376t}$$
$$p_1(t) = 0'02635e^{-0'00003217t} - 0'02635e^{-0'014376t}$$
$$p_F(t) = 1 - 1'0003e^{-0'00003217t} + 0'00032e^{-0'014376t}$$

Por tanto la función de fiabilidad con reparación en vivo del sistema FASST viene dada por:

$$\boxed{R(t) = p_1(t) + p_2(t) = 1'0003e^{-0'00003217t} + 0'00032e^{-0'014376t}}$$

Factor de Incremento en el MTTF se define como el incremento que tiene el MTTF de un sistema con una tasa de reparación concreta con respecto al mismo sistema con una tasa de reparación nula:

MTIF = 31088 / 7061 = 144'02

MTI = (31088 - 7061) / 7061 = 340 %

5.2 Cálculo de la disponibilidad y de la seguridad de funcionamiento

Para el cálculo de la disponibilidad vamos a resolver el modelo macroscópico del sistema FASST (ver Figura 7.15) pero en régimen estacionario. Al igual que en la sección anterior se ha utilizado un programa desarrollado en SHARPE para calcularla. La disponibilidad indica el porcentaje de tiempo en que el sistema está operativo, es decir suministrando servicio al usuario. Indica la probabilidad que

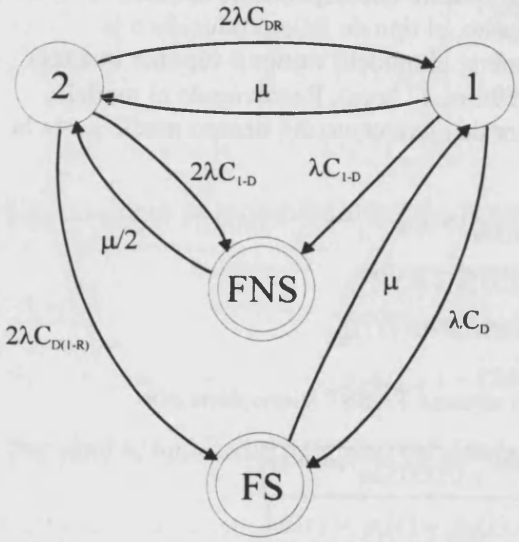
* Esta es la tasa media experimental que tardaba el autor en reparar el módulo cuando éste fallaba en las experiencias de inyección.

tiene un usuario de que cuando vaya a utilizar el sistema, éste esté disponible. Utilizando el modelo de Markov, se calcular como la probabilidad de estar bien en el estado 2 bien en el estado 1 para un tiempo infinito.

Disponibilidad = 99'483 %

Lo mismo ocurre con la seguridad del sistema. La seguridad de funcionamiento del sistema indica la probabilidad de que el sistema esté funcionando correctamente o que deje de funcionar de forma que no entorpezca la operación de otros sistemas ni comprometa la seguridad de ninguna persona. En nuestro prototipo se considera que el sistema no es seguro si ha habido algún fallo que no se ha detectado. La seguridad la obtendremos como la probabilidad de estar en régimen estacionario en el estado 2, ó el 1, ó en el estado de fallo seguro

Seguridad = 99'693 %



	Significado	Estados	Valor
C_D	Cob. de detección de fallos	3, 4, 5, 6	69.7 %
C_{I-D}	Cob. de no detección de fall.	2, 7, 10	8.3 %
C_R	Cob. de recuperación	2, 4, 5, 6, 7	72.7 %
C_{DR}	Cob. recuperación segura	4, 5, 6	67.8 %
$C_{D(I-R)}$	Cob. no recuperación segura	3	1.9 %
λ	Tasa de fallos del sistema		$260\ 10^{-6}$
μ	Tasa de reparaciones		$13.8\ 10^{-3}$

Figura 7.15: Modelo para la seguridad y la fiabilidad

```
markov fasst(flam,fmu)
3 2 2*0.678*flam
3 1 2*0.083*flam
3 0 2*0.019*flam
2 1 0.083*flam
2 0 0.697*flam
1 2 fmu
2 3 fmu
0 3 fmu/2
end

bind lam 260*10^-6
bind mu 0.0138

func avai(r1,r2)
prob(fasst,3;r1,r2) +
prob(fasst,2;r1,r2)
func seg(r1,r2) 1 -
prob(fasst,1;r1,r2)
```

```
var disp avai(lam,mu)
var segur seg(lam,mu)

expr disp
expr segur

end
```

6 Conclusiones

En este capítulo se han mostrado los resultados experimentales de la inyección de fallos física sobre el sistema FASST. Con estos datos experimentales se ha construido un nuevo modelo microscópico para la inyección de fallos y se han calculado las coberturas de detección de errores y de recuperación del sistema multiprocesador.

Como resultado se muestran los valores para la garantía de funcionamiento, en función de los tiempos de latencia y de las coberturas. Se ha calculado el tiempo medio hasta la avería y el porcentaje de tiempo en que el sistema estará disponible y funcionando en un estado seguro.

Capítulo 8

Conclusiones

Durante el tiempo que duró la investigación se ha realizado una evaluación de la garantía de funcionamiento de un prototipo del sistema FASST, diseñado en una primera fase durante el proyecto ESPRIT P5212 FASST, y rediseñado en esta tesis durante el periodo posterior a la finalización del proyecto para que sirviera como plataforma donde realizar la inyección. Los parámetros característicos del sistema se han calculado mediante el método de la inyección física de fallos, obteniendo valores reales para el caso de las coberturas de detección de fallos, la cobertura de reconfiguración del sistema y los tiempos de latencia en la detección del error y de reconfiguración del sistema.

La mejor manera para realizar la validación de un sistema es utilizarlo normalmente, con una carga real y dejarlo evolucionar de forma que el proceso de producción de fallos sea totalmente natural. Cada avería del sistema se registra y después se analiza como se han comportado los mecanismos de detección de fallos y de recuperación del sistema. Después de varios años se tendrán un conjunto de medidas totalmente válidas sobre la tasa de averías que sufre el sistema, la tasa de producción de fallos y sobre las coberturas de detección, reconfiguración y recuperación. Sin embargo la gran desventaja que tiene el método es que es sumamente largo en el tiempo, puesto que los fallos tienen lugar muy de cuando en cuando. Además si analizo los eventos del sistema en su vida operacional, quedo restringido en el análisis a los fallos detectados, que son los que disparan el proceso de toma de datos. En la presente memoria de investigación se ha presentado un método, basado en la producción artificial de fallos que acelera extremadamente el proceso de validación de la garantía de funcionamiento de los sistemas tolerantes a fallos.

Aplicando la inyección de fallos sobre el sistema prototipo FASST se han obtenido medidas, también reales sobre las coberturas de detección de fallos y de recuperación, incluyendo además el cálculo de los tiempos de latencia de detección y de recuperación. Si este trabajo no hubiera tenido lugar, por un lado no se podría saber si el sistema es apropiado para aplicaciones seguras y por otro lado tampoco se podría aplicar en sistemas reactivos* con una certeza absoluta de que se van a respetar los tiempos máximos de ejecución de las tareas de recuperación.

Se han desarrollado modelos macroscópicos analíticos de fiabilidad, seguridad y disponibilidad del sistema FASST. Estos modelos son únicos para este sistema puesto que recogen sus características

---

* Los sistemas reactivos son los sistemas tolerantes a fallos que se utilizan en aplicaciones en tiempo real.

topológicas y son imprescindibles para el cálculo de los parámetros característicos que ponen de manifiesto la garantía de funcionamiento que ofrece. Se han resuelto los modelos, después de parametrizarlos mediante los datos extraídos de las experiencias de inyección, y se ha calculado la función de fiabilidad y se han obtenido valores para el tiempo medio al primer fallo, la seguridad y disponibilidad del sistema, como se recogen en la siguiente tabla:

Parámetros del sistema FASST	
MTTF	42 semanas
Disponibilidad	99'483 %
Seguridad	99'693 %

Para el cálculo de los dos últimos parámetros, también se ha diseñado un nuevo modelo microscópico que recoge las coberturas de detección y de degradación del sistema con respecto al proceso de producción de fallos. Estos modelos tienen la ventaja sobre los vistos en la literatura que son adecuados para el proceso de la inyección de fallos y que además, permiten calcular las violaciones de la característica de parada ante fallo que se producen en los módulos de proceso *fail-stop*. Este modelo es de vital importancia para el cálculo de la seguridad del sistema, totalmente necesario cuando se quiere utilizar el sistema en aplicaciones donde existe riesgo de pérdida de vidas humanas.

Es importante comentar que como el análisis del sistema se ha realizado sobre un prototipo, no se disponía de una tasa de producción de fallos real. Todos los cálculos se han realizado suponiendo que la producción de fallos en el sistema a lo largo del tiempo sigue una distribución exponencial. La elección se ha realizado puesto que los componentes electrónicos se encuentran en el periodo de vida útil, donde se ha demostrado que su tasa de fallos es constante (el cálculo de la misma se puede ver en el Capítulo 5 donde se presenta una tabla de λ en función de la temperatura). Además, debido a que la carga del sistema para las experiencias consiste en una sencilla aplicación, no se ha considerado la posibilidad de que se produzcan fallos software, en donde los fallos ya siguen otro tipo de distribución. Esta restricción no quita validez a los resultados obtenidos, sin embargo hay que constatar que los mismos dependen fuertemente de la temperatura de funcionamiento del sistema.

Es discutible pensar que todo el trabajo no tiene validez puesto que los resultados dependen de un parámetro, la tasa de fallos, que se ha calculado *"a la ligera"* y que encima depende fuertemente de la temperatura. Aunque esto no es así, puesto que para el cálculo de λ se ha utilizado un método científico basado en el estándar MIL-HDBK-217F, no sólamente se suministran los valores que definen la garantía de funcionamiento del sistema, sino que también se incluyen los modelos de fiabilidad, seguridad y disponibilidad del sistema, de forma que perfectamente se puede comparar el sistema FASST con otros sistemas, resolviendo sus modelos considerando la misma tasa de producción de fallos en los dos.

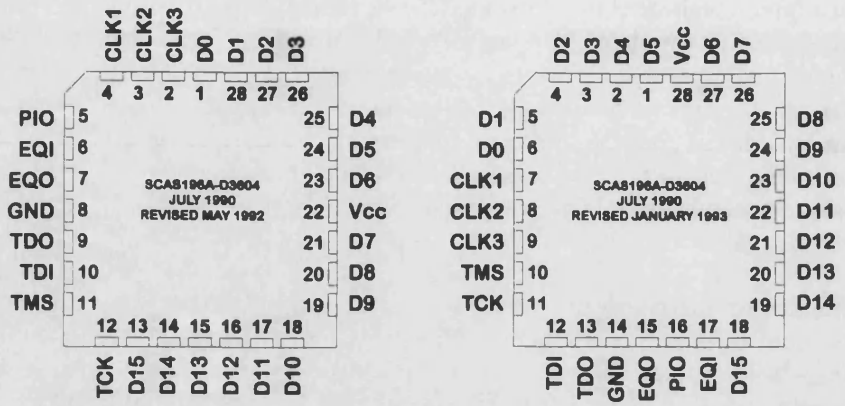
El inyector de fallos se ha puesto a punto para la realización de experiencias de inyección a alta frecuencia y se ha desarrollado un sistema híbrido de adquisición de datos basado en tres módulos sincronizados entre sí: Un ordenador personal que gobierna el proceso de la inyección, un inyector de fallos de alta frecuencia y un analizador lógico encargado de recoger las medidas oportunas.

Quisiera hacer notar expresamente las dificultades que han ido surgiendo a la hora de implementar los módulos de proceso, para que se tengan en cuenta en futuros trabajos de investigación o en proyectos ya de carácter comercial. La primera de ellas, quizás es la más notable y es en donde hay que tener especial cuidado. En todo diseño se deben elegir componentes de fácil adquisición y con un periodo de producción lo suficientemente largo como para poder construir nuevos prototipos. En el caso del sistema FASST se tuvieron grandes problemas a la hora de conseguir los integrados que implementan los transceivers BTL de Futurebus+ e incluso los integrados para la memoria cache. Además recuérdese que se tuvieron que sustituir los procesadores MIPS R4000 por los Intel DX2, debido a la

falta de información para realizar el diseño y a la falta de herramientas de emulación y compilación. Pero esto aunque supone un problema, con una elección cuidadosa se puede subsanar. Sin embargo se debe tener en cuenta que es vital reducir el *tiempo al mercado* del prototipo, o del sistema real. En el caso de FASST, una vez implementado el diseño, los integrados de Futurebus+ están obsoletos e incluso el mismo estándar se está revisando, por lo que ya se tendrán graves problemas si se quiere implementar una nueva placa. En consecuencia hay que elegir componentes nuevos y estar bien relacionado para que te suministren la mayor información preliminar posible. Un estudio previo del mercado nunca viene mal.

El segundo problema que alteró gravemente el flujo normal del trabajo fue el hecho de que el fabricante de los circuitos impresos nos suministrara placas defectuosas. Las placas tenían errores, sobre todo de líneas abiertas y de cortocircuitos. Antes de soldar un integrado es necesaria la realización de la comprobación de las líneas, puesto que si se utiliza montaje superficial, aunque se puede soldar cualquier integrado si se dispone de estaño de calidad y un poco de práctica, el desoldarlos es prácticamente imposible cuando tienen un alto número de patas. Si se quiere utilizar otra placa libre de errores hay que comprar también nuevos componentes, que no son baratos. En la placa del prototipo se han llegado a cortar líneas del segundo nivel de profundidad para poder corregir los errores. Es recomendable también al realizar el rutado del diseño el uso de tantas vías como sea posible, puesto que sirven tanto para puntos de test como para poder añadir nuevas pistas o reparar cortes con hilo de grapinar. A la hora de verificar el circuito mediante un analizador lógico, el único método que hay para acceder a las señales de los integrados de montaje superficial es pinchar las puntas de prueba en las vías, aunque también se pueden comprar zócalos especiales para montaje superficial, pero son extremadamente caros. Si se utilizan zócalos para los dispositivos reprogramables, si estos son de montaje superficial, hay que tener en cuenta que es bastante probable que se produzcan errores por malas conexiones del circuito con el integrado, por lo que se recomienda el uso de limpiadores para los contactos. Si el problema persiste, de tanto apretar el integrado contra el zócalo, es normal que se desuelde alguna de las patas del zócalo con el PCB, sobre todo las de las esquinas.

El tercer problema es poco usual, pero por la gravedad del mismo creo que merece la pena comentar. En el diseño se utilizaron integrados de Texas Instruments de la familia 74ACT8994, que son monitores de bus digital. Texas Instruments, cambió completamente la disposición de las patas (*pinout*) en el circuito en una revisión que hizo en un periodo de 8 meses. Los resultados fueron catastróficos, ya que aparte de no poder utilizar los integrados (menos mal que no eran vitales, puesto que sólo se trataba de un monitor) se produjeron cortocircuitos al soldar los integrados que fundieron alguna línea. Por descontado Texas Instruments únicamente vendía los integrados correspondientes a la nueva revisión, sin posibilidad de adquirir los antiguos. Se contactó con la fábrica en Estados Unidos y la única respuesta que se obtuvo fue que lo sentían mucho y que nos enviaban 5 integrados más por si queríamos reutilizar los que se habían quemado en otra cosa. A continuación se presentan un dibujo con los dos *pinouts*.



Durante el periodo que ha transcurrido en el desarrollo de la investigación, se han abarcado en mayor o menor detalle un número amplio de temas de investigación, todos relacionadas con el diseño de sistemas multiprocesador tolerantes a fallos, la validación de sistemas y la inyección de fallos. Numerosas ideas han surgido, sin embargo no se han podido indagar en ellas en profundidad, algunas por la falta de tiempo, otras porque se alejaban un poco del tema de la investigación y otras por la falta de medios materiales para llevarlas a cabo. Sin embargo, a continuación quiero resumir las líneas de investigación abiertas que más interés me han suscitado para una posible continuación del trabajo y para que sirvan como base de futuras tesis doctorales:

- El modulo de proceso del sistema FASST incorpora un sistema *on-board* de monitorización del bus. Este monitor se programa con instrucciones de *boundary scan* mediante el uso del JTAG, que es un bus con 4 líneas que se puede conectar por ejemplo al bus paralelo de un ordenador personal. Durante la realización de la memoria, uno de los trabajos realizados fue la implementación de un software compatible con el estándar y que servía para programar los integrados. Debido al error de Texas Instruments comentado anteriormente no se ha podido utilizar. Como continuación sugiero la implementación de un inyector de bus digital hardware que aproveche y mejore el software ya realizado.

- Estudio de los tiempos de latencia en la detección de fallos teniendo en cuenta la jerarquía de memoria del módulo de proceso. Uno de los primeros trabajos que se hizo, fue poner a punto un simulador de caches y modificarlo para que admitiera varios niveles. El objetivo fue, teniendo en cuenta cargas reales de programas comerciales, observar el tiempo que tardaban los datos en salir fuera del procesador. El trabajo se abandonó porque empezaba a desviarse un poco de los principales objetivos de la investigación. Sin embargo se puede retomar y tratar de estudiar las diferencias existentes en las latencias de detección de fallos activando y desactivando las caches internas del procesador. En este trabajo también se considera interesante estudiar cuál es la cobertura de detección de fallos del procesador, teniendo en cuenta los mecanismos de protección que incorpora.

- Otra línea de investigación que queda abierta en el presente trabajo es el diseño de sistemas operativos que, basándose en las técnicas incluidas en el soporte físico, ofrezcan a los procesos una interfaz libre de averías. En efecto, aunque las técnicas incluidas en el soporte físico de la plataforma FASST permiten el establecimiento de puntos de recuperación y la vuelta atrás del sistema, todavía es necesario que el soporte lógico realice una serie de tareas relacionadas con la tolerancia a fallos. Estas tareas incluyen aspectos como la reconfiguración del sistema tras la vuelta atrás, la integración de las interrupciones y de la entrada salida con el establecimiento de puntos de recuperación y la eventual vuelta atrás, o la gestión de la memoria virtual utilizando la memoria estable como memoria principal.

- Estudio de la cobertura de detección de fallos del bus Futurebus+. En el presente trabajo se ha calculado la garantía de funcionamiento de los módulos de proceso del sistema FASST y aunque sí que se han comentado las características de tolerancia a fallos que tiene Futurebus+, éstas no se han analizado. El nuevo trabajo consistiría en modificar el inyector para que pudiera trabajar con niveles BTL (que son los que utiliza Futurebus+), e inyectar fallos en las líneas del bus para estudiar su comportamiento.

- El prototipo actual carece de dispositivos de E/S, salvo los puertos serie que incorpora cada placa. Los dispositivos de E/S están implementados en una placa de VME que se comunica con Futurebus+ a través de otra placa que hace de puente. Cada placa tiene su procesador y un espacio de memoria local, por lo tanto es necesario desarrollar todo el software de los programas de canal necesarios para realizar las operaciones de E/S. Además se deberán desarrollar los drivers para el conjunto de dispositivos que admite la placa del bus VME.

- Con los nuevos procesadores, es fácil diseñar sistemas duales con un mínimo de lógica, puesto que incluyen señales de sincronización que facilitan la realización de la votación. Se propone la realización de nuevos módulos de proceso, con diferentes arquitecturas y que sean compatibles con Futurebus+ para que puedan cooperar con los módulos ya existentes. La experiencia recogida en el diseño del prototipo facilitará enormemente el trabajo, por lo que no se supone que sea una tarea demasiado duradera en el tiempo, siendo apropiada para la realización de algún proyecto de sistemas informáticos dentro de la Ingeniería Informática de la Universitat de València.

- Después de la realización de la inyección, se ha recogido una muestra de más de 20000 entradas sobre el comportamiento del sistema derivado de la producción de fallos de forma artificial. Con esta base de datos se pueden realizar infinidad de medidas, proponiéndose desde aquí el ajuste de los tiempos de latencia a una función de distribución conocida y su posterior estudio. Además se podrá estudiar la correlación existente entre la probabilidad de detección de fallos permanentes, transitorios o intermitentes, así como la influencia del tiempo de duración del fallo en su detección.

Anexo I

En este anexo se enumeran el conjunto de fallos y modificaciones que se han realizado a los módulos de proceso del sistema FASST para ponerlos en marcha. No están todos, sólo los más representativos:

1. Falta de inclusión de los extractores físicos en las placas. Se han tenido que taladrar las placas por los extremos y se han insertado cables trenzados metálicos para poder realizar la extracción del rack del sistema.

2. Los puentes de resistencias tenían mal la geometría. Ha sido necesaria la inclusión de nuevos puentes de resistencias de otro fabricante con otra geometría, e inutilizar varias patas de los mismos.

3. La frecuencia del reloj que llega a las EPLDS no es la misma que la que llega al procesador. Se ha puenteado la señal de reloj.

4. La EPM7032 encargada de la votación final tenía mal toda la geometría. Fue necesaria la recompilación al confundir el encapsulado LC con el QC.

5. Debido a la no disponibilidad de memoria cache se ha tenido que reprogramar la PAL que controla el espacio de direcciones cacheable por el HT44 para que no considere ninguna dirección.

6. Uno de los CY7B992 que proporcionan las señales de reloj estaba mal conectado y generaba una frecuencia distinta de los demás.

7. Con respecto al FB+ chipset, funciona mal la generación de la señal de reset puesto que al activarse la línea RE* del FB+ produce la activación de sysreset* del chipset que a su vez va al integrado generador de la señal de reset local y resetea todo el modulo. Lo que ocurre es que se entra en un bucle infinito, produciéndose inicializaciones continuamente.

   Para solucionarlo lo que se ha hecho es conectar la señal de rst* del chipset no a reset1*, sino a la señal de reset externo que nos viene del pulsador. (Hay que tener en cuenta que la entrada de reset de uno de los votadores de direcciones también se ha modificado aunque habrá que devolverla a su valor original).

8. Los pines del conector externo RS232 de la UART están invertidos. Hay que intercambiar respectivamente 1-5, 4-2, 6-9 y 7-8.

9. Los drivers del futurebus no tienen conectada a masa el pin de BVGND , lo que provocaba que no se pudieran realizar lecturas de datos ni de direcciones, sólo escrituras. Simplemente se ha conectado esta línea a tierra.

10. La FB_FPLA no pone en triestado las señales del HIF cuando el chipset de TI es maestro del HB. Se ha modificado esta EPLD para que así sea. Además se ha cambiado el tratamiento a las señales de BR*, BG* y BGACK*. Ahora funcionan de forma asíncrona.

11. Se ha añadido una resistencia de pull-up de 5k6 a las señales BSTAT[1..0], ya que son de entrada cuando el chipset de TI es maestro del HB. Su estado indefinido provocaba que se ejecutaran múltiples ciclos de backoff, dando un error de timeout en la transacción.

12. Se ha cambiado la resistencia de pull-up de las líneas DSACK[1..0] y BSTRDY* de 4k7 a 1k2. De esta forma la señal no se retrasa tanto.

13. Se ha reparado la conexión del cristal, que estaba soldada a dos líneas de direcciones en vez de a las vías correspondientes.

14. Las señales de los generadores de reloj de Cypress 4F1,4F0 han sido cambiadas puesto que no funcionaban correctamente. Ahora realmente se obtiene los 40 y 33 Mhz necesarios para los integrados de Futurebus+ y los procesadores.

15. La señal VCLK ahora si que se divide por dos y por 4. Antes solo se dividía por dos debido a un error en una de las entradas de un biestable que hacía de divisor de frecuencia. La entrada /K del primer biestable se ha puesto a cero en vez de a Vcc.

16. El tratamiento de la señal de reloj de la uart se ha variado, puesto que el divisor de frecuencia LS90 no aguanta los 40 MHz de frecuencia que tiene la señal de entrada. Lo que se ha hecho ha sido en vez de dividir 40/5 hemos dividido VCLK/2 entre 2. De esta manera también conseguimos los 8 MHz.

17. Infinitas reprogramaciones de las EPLDs del CSR y de FB+.

18. Cambio del flanco de lectura de los accesos al CSR. Si funciona el sistema a 33Mhz, se capturaban demasiado pronto los datos y al pasar de 8 a 32 bits, se perdían los primeros 8 bits.

19. Nuevo tratamiento de la generación de la señal de backoff para que una condición de error la mantenga activa durante 32 ciclos de reloj.

20. Una de las señales de interrupción ha tenido que invertirse puesto que se generaba con el valor contrario.

21. La frecuencia de ocurrencia de las operaciones de lock no es lo suficientemente rápida como para reinicializar los temporizadores del watchdog. Se ha modificado el circuito para eliminar uno de los contadores. De esta manera se puede programar la reinicialización cada 2,4,8 ó 16 operaciones de LOCK. (Antes como mínimo eran 32).

22. Se ha utilizado la entrada de la pal ERRSEL2 como entrada para habilitar el watchdog por software. Hasta que no estén habilitadas las interrupciones no es deseable tener habilitado

el temporizador, puesto que cuesta insertar instrucciones LOCK de forma artificial y no sería transparente para el usuario.

23. El registro de control donde se guardan los valores de configuración de las placas no se puede leer. Este problema se ha solventado por software utilizando una variable que guarda su contenido. Cada vez que se escribe en él hay que actualizar la variable.

24. Se han juntado los relojes para evitar problemas de clock skew en los procesadores. Antes cada microprocesador funcionaba con su propio reloj.

25. Modificación de las EPLD's de datos para que no se voten los datos durante la primera activación de #RDY en el ciclo de reconocimiento de interrupción. Se ha incluido AD2 y M/IO en vez de las entradas TDI y TDO que no se utilizaban.

26. Modificación de la EPLD votador final. Se han juntado las líneas de NMI de cada procesador en una sola (pin 31) y ésta se ha sincronizado con el flanco de bajada del reloj. El tratamiento de la NMI se ha modificado para que únicamente se genere una vez en la detección de error de comparación. La línea que se libera se utiliza para activar la señal de #BACKOFF de los procesadores durante 32 ciclos de reloj. Con esto se pretende que el sistema sea capaz de soportar un fallo transitorio que dure menos que ese tiempo. La interrupción del 8259 también ha tenido que sincronizarse con el flanco de bajada de la señal de reloj. Si no se hace así, en el momento que no se cumplía el tiempo de setup o de hold en alguno de los procesadores, o bien no se reconocía la interrupción con lo que se desincronizaban los P o bien se retrasaba una instrucción. Esto ocurría de forma aleatoria tanto en un procesador como en el otro, y a pesar de que utilizaban la misma señal de reloj y compartían la línea de interrupción.

27. El tratamiento de los errores es el siguiente:

    Cuando llegan 2 nmi antes de que se produzca una interrupción, se deja al módulo en estado de hold. Se puede permitir el acceso externo a sus recursos o no. Esta característica se puede programar a través del registro mediante la entrada ERRSEL1.

    Se deja al módulo aislado cuando se produce un error de paridad o un error de temporización.

Anexo II

1 Introducción

El IEEE 896 Futurebus+ Profile B es un subconjunto del protocolo completo de FB+. Está pensado para constituir un bus de E/S de alto rendimiento de propósito general. Las características lógicas que lo definen son:

- Arbitraje centralizado
- Ancho de direcciones de 32/64 bits
- Ancho de datos de 32/64 bits
- Conjunto de transacciones simple
- Conjunto de CSRs del FB+
- Esquema de interrupciones simple
- Arquitectura DMA de alto rendimiento

Las características físicas se basan en los requerimientos de la lógica BTL, especificada en el IEEE 1194.1 y en los requerimientos mecánicos del estándar IEEE 1301 e IEEE 1303.1. El entorno para la clase de Profile B está especificado en el estándar IEEE P1156.1, Clase 4.

En la Tabla 8.1 se presenta una lista de los estándares que están relacionados con el Futurebus+:

IEEE 896.1-1991	Futurebus+ Logical Protocol Specifications
IEEE 896.1a-1993	Errata, Correction and Clarifications of IEEE 896.1-1991 Futurebus+ Logical Layer
IEEE 896.2-1991	Backplane Bus Specification for Multiprocesor Architectures: Futurebus+
IEEE 896.2a-1994	Futurebus+ Physical Layer and Profile Specification: Errata, Correction and Clarifications
IEEE 896.3-1993	Recomended Practice for Futurebus+
IEEE 896.3a	Recomended Practices for the Electrical Environtment Withing Backplane Transceiver Logic (BTL) Futurebus+ Systems
IEEE 896.4-1993	Conformance Test Requirements for Futurebus+
IEEE 896.4a	Suplement to IEEE Standard for Conformance Test Requirements for

	Futurebus+ : Errata, Corrections and Clarifications
IEEE 896.5-1993	Futurebus+ Profile M (Military)
IEEE 896.5a-1994	Futurebus+ Profile M (Military): Errata, Corrections and Clarifications
IEEE 896.6	Futurebus+ Telecommunications Systems, Profile T (Telecommunications)
IEEE 896.7	Interconnect Between Futurebus+ Systems
IEEE 896.9-1994	Fault Tolerant Extensions to the Futurebus+ Architecture
IEEE 896.10	Futurebus+ Spaceborne Systems, Profile S
IEEE 896.11	IEEE 1355 Links on Futurebus+ Backplane Connector
IEEE 896.12	Fault Tolerant Classifications of Computer Based Systems
IEEE 1014.1-1994	Futurebus+/VME64 Bridge
IEEE P1156.1	Environtmental Specifications for Computer Modules
IEEE 1194.1	Electrical Characteristics of Backplane Transceiver Logic Interface Circuits
IEEE 1212	Control and Status Register Architecture
IEEE P1212.1	DMA Framework Architecture
IEEE 1301.1	Detailed Standard for a Metric Equipment Practice for Microcomputers Using 2 mm Conectors and Convection Cooling

Tabla 8.1: Estándares relacionados con FB+

A lo largo de este capítulo se va a revisar el estándar puestos que es interesante para entender como funcionan los mecanismos de tolerancia a fallos en el sistema multiprocesador.

2 Arquitectura

En la Figura 8.1 se puede observar un diagrama de bloques de un sistema típico conforme con las normas del Profile B.

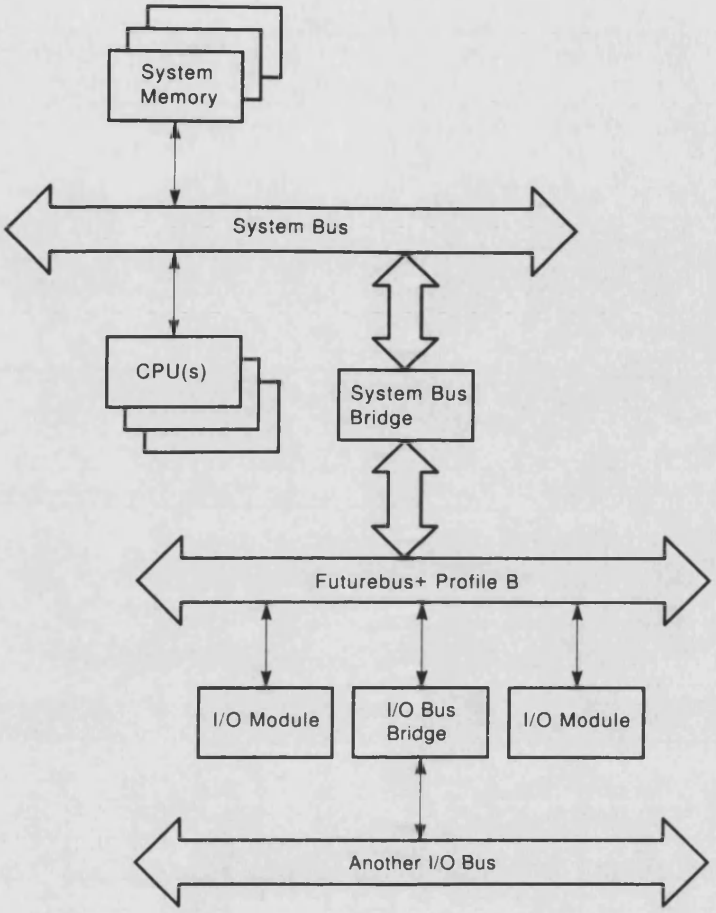


Figura 8.1: Diagrama de bloques del sistema

Un módulo es una entidad que consiste en uno o más circuitos impresos conectados al FB+ mediante un único conjunto de transceivers. Un módulo puede tener cero, uno o dos nodos. Si no tiene ningún nodo indica que es un analizador de bus, que se limita a observar las señales que pasan por el bus. Este módulo no puede contener CSRs.

Un nodo es una entidad que está controlada por un único conjunto de CSRs. Contiene un conjunto de direcciones de control y registros de estado que inicialmente están definidos en una ROM de 4k de inicialización de nodo. En un nodo existe un bus de CSR que se utiliza para interconectar dispositivos con el objetivo de reducir el tráfico del interface del Host Bus. Este último interconecta al bus del CSR, la memoria, el procesador y la cache.

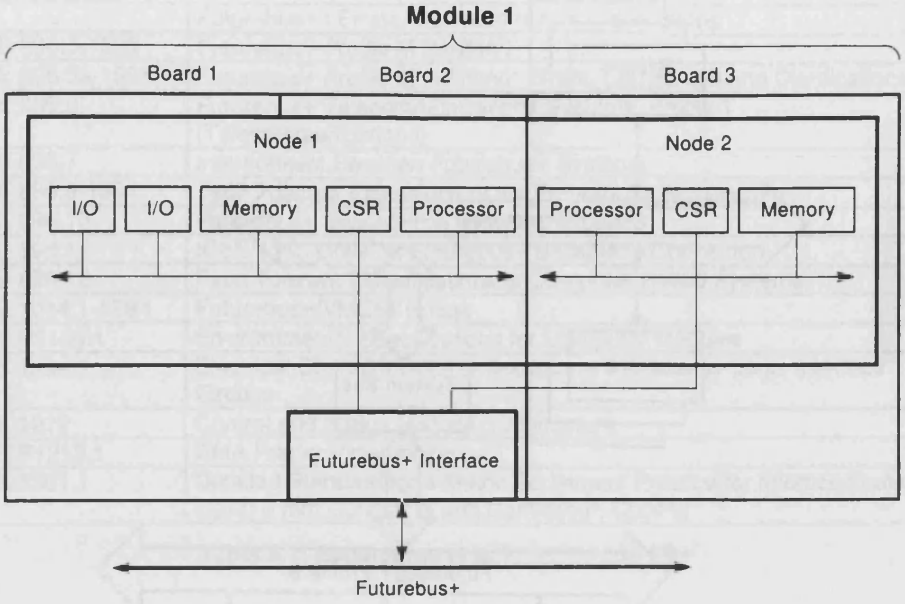


Figura 8.2: Estructura de un módulo de Futurebus+

En la Figura 8.2 podemos ver un ejemplo de un módulo de FB+ compuesto por dos nodos y dividido en tres circuitos impresos.

2.1 Mapa de direcciones del Futurebus+

El mapa de memoria del FB+ se divide en dos partes, una reservada para los CSRs de todos los módulos y otra en donde se direcciona la memoria y las unidades extendidas. Es necesario por tanto programar en cada módulo las direccionesa de estos espacios a través de unos registros que residen en el core del CSR. De esta forma el chipset cuando recibe una dirección del HB puede determinar si la transacción es local o si se debe de realizar a través del FB+. Si por el contrario la dirección viene a través del FB+, podrá actuar como esclavo en el caso de que esta dirección se corresponda con una zona de memoria local.

Este bus soporta direcciones de 32 bits y opcionalmente direcciones de 64 bits. Usando direcciones de 32 bits se define un espacio de direccionamiento de 4096 Mbytes que se divide en dos regiones principales:

- 256 Mbytes de espacio para el CSR
- 3840 Mbytes de espacio para la memoria del sistema

Para que puedan convivir módulos que utilicen direcciones de 32 bits y direcciones de 64 bits, sólo se podrán utilizar los 64 bits para las transacciones que direccionen una posición de memoria comprendida entre 0xF000 0000 y 0xFFFF FFFF F000 0000. Las demás direcciones tendrán que ser reconvertidas a direcciones de 32 bits como muestra la Tabla 8.2.

CM7*	De	A	Descripción
0	0000 0000	EFFF FFFF	Espacio memoria A32
0	F000 0000	FFFF FFFF	Espacio del CSR del nodo
1	0000 0000 0000 0000	0000 0000 EFFF FFFF	PROHIBIDO
1	0000 0000 F000 0000	FFFF FFFF EFFF FFFF	Espacio memoria A64
1	FFFF FFFF F000 000	FFFF FFFF FFFF FFFF	PROHIBIDO

Tabla 8.2: Direccionamiento híbrido

3 *Registros de comando y de estado (CSR)*

El espacio del CSR definido por el FB+ es un área de estado y de control localizada en una posición fija que para cada módulo suministra información de configuración, identificación, manejo de interrupciones, acceso de E/S, y métodos de test para cada módulo. Los CSRs pueden ser programados por medio del FB+ teniendo en cuenta que todos los datos que se pasan son de 32 bits.

Todos los CSRs del FB+ se direccionan en una zona reservada de 256 MB en el espacio A32 (direcciones de 32 bits). Esta sección está a su vez subdividida como muestran la Figura 8.3.

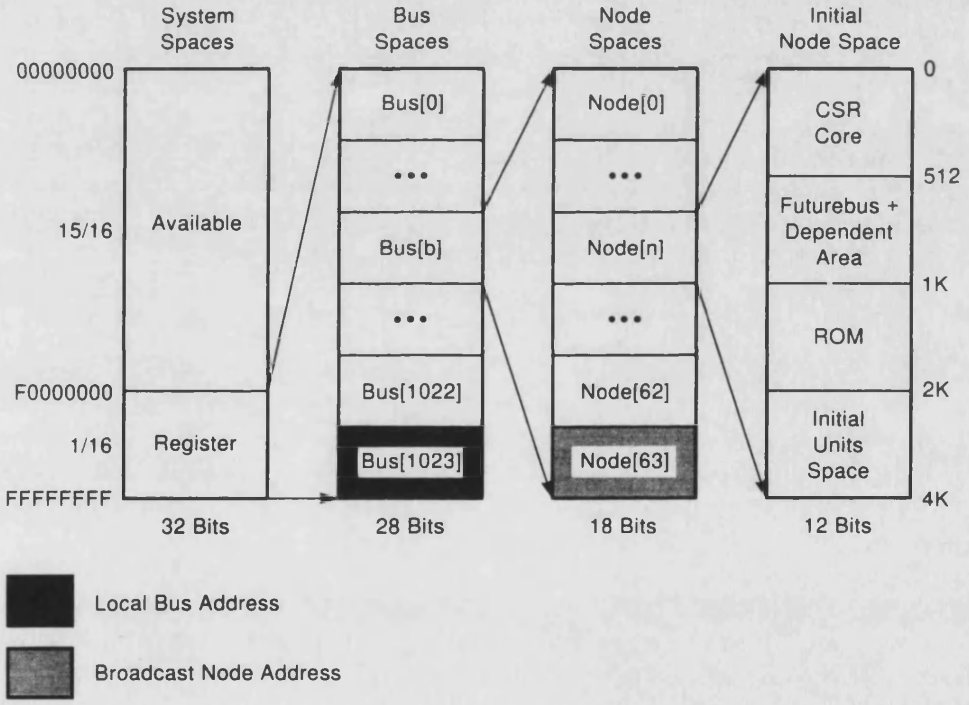


Figura 8.3: Mapa de direcciones de FB+

El espacio que ocupan los CSRs en el FB+ se divide en 1K buses y éstos a su vez en 64K nodos que contienen cada uno 4K. Este espacio se direcciona con 32 bits y se divide en:

- CSR core (512 bytes): Utilizado para inicializar, comprobar y configurar el nodo.
- Area dependiente del FB+ (512 bytes): Sirve para manejar una determinada implementación en particular

- ROM (1K bytes): Se utiliza como una ventana para acceder a la ROM del módulo en donde el monarca puede determinar el tamaño de la memoria y de las unidades extendidas del módulo.

- Espacio de unidades inicial (2K bytes): Aquí se direccionan los registros específicos para cada nodo. Si este espacio no es suficiente, se utilizará el espacio de unidades extendido definido por FB+.

El chipset de FB+ se compone de tres componentes

- DPU: Se encarga de almacenar los datos, decodificar las direcciones y de codificar/decodificar el modo empaquetado.
- IOC: Genera la temporización para el almacenamiento de los datos y la decodificación de las direcciones en la DPU. Maneja los protocolos de FB+ y el del interfaz con el Host Bus (HIF).
- ABC: Realiza el arbitraje distribuido para ganar el FB+ y transfiere los mensajes de arbitraje y de interrupciones sobre el bus de arbitraje.

El CSR bus le permite al procesador (a través del HIF) acceder a los CSRs del IOC y del ABC. Así mismo, el interface del CSR traduce las direcciones y datos de 32 bits del HIF en direcciones de 12 bits y datos de 8 bits para poder acceder al IOC y ABC.

El conjunto de integrados del FB+ se encarga de recibir las peticiones para realizar accesos externos del HIF generando la transacción de FB+ apropiada y de manejar las peticiones que llegan del FB+ generando la transacción apropiada del HIF. Inicialmente el chipset recibe una transacción y actúa como un esclavo. Como respuesta a ésta se convierte en maestro del otro lado. Para las transacciones de salida del HB, actuará como un esclavo del host y como maestro de FB+.

4 Inicialización y configuración

FB+ necesita un sistema de inicialización muy robusto para soportar la inserción en vivo de diversos módulos, además de varias funciones especiales que se realizan en el CSR. El conjunto de integrados de Texas Instruments utiliza la señal de reset de FB+ (RE*), además de 4 señales locales (RST*, SYSRESET*, BINIT* y BUSY*) y un CSR de control de reset para realizar las operaciones de reset necesarias en el módulo.

La línea de RE* del FB+ inicializa la lógica de interconexión al bus de todos los módulos y genera una señal de reset que puede ser usada en todas las placas. Dependiendo del tiempo en que esté activa esta señal indicará un estado de inserción en vivo, inicialización de bus o reset del sistema.

4.1 Encendido

Después de conectar el sistema a la alimentación se deben inicializar todos los módulos mediante un reset inicial. A este proceso se le llama *system power-up reset*. Se pueden utilizar dos mecanismos para inicializar el sistema:

1. Un agente que detecte la condición de encendido activa RE* lo bastante para causar la inicialización de los demás módulos. Normalmente es el sistema de alimentación el que se encarga de activar RE* antes de que cualquier voltaje de alimentación que va por el bus alcance el 40 % de su valor nominal. La señal permanecerá entonces activada entre 100 y 200 ms más, permitiendo a los módulos que se detecte el reset del sistema.

2. Un módulo se resetea a sí mismo y entonces realiza el reset del sistema escribiendo en el bit de reset del registro de control de reset del CSR del IOC.

4.2 Reset del sistema

Cualquier módulo que inicialice el sistema activará la señal RE* entre 100 y 200 ms. Todos los módulos monitorizan esta señal y cuando se detecta que está activa durante más de 30 ms se produce un reset del sistema. Entonces el módulo activa RE* y realiza las operaciones internas necesarias en el módulo para permitir su participación es las transacciones de bus. Cuando termina con estas operaciones desactiva RE* indicando que ya está listo para participar en las transacciones y se espera a que la línea en colector abierto del bus RE* se desactive indicando que todos los módulos están listos para realizar operaciones de bus.

4.3 Inicialización del bus

Si los módulos detectan la señal de RE* activa durante más de 2 ms inician una operación de inicialización de bus. En esta operación se inicializan los drivers de FB+ y se espera para determinar si es también un reset del sistema. Si la señal de RE* se desactiva antes de 30 ms se continúa con la operación normal.

4.4 Inserción en vivo

Cuando se inserta un nuevo módulo en el sistema, lo primero que este hace es comprobar si la señal RE* está activa. Si no es así, el módulo supone que ha sido insertado en un sistema en funcionamiento (inserción en vivo). En caso contrario se espera para determinar si es un reset completo del sistema.

El módulo insertado en vivo debe tener activa la señal de RST* para inicializar todas las máquinas de estado, mantener las señales bidireccionales en estado de alta impedancia y prepararse para el alineamiento. Cuando esta señal se desactiva, si RE* no está activa, el chipset se espera 130 ms para la inicialización de la placa antes de comenzar el alineamiento mediante la aserción de la línea REO. En este estado, se pueden recibir transacciones del FB+ en modo esclavo para servir los accesos al CSR. El alineamiento comienza activando la señal RE* para notificar a todos los otros módulos que necesita un alineamiento. Estos deben entonces programar el registro de temporización de las transacciones para limitar la transacción actual a 128 μs. Cuando termina la transacción y la secuencia actual de arbitraje, el bus pasa a un estado de inactividad (*idle*). El módulo insertado espera a detectar el estado *idle* durante al menos 1 μs y entonces activa AI y AR para completar el alineamiento. Entonces desactiva RE* permitiendo a todos los módulos que continúen con sus operaciones.

4.5 Extracción en vivo

Los protocolos de FB+ también permiten la extracción de módulos sin perturbar el funcionamiento del sistema. El operador debe de hacer saber a la placa por medio de algún mecanismo que va a ser extraída para que ésta pueda terminar con las tareas que estaba realizando y pueda copiar los datos pertinentes a otras áreas del sistema. El módulo debe entonces aislarse del sistema para no participar ya en ninguna transacción de bus. En ese momento el módulo debe únicamente de activar AI* y/o AK* y posiblemente alguna de las líneas de sincronización del arbitraje.

5 Selección del monarca

El monarca es el procesador elegido para llevar a cabo la inicialización y la configuración de todos los módulos de un bus. Para seleccionar al monarca se pueden utilizar dos mecanismos:

1. Hardware: El módulo del slot 1 es el monarca. Este método es apropiado cuando disponemos de un árbitro central, puesto que puede ser éste el que haga de monarca.

2. Software: Todos los módulos envían un mensaje de arbitraje que contiene su dirección geográfica y un código indicando que el mensaje es un mensaje de elección de monarca. El primer mensaje recibido indica quién es el monarca.

Para poder seleccionar al monarca, el chipset de TI asigna al nodo una dirección que identifica los recursos locales al CSR sin tener en cuenta la dirección geográfica del nodo. De esta forma el monarca puede configurar su CSR local justo después del reset, sin la necesidad de conocer su dirección geográfica, siendo también independiente el software de inicialización del slot en donde se esté ejecutando. Los CSRs de envío de mensaje de arbitraje de la FIFO de mensajes son registros que se localizan en una misma dirección en donde las escrituras las sirve el registro de envío y las lecturas las sirve la FIFO. Existe un bit en el registro de estado de interrupción del CSR del IOC que nos indica que se han recibido todos los mensajes de arbitraje. Estos CSRs se utilizan en la selección del monarca por el método 2. Para enviar el mensaje, se escribe en CSR de envío de mensaje de arbitraje y cuando se activa el bit de FIFO de recepción no vacía, el primer mensaje de la cola contiene la identidad del monarca.

6 Configuración del nodo

El monarca debe de establecer el protocolo de FB+ apropiado que permita tanto la comunicación con todos los módulos del sistema como la comunicación optimizada entre módulos compatibles con un profile en particular. Se deben evaluar tanto las características del nodo como las del sistema para obtener un rendimiento máximo (tipo de arbitraje, mapa de memoria, valores para los temporizadores, retardos de bus).

El monarca puede configurar todos los nodos a través del FB+, ya que inmediatamente después del reset éstos ya pueden recibir transacciones del CSR. Para que el módulo actúe como maestro de alguna transacción de FB+, se deben reprogramar dos registros: el que define el tipo de arbitraje y el que habilita al maestro. Si este bit está inactivo y se quiere hacer una transacción remota, no se pide el FB+ sino que se insertan estados de espera de forma indefinida en la transacción del host. En ese momento, si el monarca quiere habilitar el bit desde el FB+ se echa hacia atrás la transacción del HIF para que el monarca pueda proceder. La operación de configuración se puede realizar por medio de transacciones al HB (el monarca se autoconfigura) o accediendo al CSR a través del FB+ (el monarca configura a todos los posibles maestros).

Los módulos que tienen memoria compartida y unidades extendidas necesitan una programación adicional aparte de la configuración mínima. Los registros de base de memoria (memory base) y de límite de memoria (memory limit) definen las direcciones de memoria que debe servir el módulo local. Lo mismo ocurre con los registros base y límite de las unidades extendidas. El monarca realiza la programación de estos registros tomando la información de una ROM que reside en el espacio del CSR local al módulo. Cuando los registros están programados, el módulo responde como un esclavo a todas las transacciones de FB+ que hagan referencia a estas direcciones.

Los atributos que definen las transferencias también tendrán que ser programados para conseguir el mayor rendimiento posible. Se tendrá que programar el retardo de propagación del bus, prioridad de

las interrupciones y muchos otros registros de configuración. También se tendrá que reprogramar la temporización de las operaciones de reintento de bus y la de los errores de timeout en las transacciones.

El chipset también permite la generación de interrupciones para manejar condiciones de error o de estado. Las interrupciones del módulo necesarias para el profile B las genera el IOC, mientras que las interrupciones del sistema las genera el ABC utilizado un mecanismo de interrupciones dirigidas (*targeted interrupts*).

Todas las interrupciones son enmascarables y si se activa un bit en el registro de habilitación de la interrupción entonces se genera la interrupción.

Existe un conjunto de registros de test en el CSR que permiten a cualquier procesador del sistema iniciar un test local. Si escribimos en el registro de inicio del test se produce una interrupción si está habilitado el bit correspondiente y pone el registro de estado del test en el estado de comprobando (10000) o permanece en el estado nulo (00000) indicando que no se ha implementado ningún test. El software del módulo será el encargado de escribir el valor apropiado del estado del test en el registro de estado del test cuando éste finalice.

7 Descripción del Host Interface

El Host Interface (HIF) conecta el chipset del FB+ con el bus del módulo local. Este módulo puede estar constituido por procesadores, memoria, registros de control y de estado y por unidades de E/S. El módulo se comunica con el FB+ a través de este interface y también se utiliza para pasar información de CSR que llega desde el FB+ a través de la DPU hacia los otros dispositivos del chipset o hacia el CSR local.

El HIF funciona mediante un protocolo síncrono utilizando transferencias simples o en ráfaga, soportando además características de tiempo real y de tolerancia a fallos.

El principal objetivo del HIF es el de suministrar escalabilidad y versatilidad a un conjunto de sistemas. Con objeto de satisfacer estas necesidades, se deben cumplir los siguientes objetivos:

- Suministrar un interface genérico que comunique fácilmente con muchos procesadores, dispositivos de E/S, y buses.
- Suministrar datos y direcciones escalables. Todas las combinaciones de direcciones de 32 o de 36 bits y datos de 32 o 64 bits son soportados.
- Suministrar transferencias síncronas en ráfaga para dispositivos tanto rápidos como lentos. Los lentos necesitan un estado de espera entre cada dato.
- Suministrar un mecanismo síncrono para las transferencias simples.
- Soportar hosts tanto individuales como multimaestros.

7.1 Transacciones

Las transferencias en ráfaga permiten al host transferir 8, 16, 32 o 64 bytes en una única transacción. Esta consiste en una fase de direcciones seguida por una ráfaga de datos que puede ser de alta velocidad o de baja velocidad. En la de alta velocidad se transmite un dato en cada flanco de subida del reloj, mientras que en la lenta se transmiten los datos en flancos alternos cuando la señal de BSTRDY* está activa. El esclavo puede seleccionar el ancho y la velocidad de la ráfaga mediante las señales DSACK1* y DSACK0* respectivamente.

El HIF soporta anchos de transferencia de 8, 16, 24, 32 ó 64 bits. Las transferencias simples pueden ser de 8, 16, 24 ó 32 bits. El maestro utiliza TSIZE<1:0> para indicar el ancho deseado teniendo que estar DW64 desactivado. Las transferencias en ráfaga pueden ser de 32 o de 64 bits de ancho, dependiendo si la señal DW64* está activada o no.

Existen 3 tipos de estados de espera:

1. Estados de espera del maestro: El maestro pide el HIF, vuelca la información de las direcciones, y entonces puede mantener HDS* desactivada hasta que los datos están listos para ser enviados.

2. Estados de espera en las direcciones del esclavo: El esclavo puede insertar estados de espera en la fase de direcciones no respondiendo con la activación de DSACK1*, DSACK0* y BSTRDY* hasta que está listo para empezar con la fase de transferencia de datos.

3. Estados de espera del esclavo en las ráfagas: En las transferencias a ráfagas de baja velocidad se pueden insertar estados de espera si el esclavo no activa la señal BSTRDY* en los ciclos de datos.

El maestro del HIF puede ser vuelto atrás por el esclavo antes de comenzar la fase de transferencia de datos para asegurar el progreso del sistema.

El esclavo puede indicar una condición de error en cualquier momento antes de la terminación de la transacción mediante la activación de las señales BSTAT1*, 0*.

El HIF soporta direcciones tanto de 32 bits como de 36 bits, existiendo un registro en la DPU para habilitar el modo de 36 bits. En este modo, las 4 direcciones más significativas ocupan HAP<3:0>.

Para realizar el arbitraje, el HIF utiliza un protocolo de entendimiento a 3 hilos. Como varios maestros potenciales pueden coexistir en el Host Bus, es necesaria la actuación de un árbitro para determinar la posesión del bus.

Las propiedades de la transferencia (tales como anchura de datos, longitud y velocidad) de las transacciones del HIF son el resultado de una negociación entre el maestro y el esclavo. El maestro del bus es el responsable de controlar el tipo de transferencia y la cantidad de datos que se van a transferir durante la transacción. El maestro tiene cierto control sobre la velocidad de la transmisión, pero es el esclavo el que tiene la última palabra. El maestro puede pedir el ancho de los datos, la cantidad de los mismos y si los datos se transfieren en modo simple o en ráfaga. Los esclavos pueden redefinir el ancho de la transferencia y el modo de operación (simple o burst)

7.2 Transacción de escritura simple

El maestro inicia la transacción poniendo los valores apropiados en las líneas HA<31:0>, HAS*, TR/W*, HDS*, HIP*, DW64* y TBST*. HAS* se activa durante un ciclo para que el esclavo pueda capturar las direcciones. El maestro también activa TSIZE<1:0> para indicar un ancho de transferencia de 32 bits. TBST* está a nivel alto durante toda la transacción para indicar una transferencia simple. Cuando el maestro activa los datos y la paridad también activa HDS* para indicarle al esclavo que los datos son válidos. Después de activar HDS*, el maestro debe de esperar a que el esclavo reconozca el ciclo de datos para dar por terminada la transferencia.

El esclavo, una vez ha evaluado la dirección y ha determinado que es la unidad seleccionada para realizar la transacción, espera la activación de HDS* para realizar el reconocimiento de los datos activando DSACK1*, DSACK0* y BSTRDY*. Si no se activan estas señales, el esclavo está insertando estados de espera.

7.3 Transacción de lectura simple

El maestro inicia la transacción poniendo los valores apropiados en las líneas HA<31:0>, HAS*, TR/W*, HDS*, HIP*, DW64* y TBST*. HAS* se activa durante un ciclo para que el esclavo pueda capturar las direcciones. El maestro también activa TSIZE<1:0> para indicar un ancho de transferencia de 32 bits. TBST* está a nivel alto durante toda la transacción para indicar una transferencia simple. HDS* permanece inactivo hasta que el maestro está listo para aceptar los datos y la paridad. Cuando está listo activa HDS* y espera a que el esclavo ponga los datos en el bus y reconozca la transferencia mediante las señales DSACK1*, DSACK0* y BSTRDY*.

El esclavo, una vez ha evaluado la dirección y ha determinado que es la unidad seleccionada para realizar la transacción, puede poner cuando quiera los datos en el bus, pero debe esperar la activación de HDS* para realizar el reconocimiento de los datos activando DSACK1*, DSACK0* y BSTRDY*. Con el reconocimiento le indica al maestro que debe de capturar los datos en el siguiente flanco de subida del reloj. Si no se activan estas señales, el esclavo está insertando estados de espera.

7.4 Transacciones parciales

Las transacciones parciales se definen como aquellas en las que se transmiten menos de 32 bits. En la Tabla 8.3 se puede ver como mediante el uso de las señales TSIZE<1:0> y HA<1:0> se definen respectivamente la cantidad de datos a transferir y sobre qué líneas.

TSIZE<1:0>	HA<1:0>	HD<31:24>	HD<23:16>	HD<15:8>	HD<7:0>	ANCHO
LL	LL	Válido	Válido	Válido	Válido	32 bit
LH	LL				Válido	8 bit
LH	LH			Válido		8 bit
LH	HL		Válido			8 bit
LH	HH	Válido				8 bit
HL	LL			Válido	Válido	16 bit
HL	LH		Válido	Válido		16 bit
HL	HL	Válido	Válido			16 bit
HH	LL		Válido	Válido	Válido	24 bit
HH	LH	Válido	Válido	Válido		24 bit

Tabla 8.3: Tamaño de las transacciones de datos

7.5 Transacciones en ráfaga (Burst transactions)

Una transacción de este tipo puede tener 8,16,32 ó 64 bytes de longitud, pudiéndose usar un ancho de datos de 32 o de 64 bits. Estas transferencias siempre se hacen sobre direcciones alineadas a un múltiplo par de la longitud de la transacción. El maestro activa TBST* para indicar que se trata de una ráfaga, TSIZE<1:0> se ignora y se utiliza DL<1:0> para determinar la cantidad de datos a transferir. Al comienzo de la transacción el maestro suministra la dirección de comienzo de la misma, y a partir de ésta, tanto el maestro como el esclavo la van incrementando, por lo que deberán de disponer de mecanismos de autoincrementado para las direcciones. En estas transacciones es el esclavo el que elige la velocidad de la misma, por lo tanto el maestro debe de estar capacitado para asumir cualquier velocidad.

La definición de las señales de reconocimiento es la siguiente:

DSACK1*	DSACK0*	RESULTADO
L	L	Baja velocidad, 32 bit
L	H	Alta velocidad, 32 bit
H	L	Baja velocidad, 64 bit
H	H	Alta velocidad, 64 bit

En el modo rápido de transferencia no se admiten estados de espera, si bien el maestro puede retrasar el comienzo de la fase de datos manteniendo HDS* a nivel alto o puede ser el esclavo el que haga lo mismo retrasando el reconocimiento del primer dato. En el modo lento de 32 bits se transmiten como máximo 4 bytes cada dos ciclos, pudiendo el esclavo insertar estados de espera. Para ello BSTRDY* debe estar activa un ciclo después de la transferencia de datos anterior, seguidamente se desactiva durante el número deseado de ciclos de espera. Cuando el esclavo está listo para transferir los datos activará BSTRDY*.

El maestro, al comienzo de la transacción activa DW64* y TBST* para iniciar una transferencia de 64 bits de ancho. Si no se activa DW64*, el esclavo sólo puede responder como capacitado para la transferencia de 32 bits. En este tipo de transferencias, los 32 bits de datos más significativos se pasan a través del bus de direcciones.

La fase de datos se retrasa dos ciclos en este tipo de transferencias, durante el primero el maestro evalúa DSACK1* y DSACK0* por si el esclavo va a cambiar el ancho de la transacción. En el segundo ciclo, el maestro pone en alta impedancia el bus de direcciones para tener los datos ya listos en el tercer ciclo.

El esclavo tiene la capacidad de convertir una transferencia en ráfaga en una simple. Para ello pondrá DSACK1* y DSACK0* a nivel bajo sin activar BSTRDY*, con ello los datos estarán listos en el segundo ciclo de reloj después de la activación de las señales de reconocimiento y la transacción se habrá convertido en una simple. Para evitar que se pierda la posesión del bus en estas circunstancias, el maestro debe de mantener activo HBGACK* para poderse reconectar tantas veces como necesite hasta que termine la transferencia. Cuando se produce la reconexión, la dirección se incrementa en 4 bytes; sin embargo el maestro puede todavía activar TBST* y DL<1:0> como si fuera a pedir una transferencia en ráfaga. El esclavo, durante toda la transferencia responderá de la misma manera, sin activar BSTRDY*.

Cuando el chipset de TI funciona como esclavo en una transacción del HIF, transfiere primero la palabra más crítica (*Critical-word first*). Las palabras que siguen en orden a la palabra crítica se van transfiriendo a continuación hasta llegar al límite de los datos, para pasar a continuación a transferir las palabras que van por debajo de la crítica empezando por el límite inferior. Para las transferencias en ráfaga de 8 bits, se debe evaluar HA<2> para determinar la posición de comienzo de los datos. De la misma forma, para las transferencias de 16 bits se debe evaluar HA<3:2>, para las de 32 bits HA<4:2> y para las de 64 bits HA<5:2>. Si se quiere deshabilitar este modo de transferencia basta con poner a nivel alto la patilla DMAMODE de la DPU.

7.6 Operaciones de transferencia de bloques extendidos

El pin de entrada del IOC MORE* se utiliza para unir varias transacciones del Host Bus en una única transferencia de Futurebus+ cuando el tamaño es mayor de 64 bytes. Este modo de transferencia es parecido al de LOCK*, puesto que se evita la necesidad de arbitrar el FB+, sin embargo se diferencia en que ya no hace falta realizar las fases de conexión y desconexión. La fase de datos del FB+ se extiende al transferir múltiples paquetes de datos en una única posesión del bus, sin embargo sí que será necesaria realizar varias fases de direcciones en el HIF. Este modo de transferencia se logra

activando el bit PR* del registro CM0 del chipset. Si se detecta activo este bit durante la fase de direcciones, el chipset continúa enviando reconocimiento de datos en vez de proceder a la fase de desconexión. Cuando se utiliza esta característica, el usuario debe de asegurarse de que el bloque a transferir no cruza un límite de memoria y que el FB+ utiliza un modo de transferencia por paquetes. El IOC deshabilita el modo de transferencia CWF cuando se activa MORE*.

Para habilitar este tipo de transferencias se debe de activar el pin de MORE* cuando HAS* esté activo, permaneciendo así hasta la última transacción, en donde se tendrá que desactivar antes de que baje HIP*. De este modo finaliza la posesión del FB+. La DPU supone que todas las direcciones van contiguas después de la primera, por lo tanto FB+ ignora las siguientes a no ser que sea interrumpida la transacción.

7.7 Operación de ignorar

El pin de IGNORE* del IOC se utiliza para informar al chipset que debe ignorar la transacción actual del HB. Este modo es útil cuando una dirección hace referencia a dos posiciones distintas (espacio privado y espacio del CSR, por ejemplo). Al activar este pin el IOC no responde.

7.8 Condiciones de reintento y de error

Estas dos condiciones hacen que la transacción sea suspendida. En el reintento el maestro se retira del bus para rearbitrar en un instante posterior. En el error el maestro detiene la transacción, necesitándose un hardware adicional para recuperar el error.

La condición de reintento se denomina *back-off retry*. Esta condición se produce cuando se detecta un conflicto en el bus (comienza una transacción del HIF y el esclavo está todavía realizando otra transacción, el esclavo no dispone aún de los datos o se accede al chipset a la vez como un esclavo tanto del HB como del FB+). En cualquier caso el maestro se debe desconectar después de detectar BSTAT1* y BSTAT0* a nivel bajo, para volver a realizar un rearbitraje posterior.

Las condiciones de error se producen por fallos en el protocolo y en la paridad. En este caso, el maestro se debe desconectar del bus para ya no volver a conectarse. Esta condición se indica mediante BSTAT1* y BSTAT0* = LH.

7.9 Arbitraje del Host Bus

El HIF utiliza tres líneas (HBR*, HBG* y HBGACK*) para arbitrar el HB. El árbitro es externo al chipset. Cada posible maestro tiene una única señal de HBR* y HBG* para pedir el bus y para que se le avise de que lo ha conseguido. HBGACK* es una señal en colector abierto que indica quién es el maestro actual, permitiendo al HIF la selección del próximo maestro. La secuencia de eventos del arbitraje es la siguiente:

1. El posible maestro pide la posesión del bus activando HBR*
2. El árbitro activa la señal de HBG* del dispositivo que ha ganado el bus.
3. El nuevo maestro detecta la desactivación del HBGACK* por parte del maestro anterior.
4. El nuevo maestro comienza con la posesión asertando HBGACK*.

5. El nuevo maestro puede desactivar HBR* cuando quiera para permitir al árbitro seleccionar al próximo maestro.

8 CSR bus Interface

El bus del CSR se utiliza para conectar al host la ROM de identificación y de capacidades, las unidades definidas por el usuario, y los CSRs del IOC y del ABC. Este bus está separado del host para reducir la carga del host proporcionada por los dispositivos no críticos. Es un bus de 8 bits que se controla de una manera similar a las memorias estáticas y a la mayoría de los dispositivos de E/S. Debe de satisfacer los siguientes objetivos:

- Hacer que todos los dispositivos no críticos carguen de forma reducida y única al host.
- Suministrar un interface simple para la PROM, RAM, configuración de los controladores y la E/S definida por el usuario.
- Ser capaz de realizar transacciones de lectura, escritura y bloqueadas.
- Permitir múltiples maestros para que puedan acoplarse los dispositivos de E/S que utilicen DMA.

9 Transacciones remotas

Después de que un módulo ha obtenido la posesión del bus, utiliza el bus paralelo para realizar las transacciones. Algunas transacciones de bus son parte de transacciones de más alto nivel de un sistema. Cada transacción de sistema está constituida por dos partes, una petición y una respuesta. Si ambas fases tienen lugar en una única transacción de bus, a esta transacción se le denomina transacción conectada (*connected transaction*), sin embargo si la petición tiene lugar en una transacción y la respuesta en otra posterior, se les denomina transacciones divididas (*split transactions*). Las transacciones del sistema suelen ser divididas para no perder tiempo esperando a que un dispositivo lento responda. En el profile B del FB+ únicamente las dos transacciones de lectura (lectura desbloqueada y lectura parcial) pueden ser convertidas en transacciones divididas.

Las transacciones divididas necesitan de información adicional aparte de la necesaria para las transacciones conectadas. Por este motivo se suministran facilidades para transferir el identificador global y la prioridad original del peticionario y el estado del que responde en la fase de desconexión. Todos los maestros potenciales del bus necesitan soportar la capacidad de responder a las transacciones divididas.

Cada transacción tiene lugar en tres fases distintas, fase de conexión, fase de datos y fase de desconexión.

9.1.1 Fase de conexión

En la fase de conexión el maestro le pasa la información al esclavo sobre la operación a realizar y recibe información de estado de éste. Para realizar la conexión, el maestro activa las líneas AD[]*, CA[2..0]* y CM[7..0]* y las valida mediante la aserción de la línea AS*. El maestro especifica el tipo de transacción que desea realizar y si el esclavo no se ve capacitado para realizarla, bien activa la línea BE*, o si puede, bien convierte la transacción en una que sí que puede manejar. La finalización de la fase de conexión se indica mediante la desactivación de AIf.

9.1.2 Fase de datos

Si se van a transferir datos, a continuación se pasa a una fase de datos que comienza con la aserción de DS* por parte del maestro. Los datos se transfieren según el método acordado en la fase de conexión (modo forzado o modo por paquetes). No todas las transacciones tienen una fase de datos, puesto que en las transacciones partidas o cuando se ha producido un error en la fase de conexión no se transmiten datos.

La transmisión de un dato comienza con la transición que sobre DS* hace el maestro. A continuación el esclavo desactiva una de las líneas de reconocimiento (DI* o DS*). El maestro pasa al esclavo información de orden y de datos durante la primera parte, y durante la segunda parte el esclavo devuelve información de datos y de estado. La transmisión de datos en la que el maestro activa DS* se llama transmisión impar y la transmisión en la que la desactiva impar. Los protocolos en los que se utilizan los dos flancos de la señal para sincronizar se llaman protocolos de entendimiento a dos flancos.

Las transacciones reconocidas (*compelled transactions*) utilizan un protocolo en el que el esclavo está obligado a suministrar una respuesta antes de que el maestro pueda continuar. El protocolo a usar se negocia durante la fase de conexión. Este tipo de transferencia consiste en una fase de conexión seguida de un bloque de una o más fases de datos.

Los módulos compatibles con el profile B de FB* soportan las siguientes transacciones reconocidas:

- Lectura no bloqueada (Read Unlocked): Esta transacción se utiliza para transferir datos desde el esclavo al maestro. El esclavo puede convertir una transacción conectada en una dividida haciendo que la transacción inicial se convierta en una transacción sólo de direcciones.

- Lectura parcial (Read Partial): Esta transacción se diferencia de la anterior en que el maestro le dice al esclavo que únicamente va a evaluar algunos de los bytes transferidos.

- Sólo direcciones (Address Only): Esta transacción se produce cuando el esclavo convierte una transacción conectada en una dividida o cuando se produce un error en la fase de conexión.

- Respuesta a lectura (Read Response): Esta transacción es la respuesta a una transacción de lectura anterior que fue convertida en una transacción dividida. La transacción incluye una fase de datos en donde se pasan los datos al módulo original que los pidió. En ella, en la fase de conexión se transmiten el identificativo global y el identificativo de la transacción del módulo peticionario, y en la fase de desconexión información de estado. Únicamente puede existir una respuesta a esta transacción, es decir que la fase de respuesta no puede ser a su vez dividida.

- Escritura no bloqueada (Write Unlocked): Esta transacción se utiliza para transferir datos desde el maestro al esclavo. El esclavo no puede convertir esta transacción en una transacción dividida.

- Escritura parcial (Write Partial): Esta transacción se diferencia de la anterior en que el maestro le dice al esclavo que únicamente va a evaluar algunos de los bytes transferidos.

Las transferencias de datos empaquetados (*packet data transfer*) consisten en una dirección seguida por un bloque de datos de longitud fija en posiciones consecutivas. Utilizan un protocolo sin reconocimientos (*noncompelled protocol*) para los datos de forma que se logran tasas de transferencia

de bloques muy elevadas. Los participantes en una transacción de este tipo deben operar a la misma frecuencia. En la inicialización del sistema se configuran las dos tasas estándar de transferencia y se escriben los resultados en un registro en cada módulo. La velocidad baja es igual a la velocidad del dispositivo más lento capaz de realizar transferencias de paquetes, mientras que la más alta se hace igual a la velocidad más alta en que la mayoría de los módulos puede operar. En la fase de conexión de la transacción se decide la velocidad de la transferencia, aunque siempre se puede forzar a utilizar la velocidad más baja.

Los módulos que participan en una transferencia de datos empaquetada deben saber a priori cuántas transferencias simples va a consistir la transacción. Para ello existe un registro en el CSR que indica las longitudes soportadas (2, 4, 8, 16, 32 y 64 palabras, estando una palabra compuesta por el ancho total del bus de datos). En la fase de datos el maestro especifica una de estas longitudes estándar.

9.1.3 Fase de desconexión

Es la fase en la que el maestro termina la transacción y se desconecta del esclavo. El maestro puede entonces terminar con la posesión del bus pasando al estado idle o puede empezar otra transacción iniciando una nueva fase de conexión.

En la fase de desconexión, en el caso de una transacción dividida, el maestro le pasa la suficiente información al esclavo para que éste le pueda identificar y reconectarse en una fase posterior. Cuando el módulo que debe responder se hace maestro del bus para pasar los datos, en la fase de desconexión indicará el éxito o el fallo de la transacción. La información la transmite el maestro al esclavo a través de las líneas AD[]* y CM[]* al comienzo de la fase. Al terminar el esclavo transmite información de estado con la desactivación de AKf.

Cuando un módulo no puede responder en un momento dado porque tiene el recurso requerido ocupado, lo notifica al maestro mediante la condición de ocupado (*Busy*) que no modifica el estado del sistema. Mediante esta característica se asegura el progreso del sistema a través de los registros *Busy Retry* del CSR.

Estas transacciones requieren el uso del FB+ para completarse. Para servir una transacción de un módulo A a un módulo B, el chipset del módulo A actúa como esclavo del HIF y realiza una transacción del FB+ como maestro del mismo. Por el otro lado, el chipset del módulo B se identifica como el esclavo seleccionado del FB+ y realiza una transacción a su HIF como maestro.

9.1.4 Ejemplo de transacción de escritura

En este ejemplo vamos a suponer que se produce una escritura en ráfaga al HB con la señal MORE* activada a una posición remota de memoria.

1. El procesador del módulo A realiza una transacción de escritura en el HIF A a una dirección que está localizada en el módulo B.

2. La DPU decodifica la dirección e informa al IOC que la transacción es remota mediante la deselección de HADEC (LLLL). HADEC es válido dos ciclos después de la activación de HAS*.

3. El IOC decide participar en la transacción del HIF del módulo A y pide el FB+ mediante la activación de la línea RQ0. Mientras tanto está insertando estados de espera hasta que

determina que el HIF puede llenar la FIFO de la DPU para no tener que hacer esperar al FB+.

4. Cuando el árbitro activa GR y el maestro anterior desactiva ET* el módulo A comienza su posesión del FB+. A los transceivers se les pasa las señales de capacidades (CA<2:0>O = LLL), comandos (CM<7:0> = 0x31) y direcciones (AD<31:0>) junto con la activación de las señales de habilitación de los transceivers BTL de comandos y direcciones (CMWR* y ADDRV*). A continuación se activa ASO para empezar con la transacción del FB+.

5. El IOC del módulo B activa la señal AKO nada más detectar la activación de ASI, capturando también la DPU las direcciones e indicando al IOC que el acceso es a la memoria del host mediante las señales FADEC (HLLH). El IOC utiliza FADEC para determinar que es el esclavo seleccionado y responde mediante la activación de ST20 y la desactivación de AIO, iniciando al mismo tiempo la petición del Host Bus mediante la activación de HBR*. Cuando consigue el bus (HBG* activado y HBGACK* desactivado), activa HBGACK*, desactiva HBR* y realiza el entendimiento con la DPU y el controlador de memoria para suministrar la dirección mediante la activación de las líneas de direcciones, HAS* y HIP*.

6. Cuando el IOC del módulo A detecta la activación de AII y la desactiva, completa la transacción del host realizando el entendimiento con el maestro del HIF del módulo A e indicando a la DPU que la transferencia es de tipo ráfaga (HMODE = HLH) entre el HIF y la FIFO. Cuando ya se pueden transferir los datos al FB+ la DPU activa DATAAV* mientras existan datos en la FIFO. Cuando no le queda espacio suficiente a la FIFO para otra transferencia en ráfaga del host bus, se desactiva SPACEAV* hasta que se vuelva a hacer sitio pasando los datos a través del FB+. Cuando la transacción del HIF termina se libera este bus para que otro maestro pueda ocuparlo. El IOC y la DPU continúan transmitiendo los datos sobre el FB+ hasta que se completa toda la transferencia. El IOC del módulo A utiliza DSO para informar al esclavo que el siguiente paquete de datos está listo para ser transferido por el FB+. DKI y DII se utilizan para determinar si el esclavo está listo para la transferencia. El IOC del módulo B utiliza DKO y DIO para indicar que esta listo para recibir el siguiente paquete de datos.

7. En el módulo B, HDS* permanece a nivel alto hasta que los datos se transfieren a la FIFO. Esta indica que los datos están disponibles para ser transferidos al HIF B activando DATAAV* y HDS*. Si la FIFO se llena se desactiva DATAAV*.

8. Después de que termine la transferencia de datos sobre el FB+ el IOC del módulo A se desconecta (desactiva ASO) y libera el bus (desactiva RQ0). El IOC/DPU del módulo B realiza el protocolo de desconexión del FB+ y completa la transmisión de los datos al HIF B desconectándose después del mismo mediante la desactivación de HBGACK*.

9.1.5 Ejemplo de transacción de lectura

En este ejemplo vamos a suponer que se produce una lectura en ráfaga a una posición remota de memoria. Se enlazan varias lecturas simples en una única transferencia de FB+ multipaquete.

1. El procesador del módulo A realiza una transacción de lectura en el HIF A a una dirección que está localizada en el módulo B.

2. La DPU decodifica la dirección e informa al IOC que la transacción es remota mediante la deselección de HADEC (LLLL). HADEC es válido dos ciclos después de la activación de HAS*.

3. El IOC decide participar en la transacción del HIF del módulo A y pide el FB+ mediante la activación de la línea RQ0. Mientras tanto está insertando estados de espera hasta que los datos han sido transmitidos por el FB+ y están listos en la FIFO.

4. Cuando el árbitro activa GR y el maestro anterior desactiva ET* el módulo A comienza su posesión del FB+. A los transceivers se les pasa las señales de capacidades (CA<2:0>O = LLL), comandos (CM<7:0> = 0x31) y direcciones (AD<31:0>) junto con la activación de las señales de habilitación de los transceivers BTL de comandos y direcciones (CMWR* y ADDRV*). A continuación se activa ASO para empezar con la transacción del FB+.

5. El IOC del módulo B activa la señal AKO nada más detectar la activación de ASI, capturando también la DPU las direcciones e indicando al IOC que el acceso es a la memoria del host mediante las señales FADEC (LLLH). El IOC utiliza FADEC para determinar que es el esclavo seleccionado y responde mediante la activación de ST20 y la desactivación de AIO, iniciando al mismo tiempo la petición del Host Bus mediante la activación de HBR*. Cuando consigue el bus (HBG* activado y HBGACK* desactivado), activa HBGACK*, desactiva HBR* y realiza el entendimiento con la DPU y el controlador de memoria para suministrar la dirección.

6. Los datos del HIF B se ponen en la FIFO. Tan pronto como se detecta la activación de DSI y están disponibles los datos (DATAAV* activo), el IOC del módulo B activa DKO y comienza la transferencia empaquetada. Cuando está listo para el próximo DSI, desactiva DIO. La siguiente transferencia de datos comienza cuando la señal DSI está desactivada y el siguiente paquete de datos está listo. En esta transferencia primero se activa DIO y al final de la segunda transferencia empaquetada se desactiva DKO. Las subsiguientes transferencias se desarrollan de la misma forma. Cuando no le queda espacio suficiente a la FIFO para otra transferencia en ráfaga del host bus, se desactiva SPACEAV* hasta que se vuelva a hacer sitio pasando los datos a través del FB+.

7. Cuando termina la transferencia en el HIF del módulo B, se libera el HB y se continua transfiriendo los datos sobre el FB+.

8. Cuando llegan los datos al módulo A, la FIFO lo indica mediante la activación de DATAAV*. Si ésta se llena, se desactiva SPACEAV* indicando que no se pueden recibir más datos del FB+.

9. Después de que termine la transferencia de datos sobre el FB+ el IOC del módulo A se desconecta (desactiva ASO) y libera el bus (desactiva RQ0). El IOC/DPU del módulo B realiza el protocolo de desconexión del FB+. El IOC/DPU del módulo A completa la transmisión de los datos al HIF A desconectándose después del mismo mediante la desactivación de HBGACK*.

10 Arbitraje del FB+

Cuando un módulo necesita enviar o recibir datos de otro módulo o de la memoria del sistema debe primero adquirir el bus para realizar la transferencia. Como más de un módulo puede pedir el bus simultáneamente, es necesario la existencia de un arbitraje para que sólo un módulo pueda poseer el bus en un instante determinado. Los módulos tienen una prioridad para acceder al bus basada en un

número de prioridad que tiene cada uno. En el arbitraje se utiliza un algoritmo que asegura que todos los módulos que usen la misma prioridad tendrán la misma probabilidad de acceder al bus. Si este algoritmo no es capaz de dar un ganador de bus, se utiliza la dirección geográfica del módulo para decidir en último caso quién será el próximo maestro del bus.

En el FB+ se pueden distinguir dos métodos de arbitraje: centralizado y distribuido. En el esquema centralizado, cada módulo utiliza líneas separadas de petición (*request*) y de asignación de bus (*grant*), que van dirigidas a un árbitro central. Este dispositivo resuelve las peticiones simultáneas y distribuye la concesión entre los módulos peticionarios. En el arbitraje distribuido se resuelven las peticiones simultáneas y la asignación del bus en una competición que tiene lugar en un bus dedicado a este tipo de arbitraje, que nada tiene que ver con el bus paralelo. Utiliza sus propias líneas de sincronización y de datos y su propio protocolo. Este bus de arbitraje distribuido también se utiliza para enviar mensajes en sistemas tanto con arbitraje centralizado como con arbitraje distribuido pudiendo enviar señales o dar cuenta de eventos entre los distintos módulos. En los sistemas conformes con el Profile B se utiliza para enviar un mensaje de fallo de alimentación.

10.1.1 Sistema con arbitraje totalmente distribuido

Dentro del módulo, un controlador de arbitraje distribuido (ABC) utiliza el bus de arbitraje para realizar la petición del bus y para enviar o recibir mensajes de arbitraje distribuido.

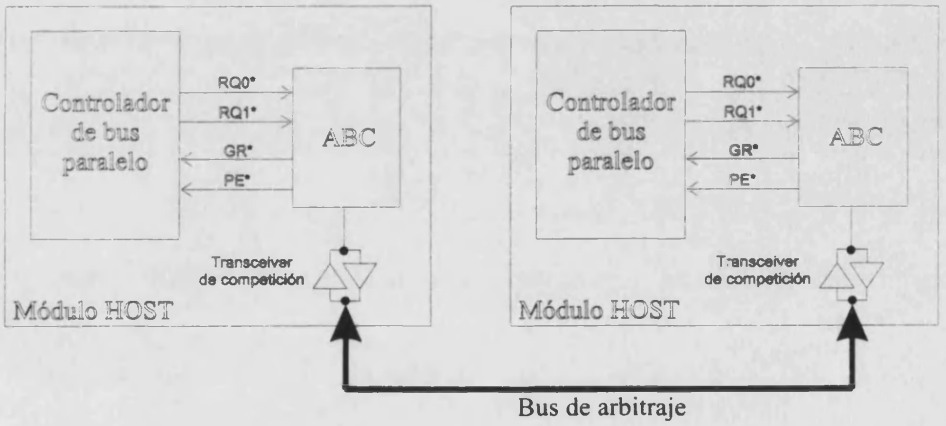


Figura 8.4: Arbitraje distribuido

10.1.2 Sistema con arbitraje centralizado con mensajes de arbitraje

En el modo centralizado pueden existir módulos con capacidad para realizar arbitrajes distribuidos que utilizan el bus de arbitraje para programar las prioridades del módulo en el árbitro centralizado y para recibir o enviar mensajes de arbitraje. El árbitro central es el que maneja la adquisición del bus por parte de todos los módulos. Es interesante que en el módulo del árbitro central se incluya un módulo de arbitraje distribuido para poder recibir mensajes de arbitraje y poder programar las prioridades de los módulos del sistema. El arbitraje centralizado ofrece el mayor rendimiento puesto que minimiza el tiempo de concesión de bus al peticionario.

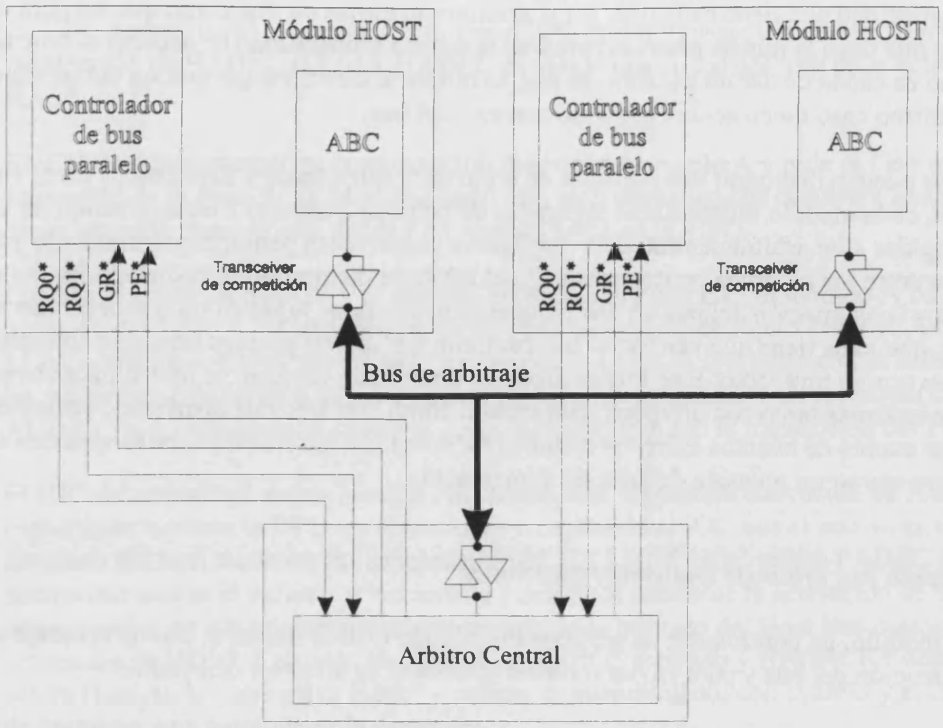


Figura 8.5: Configuración de árbitro central con arbitraje distribuido para los mensajes

10.2 Protocolo de arbitraje distribuido

En el modelo de arbitraje distribuido, el arbitraje se utiliza para que un módulo peticionario del bus pueda ser el maestro del bus de transferencia de datos. El bus de arbitraje se puede utilizar para seleccionar al próximo maestro del bus de datos o para enviar mensajes, independientemente de la actividad que realice el maestro actual sobre el bus de datos. Cuando termina el maestro de utilizar este bus, el proceso de arbitraje llega a su fin. De esta forma el rendimiento aumenta. Mientras que el siguiente maestro espera a que se libere el bus, puede tener lugar un nuevo proceso de arbitraje con mayor prioridad, haciendo que este nuevo maestro se tenga que retirar.

El protocolo usado es de tipo asíncrono a tres hilos. Se utilizan tres señales en colector abierto (AP*, AQ* y AR*) que son monitorizadas y controladas por todos los módulos durante las diversas fases en las que se descompone el proceso de arbitraje. Se utilizan para mantener a todos los módulos sincronizados con las actividades del bus de arbitraje. Estas actividades se dividen en tres pares de fases. La primera fase de cada par siempre empieza con la aserción por parte de un módulo de alguna de las tres líneas de sincronización. La segunda fase de cada par siempre empieza por la desactivación de otra de las líneas de sincronización por parte de todos los otros módulos. Cuando el último módulo desactiva esta línea es cuando comienza realmente la fase, debido a que las líneas son en colector abierto, por lo tanto será el módulo más lento el que controle los cambios de fase en el proceso de arbitraje. En la Tabla 8.4 se puede ver la tabla de estados de las señales de sincronización.

Phase	AP*	AQ*	AR*	Phase-Pair Name
0	0	0	1	Idle
1	1	0	1	Competition
2	1	0	0	
3	1	1	0	Error-Check
4	0	1	0	
5	0	1	1	Message Confirmation
0	0	0	1	Idle

Tabla 8.4: Tabla de estados de las señales de sincronización

Todos los módulos que utilicen las líneas de mensaje de arbitraje deben estar sincronizados con las actividades de este bus y deben activar las líneas de sincronización en el momento adecuado, incluso aunque no estén participando en el arbitraje actual. En la se pueden ver las distintas señales que utiliza el bus.

El bus de arbitraje AB()* sirve para que los módulos pongan en el sus números de competición (CN[]*) durante la primera fase del arbitraje. También es un bus en colector abierto por lo tanto lo que aparece por las líneas es la función OR de todos los números de competición. A medida que los módulos reconocen peticiones con más alta prioridad (evidenciado por la existencia de un número mayor en el bus de arbitraje), van quitando los bits menos significativos de sus números de competición. Este proceso continúa hasta que se encuentra al peticionario con mayor prioridad.

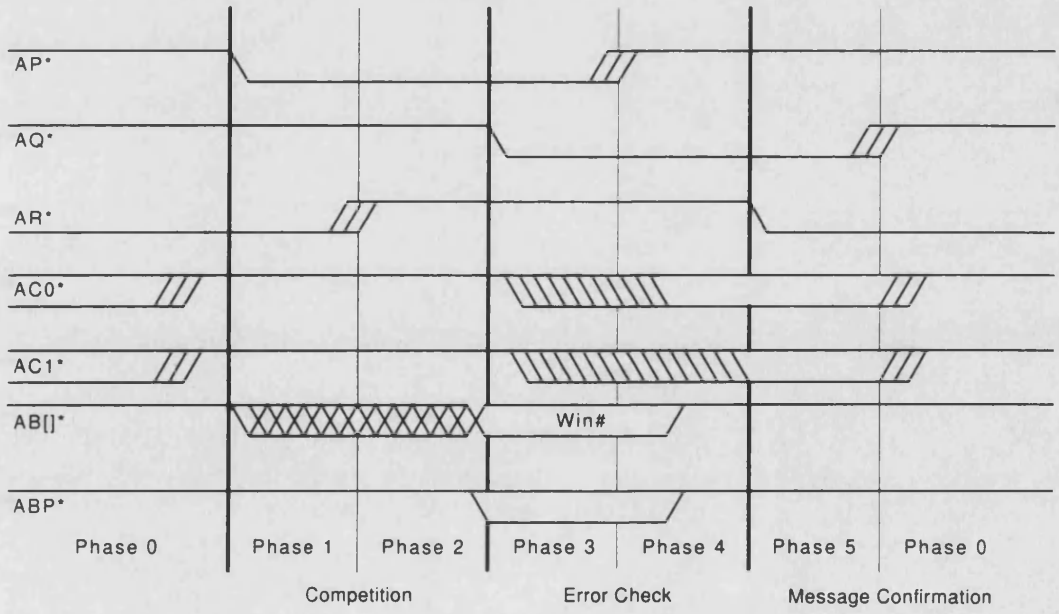


Figura 8.6: Fases del bus de arbitraje

El proceso de arbitraje tiene lugar mediante las siguientes fases, como se puede ver en la Figura 8.6:

- Fase 0: El bus de arbitraje está inactivo esperando a que un módulo pida un arbitraje. Un módulo activa AP* para pasar a la fase 1.

- Fase 1: Es la fase de decisión en donde se da una última oportunidad a los módulos que quieran entrar en el arbitraje actual. Los competidores ponen sus prioridades en el bus. La fase 1 se completa cuando todos los módulos han desactivado la señal AR*.

- Fase 2: Es la fase de competición en donde los módulos ya han puesto sus números de competición sobre el bus de arbitraje. Los transceivers identifican al ganador activando el transceiver del ganador la señal de WIN. El controlador de arbitraje espera cierto tiempo antes de comprobar el estado de esta señal. El tiempo que espera se define en un registro del CSR (Competition Setting Time). El módulo ganador activa la señal AQ* para pasar a la fase 3.

- Fase 3: Es la fase de comprobación de error puesto que ya hay un ganador y tiene lugar cuando todos los módulos evalúan el número de competición del ganador para determinar el tipo de arbitraje (mensaje o adquisición de bus), si se necesita una segunda pasada o si han habido errores. Todos los módulos esperan a que el maestro actual complete la transacción (mediante la desactivación de AS*). Si la desactivación de AS* se produce para un mensaje de arbitraje o si necesita de una segunda fase, cuando todos los módulos desactiven AP*, se pasará a la fase 4.

- Fase 4: Tiene lugar cuando el maestro actual tiene la opción de activar una bandera para cancelar el cambio del dueño de bus en la fase 5 para poder realizar transacciones adicionales. Todos los módulos esperan a que el maestro active AR* e inicie el paso a la fase 5. En el caso de modo centralizado, mensajes de arbitraje o cuando se necesita una segunda fase de arbitraje, todos los módulos pasan a la fase 5 activando AR*.

- Fase 5: Tiene lugar si no han ocurrido errores y no se necesita otra segunda fase de arbitraje. En esta fase el módulo elegido se hace maestro o el mensaje de arbitraje es aceptado por todos los módulos. El proceso de arbitraje pasa a la fase 0 cuando todos los módulos han desactivado AQ*. Si se necesita otro paso, éste empieza en la fase 0 y el proceso se repite.

Durante la fase 2 los números de competición de todos los módulos que deseen participar en el arbitraje se ponen en el bus y es la lógica de competición de cada módulo la que resuelve la prioridad mayor presente en el bus. El modo de arbitraje (centralizado o distribuido) y el número de prioridad más alto permiten a los dispositivos determinar cual ha sido el tipo de arbitraje realizado: mensaje, petición de bus, actualización de prioridad. Si los competidores requieren una resolución de más de 8 bits, se realizarán dos pasos en el arbitraje.

Las señales AC1* y AC0* indican condiciones de error y suministran control adicional sobre el protocolo del arbitraje. Existe otra señal, ABP* (Arbitration Bus Parity) que es activada por el ganador del arbitraje para indicar la paridad par de su número de competición.

10.3 Protocolo de arbitraje centralizado

Si existe un árbitro central se debe usar este protocolo para adquirir el bus. El bus de arbitraje se puede usar para mensajes y para programar las prioridades. Este protocolo es un protocolo asíncrono de dos señales. Para realizar el entendimiento entre el módulo peticionario y el árbitro central se utilizan las señales de petición RQ0 y RQ1 y la señal de concesión GR. Dependiendo de la señal de petición que se utilice se tendrá una prioridad u otra, siendo ambas programables. Cuando un módulo activa una señal de petición, el árbitro central revisa las peticiones de todos los módulos y concede el bus al de

mayor prioridad. El maestro de bus, cuando está realizando la última transacción desactiva la línea de petición y a continuación el arbitro desactiva su señal de concesión, activando seguidamente la línea de concesión del próximo maestro. El nuevo maestro podrá empezar a utilizar el bus cuando el maestro anterior desactive la señal ET* indicando que la última transacción ha finalizado.

RQ0* (Request Low) y RQ1* (Request High): Los módulos activan estas señales para pedir el bus con una prioridad baja o con una prioridad alta respectivamente. RQ0* se utiliza en todas las transacciones menos en las respuestas a lectura. Cualquiera de estas líneas se mantiene activa hasta la última transacción, en la que normalmente se desactiva al activar ET*, pero que sin embargo se puede mantener activa hasta que se desactiva AS*. Nada impide activar RQ0* aún estando activada RQ1* y viceversa.

GR* (Grant): El árbitro central activa esta línea para indicarle al módulo que es el maestro elegido en respuesta a su petición. El módulo espera a que se desactive ET* antes de comenzar la transacción por el bus paralelo. El árbitro deja esta línea activa hasta que el maestro elimina su petición.

PE* (Preempt): Existe una señal de desalojo que envía el árbitro a todos los módulos para informar al maestro actual que un módulo con mayor prioridad está esperando el bus. El maestro decidirá si le deja el bus o no. El árbitro desactiva PE* a la vez que desactiva GR*.

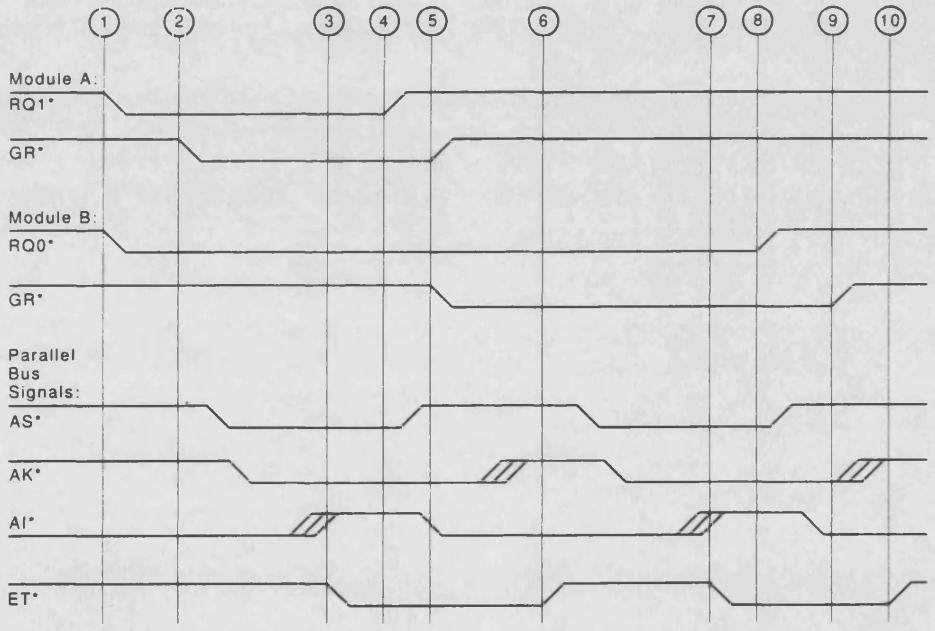


Figura 8.7: Arbitraje centralizado para dos módulos

En la Figura 8.7 podemos ver la secuencia de fases que sigue un arbitraje para dos módulos:

1. Los módulos A y B realizan la petición del bus

2. El árbitro central activa la GR* del módulo A haciéndolo el maestro elegido. Como el bus está libre (ET* desactivada), el módulo A empieza inmediatamente una transacción.

3. El maestro activa ET* después de empezar con la transacción.

4. Durante la última transacción el módulo A ya no quiere más el bus y cesa su petición.
5. El árbitro detecta el cese y desactiva la concesión al módulo A pasándosela al módulo B mediante la activación de su GR*. Ahora B es el maestro elegido.
6. El módulo A, después de desactivar las señales del bus paralelo desactiva ET*.
7. El módulo B es el maestro actual y activa ET* después de empezar la primera transacción.
8. Durante la última transacción el módulo B ya no quiere más el bus y cesa su petición.
9. El árbitro detecta el cese y desactiva la concesión al módulo A. Como no hay más peticiones el árbitro no genera ninguna concesión.
10. El módulo B, después de desactivar las señales del bus paralelo desactiva ET*.

10.4 Protocolo de mensajes arbitrados

Las líneas que se usan para realizar el arbitraje del FB+ se pueden utilizar para que los módulos envíen mensajes arbitrados. Este bus de arbitraje se puede ver como un bus independiente del bus de protocolo paralelo. Utiliza sus propias líneas de sincronización y de datos y su propio protocolo.

Los mensajes de arbitraje se utilizan para señalizar eventos del sistema, interrupciones o para pasar información general de un módulo a otro sobre el bus de arbitraje. En los mensajes, el bit más significativo del número de competición de 8 bits es siempre un 1, definiendo 128 posibles mensajes y dotándolos de una prioridad elevada. Si utilizamos arbitraje distribuido, para enviar un mensaje lo tenemos que realizar en dos fases, siendo el número de competición en la primera 0xFF dándole a los mensajes la más alta prioridad.

Los mensajes arbitrados de forma centralizada se utilizan para programar o para actualizar las prioridades de los distintos módulos en el árbitro central. Se pueden programar hasta 256 prioridades para cada uno de los dos niveles de petición. Sólo el árbitro central necesita recibir estos mensajes.

EL mensaje de arbitraje de fallo de alimentación es un mensaje de emergencia que se utiliza para notificar a todos los módulos este evento.

10.5 El controlador de arbitraje de Texas Instruments para FB+ (ABC)

Este integrado funciona tanto en modo de arbitraje centralizado como distribuido. Funciona junto a un transceiver de competición que es el encargado de pasar al FB+ los números de competición y de decidir si se ha ganado el arbitraje o no. En el FB+ existe la señal de PE* que la activa el arbitro central durante la inicialización para indicar a todos los módulos el modo de operación. Este integrado lee el estado de esta señal para programarse bien en modo de arbitraje centralizado, bien en modo de arbitraje distribuido.

El ABC proporciona unas señales de error a los otros integrados de FB+ para que se puedan activar en ellos los bits que indican condiciones de error en los registros del CSR.

El ABC se programa mediante el interface del CSR. Este integrado funciona como un esclavo del bus del CSR, permitiendo la lectura o la escritura de los registros para configurarlo, establecer las prioridades del arbitraje, enviar mensajes, observar el estado del dispositivo y recibir mensajes e interrupciones. Los registros del núcleo del CSR y los dependientes de FB+ que están implementados

en este dispositivo están mapeados directamente de acuerdo con las especificaciones del estándar del IEEE 896.2. Los registros que son específicos de este dispositivo están mapeados directamente en el área reservada para el espacio de unidades inicial definido en el mismo estándar anterior. Las direcciones que ocupan han sido coordinadas con todos los productos de FB+ de TI (Texas Instruments) y con todos los dispositivos que cumplen el estándar del CSR para que no hayan conflictos de direccionamiento.

En este integrado existe una entrada de referencia de reloj que sirve como base de tiempos para programar los retardos de los distintos temporizadores que incorpora.

10.5.1 Mensajes de arbitraje

Este integrado puede enviar mensajes independientemente de las actividades del bus de datos mediante el protocolo de arbitraje distribuido, estando bien en modo central (mensajes de arbitraje general) bien en modo distribuido (mensajes de arbitraje distribuido). Esta capacidad permite a los módulos de FB+:

- Generar y enviar eventos del sistema (interrupciones)
- Enviar y recibir mensajes definidos por FB+
- Programación de prioridades y envío de información al árbitro central

Los mensajes de arbitraje general se distribuyen a todos los módulos y se envían escribiendo un valor de 7 bits en un de los registros del CSR (existiendo por tanto 128 mensajes diferentes). La recepción de los mismos se realiza de forma diferente dependiendo de su valor:

Interrupciones dirigidas (0x80 - 0x9F): Estos mensajes producen que se active un bit específico en el registro de interrupciones dirigidas (CSR 0x50), generándose una interrupción si esta ha sido habilitada (CSR 0x54). Escribiendo en el primer registro a través del bus de datos también se puede generar una interrupción.

Existen también dos registros adicionales en el CSR. El registro de desactivación de la interrupción dirigida (CSR 0xe9c) desactiva la interrupción mediante la escritura de un '1' en el bit específico. Leyendo en este registro se puede conocer las interrupciones que están activas. El segundo registro es el registro de identificación de la interrupción dirigida (CSR 0xea0) que codifica el valor de la interrupción activa más significativa.

Mensajes recibidos en la FIFO: Los mensajes que llegan que no son interrupciones dirigidas se guardan en una FIFO de 4 palabras. Si la FIFO no está vacía se produce una interrupción local al módulo para poder leer los mensajes en el orden en que fueron recibidos. Existen varios registros de máscara en el CSR para poder definir el rango de mensajes que van a ser recibidos. Se pueden enmascarar todos los mensajes menos los de número 0xfe y 0xff. El último indica un fallo de alimentación y activa el pin de PFAIL del integrado. Si la FIFO se llena y llega un nuevo mensaje, se ignora y se genera una interrupción de sobrepasamiento de la FIFO.

Para evitar que la FIFO se llene se pueden usar las siguientes opciones:

1. Ignorar mensajes duplicados para evitar que un mismo mensaje se guarde varias veces en la FIFO.

2. Si se llena la FIFO y tiene lugar la transferencia de un nuevo mensaje, el arbitraje se detiene en la fase 3 hasta que no se lea algún mensaje de la FIFO.

3. Se utiliza la línea de error del bus de arbitraje AC0 para indicar a todos los módulos que la FIFO está llena. El error cancela la recepción del mensaje en la fase 5 haciendo que se vuelva a enviar hasta que todos los módulos lo reciben correctamente.

Los mensajes de arbitraje centrales los utilizan los módulos con arbitraje distribuido para programar la prioridad del módulo y la dirección geográfica de los dos niveles de prioridad definidos por FB+ (RQ0 y RQ1) en el dispositivo de arbitraje centralizado del sistema. Estos mensajes se envían automáticamente al programar los registros de prioridad del ABC funcionando en modo centralizado. Estos mensajes sólo los utiliza el árbitro central, por lo que son ignorados por los demás módulos.

Cuando se envía un mensaje de arbitraje centralizado se genera una interrupción de actualización de prioridad, en vez de la interrupción de envío de mensaje para avisar al módulo local que el arbitro central ha sido cargado con la nueva prioridad.

10.5.2 Adquisición del bus en modo distribuido

La activación de las líneas de petición RQ0 o RQ1 inicia el proceso de arbitraje si el módulo está alineado y además se ha activado el bit de habilitación de maestro en el CSR de control. Dependiendo de la línea que se active se utilizará una prioridad o la otra. Si el arbitraje se gana se activa la línea de GR. La petición se debe de desactivar cuando se ha recibido GR y ya no se desea utilizar más el bus. GR se desactiva al desactivar las líneas de petición permitiendo la realización de un nuevo arbitraje.

Los valores de los registros de prioridad se utilizan para construir los números de competición usados en el proceso de adquisición del bus en modo distribuido. El número de prioridad más alto es el que gana la competición y se elige como el nuevo próximo maestro. El proceso de arbitraje puede necesitar una o dos fases. Si un módulo compite con una prioridad de 0xfe ó 0xff sólo se necesita un paso. Para sistemas con prioridad única o prioridad dual si utilizamos como prioridades 0xfe y 0xff se incrementa el rendimiento eliminando la necesidad de efectuar la segunda fase.

Bit	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0
Paso 1	1	PR0	RR	GA4	GA3	GA2	GA1	GA0

Cuando varios módulos con la misma prioridad compiten por el bus, se utiliza el bit de *round-robin* y la dirección geográfica del módulo para determinar al ganador.

Bit	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0
Paso 1	0	PR7	PR6	PR5	PR4	PR3	PR2	PR1
Paso 2	1	PR0	RR	GA4	GA3	GA2	GA1	GA0

La prioridad del módulo se puede modificar aún cuando ya se ha activado alguna de las líneas de petición. Esta característica se puede utilizar cuando por ejemplo se activa una nueva tarea con mayor prioridad mientras se está esperando el bus para otra tarea, o cuando la urgencia de la tarea que está esperando el bus varía con el tiempo. Para cambiar la prioridad se pueden utilizar dos métodos:

1. Activar la otra línea de petición que no esté activada para que se utilice su prioridad en el caso de que esta sea mayor que la que ya había.
2. Escribir en el registro de prioridad de la línea de petición activa.

El arbitraje distribuido también permite la expulsión (*preemption*) del módulo que tiene el bus. Se pueden dar dos tipos:

1. Deposición del maestro elegido: Tiene lugar cuando el próximo maestro que ya ha sido elegido tiene menor prioridad que otro módulo que quiere utilizar el bus. El módulo con mayor prioridad puede forzar un nuevo arbitraje mediante una nueva petición de bus, una actualización de la prioridad del registro o un mensaje de arbitraje.
2. Expulsión del maestro actual: Cuando en el proceso de arbitraje se selecciona un nuevo maestro, el maestro actual compara su prioridad con la del nuevo. El resultado de la comparación determina si la señal de expulsión (PE) debe de ser activada. Esta señal indica al maestro actual que otro módulo con mayor prioridad quiere utilizar el bus, pero no significa que vaya a haber una expulsión.

Para permitir el *juego limpio* en el proceso de arbitraje, existe un bit de *round-robin* que se activa cada vez que se pierde una competición debido a que la ha ganado un módulo con la misma prioridad pero con una posición geográfica mayor. El bit se desactiva cuando se adquiere el bus. De esta forma, los módulos con igual prioridad tienen la misma probabilidad de adquirir el bus independientemente de la posición geográfica que utilizan. Existe un bit de *round-robin* para cada registro de prioridad.

El bloqueo del bus (*bus parking*) elimina la necesidad de realizar un nuevo arbitraje cuando el maestro anterior quiere volver a utilizar el bus y ningún otro módulo lo ha pedido mientras tanto. Las líneas de petición del bus se activan normalmente y el bus se concede de forma inmediata.

Cada vez que se le concede el bus al módulo se genera una interrupción para que éste pueda actualizar los registros de prioridad para las subsiguientes adquisiciones del bus.

10.5.3 Interrupciones locales al módulo

El ABC tiene una línea de interrupción para avisar al módulo local de que se necesita realizar alguna acción. Esta línea se activa si alguna de las condiciones de interrupción que están habilitadas está activa. La rutina de tratamiento puede leer el registro de estado de las interrupciones o el registro de desactivación para determinar la interrupción que se está procesando. El registro de identificativo de estado de la interrupción codifica la interrupción activa con mayor prioridad. Las interrupciones se eliminan escribiendo un '1' en el registro de desactivación en el bit que se quiere desactivar y de la misma forma se pueden activar manualmente escribiendo un '1' en el registro de activación de las interrupciones.

11 Manejo de fallos

Este bus está preparado para trabajar en sistemas con características de tolerancia a fallos. Para poder disponer de una estrategia de manejo de fallos válida, todos los nodos deberán de sufrir los mismos síntomas de error. Mediante pasarelas que conecten varios buses se podrá propagar la detección de un error en el bus. En este apartado se describen las prácticas utilizadas para guardar el estado después de la detección de un error, la recuperación mediante la retransmisión, y la notificación de los errores al siguiente nivel del sistema.

En el manejo de errores vamos a considerar tres categorías:

- Protocolo paralelo: Interpretación de las líneas de estado
- Captura de los síntomas de error a nivel físico por medio del CSR
- Indicación del error mediante interrupciones

A continuación se presenta una lista con los posibles errores que van a ser considerados:

- Fase de conexión

  Dirección no existente
  Error de paridad en las direcciones
  Error de paridad en una orden
  Error de paridad en una etiqueta
  Error de protocolo
  Temporización excedida en una transacción
  Intervalo de retransmisión excedido
  Identificativo de transacción no existente
  Respuesta dividida no esperada

- Fase de datos

  Error de paridad en los datos
  Error de paridad en una etiqueta
  Error de paridad en una orden
  Temporización excedida en una transacción
  Error de longitud
  Error de protocolo

- Errores internos al nodo y producidos en la parte remota de las pasarelas que pueden producir errores en los datos.

- Errores en el arbitraje distribuido y en los mensajes de arbitraje

- Errores de temporización en las transferencias divididas

De acuerdo con el Profile B, existen una serie de condiciones y de líneas de estado que no son soportadas, las cuales los maestros las interpretan como errores de protocolo:

- Estados de espera
- Estados de intervención
- Estados de transmisión simultánea

También existen una serie de errores no soportados en los Profile B que se pueden ignorar:

- Bandera de transacción (Conflicto de estado)
- Error de paridad en la etiqueta
- Error de arbitraje distribuido

y una serie de condiciones y de señales de estado que no son errores del protocolo paralelo y que no se consideran:

- Estado de ocupado
- Inicialización de alimentación
- Inicialización de bus
- Reset del sistema
- Inserción en vivo
- Errores relacionados con la alimentación del sistema

11.1 Estrategias de manejo de errores

Las reglas generales que sigue la señalización de un error en el protocolo paralelo son:

- Los errores en el protocolo paralelo se notifican en el bus usando las líneas BE* (Beat Error) o TE* (Transaction error). Cualquier módulo puede indicar un error en la fase de conexión. En las demás fases sólo pueden intervenir los módulos implicados.

- Cualquier módulo que detecte un error o que reciba una señal del protocolo paralelo que indique la detección de un error por otro módulo debe de activar los bits apropiados en los registros ERROR_HI y ERROR_LO del CSR.

- Las órdenes y operaciones no soportadas por el Profile B deben ser tratadas como errores de protocolo.

Los *errores precisos* son detectados en el momento en que se producen, mientras que los *errores imprecisos* no se detectan en el momento en que se producen, estando el sistema progresando con la condición de error. Por ejemplo se puede retrasar la señal de error debido a un problema en las temporizaciones del módulo que lo debe detectar o el error puede estar relacionado con un fallo de temporización en una transferencia dividida.

11.1.1 Detección de errores por parte del maestro

Cuando se detecta el error, si la transacción falla, el maestro debe de guardar la orden y la dirección usada en la fase de conexión en los registros de error. Además deberá de capturar las líneas de estado y de capacidades en el momento en que detecta la activación de algunas de las señales de error del protocolo paralelo (BE* o TE*) y debe de enviar una interrupción para notificar el error al software del nivel superior.

Cuando el maestro detecta el error, o la notificación del mismo, en las transacciones conectadas debe de pasar a la fase de desconexión.

El único módulo que puede detectar un error de temporización en una transacción (TRANSACTION_TIMEOUT) es el maestro. En este caso basta con que active la señal de Reset (RE*) para producir una inicialización de bus e inicializar sus máquinas de estados. Si por el contrario se detecta un sobrepasamiento en el nivel de ocupado (BUSY_TRESHOLD_EXCEEDED) simplemente debe de desconectarse del bus y no activar ninguna línea de error.

Los errores detectados en las pasarelas o en la respuesta a las transacciones divididas son notificados en la fase de desconexión de la transacción de respuesta de lectura. El maestro debe de activar el bit de UNIT_SPECIFIC error en el registro del CSR ERROR_HI. Para indicar el error, en vez de utilizar las líneas de estado se utilizan las líneas AD[2..0]* de la siguiente manera:

- Un error de memoria no existente (NXM) se señala desactivando AD2*.
- Un conflicto de estado se señala asertando AD1*.
- Cualquier otro tipo de error se señala asertando AD0* (TE*).

11.1.2 Detección de errores por parte del esclavo

Cualquier módulo que detecte un error de paridad en una dirección o en un comando no debe de activar SL* (SELECTED), sino que debe de indicar el error de la fase de conexión mediante BE*. Si no se activa ninguna de estas dos líneas se supone que es un error de NONEXISTENT_MEMORY.

Si el esclavo participa en la transacción, cuando se detecta el error se deben capturar las líneas CM[]*, ST[]*, y CA[]* en el registro del CSR ERROR_HI. En otros registros se deberán de capturar también las líneas de direcciones y de datos.

En la fase de datos se puede activar BE* para indicar un *error preciso* siempre que:

- Se detecte un error de paridad y esté activada la notificación del error.
- Se detecte un error de protocolo.

El esclavo que participa en la transacción puede indicar un *error impreciso* (errores que se activan cierto tiempo después de producirse, de tal forma que no se pueden atribuir a una instrucción o ciclo de bus en concreto) activando TE*, siempre que:

- Se detecte un error de paridad y esté activada la notificación del error, pero ésta no se realiza inmediatamente.
- TE* se debe de activar antes del final de la fase de desconexión si el esclavo había activado antes BE*.

Pueden utilizar ED* para terminar la transacción en caso de error y notificar el error usando TE* en la fase de desconexión.

En la fase de desconexión se notifican los errores mediante la activación de la línea TE*. Para poder utilizar esta señal se debe de cumplir:

- Se detecte un error de paridad en los datos o en una orden y esté activada la notificación del error.
- Se detecte un error de paridad en los datos o en una orden y esté activada la notificación del error, pero ésta no se ha producido en la fase de datos.
- Cualquier error interno ocurrido en el esclavo que no sea un error de bus pero que puede afectar a los datos.
- Se detecta un error de protocolo.

11.1.3 Captura de los errores en el CSR

Los errores que se pueden capturar son los siguientes:

Resumen del error (ERROR_SUMARY): Indica que los síntomas de error han sido capturados en los registros ERROR_HI y ERROR_LO. Los tipos de error son:

PROTOCOL_ERROR
COMMAND_PAR_ERROR
DATA/ADDR_PAR_ERROR
TRANSACTION_TIMEOUT
SPLIT_TIMEOUT
LENGTH_ERROR
BUSY_RETRY_THRESHOLD_EXCEEDED

NONEXISTENT_TRANSACTION_ID
NONEXISTENT_ADDRESS
UNIT_SPECIFIC_ERROR

Las señales que se deben capturar son:

CAPABILITY_LINES (CA[2..0]*)
STATUS_FIELD (ST[7..0]*)
COMMAND_FIELD (CM[7..0]*, CP)
FAILING_ADDRESS (AD[63..0]*)
DATA_HOLD (D[255..64]*, AD[63..0]*)
BYTE_LANE_IN_ERROR

Otros bits de estado a capturar:

VENDOR_DEFINED

El maestro captura el estado de las señales de orden, de las señales de capacidades y las direcciones durante la fase de conexión. Si no hay error se vuelven a capturar al comienzo de la siguiente transacción, pero en caso contrario se mantiene el mismo estado. El esclavo, por su parte capturará el estado de error y retendrá el CSR con el registro de los errores. Las demás señales se capturan en el momento en que se detecta el error.

Los errores internos y aquellos que tienen lugar en un módulo que se encuentra al otro lado de un puente son notificados durante la fase de desconexión utilizando AD[2..0]. El esclavo debe de comprobar estas señales durante la fase de desconexión de una respuesta dividida.

11.2 Errores de Futurebus+

Los síntomas de error dependen de la fase del protocolo paralelo en la que se detecte el error o se notifique por el bus. A continuación vemos una descripción de los errores posibles dependiendo de la fase en la que se produce.

11.2.1 Errores en la fase de conexión

Error de dirección no existente: Este error lo detecta el maestro cuando todos los esclavos desactivan AI* y no son activadas ni la línea BE* ni la SL*. Es un error de protocolo paralelo que el maestro puede identificar mediante las líneas de estado. El registro de error permanece válido hasta que una transacción de bus lo limpia.

EL maestro al detectar el error, inmediatamente procede a ejecutar la fase de desconexión mediante la desactivación de AS*.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos:

ERROR_SUMMARY	MASTER
CON_PHASE	NONEXISTENT_ADD
CA[2..0]*	ST[7..0]*
CM[7..0]*	

mientras que el registro ERROR_LO no tendrá ningún bit activo.

Error de paridad en las direcciones: Este error lo detecta cualquier módulo que decodifique las direcciones y la señal de orden AW* después de que AS* haya sido activada.

Los esclavos que detecten este error no deben activar SL*, ni tampoco BE* en el caso de que no tengan habilitado el bit de PARITY_ERROR_REPORTING_ENABLE. Si por el contrario este bit sí está habilitado se activará BE* en la fase de conexión para dar cuenta del error y también se activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se debe de capturar la dirección del bus y el byte que produjo el error. Además ERROR_HI debe de contener:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Si el módulo que detecta el error es el destino de la transacción, el maestro verá que se ha producido un error de dirección no existente. Si por el contrario es otro módulo no implicado, el maestro no se enterará de nada, aunque la información permanecerá capturada en el CSR del esclavo que detectó el error.

Error de paridad en las órdenes: Este error lo detecta cualquier módulo después de haber activado AS*. La paridad se comprueba en todas las líneas de orden (CM[7..0]*).

Al detectarse el error los esclavos no deben activar SL*, ni tampoco BE* en el caso de que no esté activado el bit PARITY_ERROR_REPORTING_ENABLE. Si por el contrario este bit sí está habilitado se activará BE* en la fase de conexión para dar cuenta del error y también se activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se debe de capturar la dirección del bus. Además ERROR_HI debe de contener:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Si el módulo que detecta el error es el destino de la transacción, el maestro verá que se ha producido un error de dirección no existente. Si por el contrario es otro módulo no implicado,

el maestro no se enterará de nada, aunque la información permanecerá capturada en el CSR del esclavo que detectó el error.

Error de capacidades: El maestro detecta el error cuando el esclavo activa una capacidad no soportada sobre las líneas CA[]* o una combinación no soportada de las señales de estado sobre las líneas ST[]*.

Nada más que el maestro detecta el error pasa a la fase de desconexión, aunque este error no es señalizado sobre el bus en una línea de estado. (ni BE* ni TE* se activan). Si por el contrario es el esclavo el que lo detecta, activará la línea BE* y también activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI, en el caso de que haya sido el maestro el que haya detectado el error deben estar activos los bits de:

ERROR_SUMMARY	MASTER
CON_PHASE	PROTOCOL_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

Si por el contrario es el esclavo el que detecta el error, el registro del CSR del maestro ERROR_HI contendrá:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo, el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	CON_PHASE
PROTOCOL_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de operación ilegal: El esclavo detecta el error mediante la decodificación de las líneas de orden después de que AS* haya sido activado. Dependiendo del contenido de las líneas de dirección y de capacidades se producirá el error o no.

El esclavo que detecta el error activará la línea BE* durante la fase de conexión y notificará su existencia activando TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	CON_PHASE
PROTOCOL_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de temporización en una transacción: Antes o al comenzar una transacción, el maestro debe de inicializar el temporizador de la transacción. Un error de este tipo se detecta cuando el temporizador termina antes de que finalice la fase de desconexión. La duración del temporizador por defecto es de 128μs ± 10%, sin embargo se puede cambiar escribiendo en el registro del CSR TRANSACTION_TIMEOUT.

Al detectar el error, el maestro debe de activar RE* para producir una inicialización del bus. En algunos sistemas se entiende este error como un error fatal, por lo que se producirá una inicialización (reset) del sistema.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
CON_PHASE	TRANSACTION_TIMEOUT
CA[2..0]*	ST[7..0]*
CM[7..0]*	

Error de número de reintentos por ocupado excedidos: El número de reintentos por ocupado ha excedido el valor del campo RETRY_TRESHOLD del registro del CSR BUSY_RETRY_COUNTER.

Este error no se notifica en ninguna señal de estado del protocolo paralelo.

En los registros del maestro se capturará la dirección de destino, en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER

y en el registro del CSR ERROR_LO deben estar activo el bit de:

BUSY_RETRY_TRESHOLD_EXCEEDED

Error de número de reintentos por error excedidos: El número de reintentos por error ha excedido el valor del campo RETRY_TRESHOLD del registro del CSR ERROR_RETRY_COUNTER.

Este error no se notifica en ninguna señal de estado del protocolo paralelo.

En los registros del maestro se capturará la dirección de destino, en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER

y en el registro del CSR ERROR_LO deben estar activo el bit de:

ERROR_RETRY_TRESHOLD_EXCEEDED

En los registros del esclavo debe de haber la suficiente información para averiguar cual fue el error que hizo que fallara el último reintento.

Error de paso: Se produce cuando el maestro detecta la señal BE* activada en la fase de conexión, significando que otro módulo en el sistema notifica que ha detectado un error en esta fase.

El maestro al detectar BE* entrará en la fase de desconexión mediante la desactivación de AS*.

En los registros del maestro se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En los registros del esclavo debe de haber la suficiente información para averiguar por qué se generó este error de paso.

Error de respuesta dividida no esperada o de ID de transacción no existente: Este error se produce bien porque existe una respuesta a una transacción dividida no solicitada (el identificativo de la transacción coincide pero no existe tal transacción dividida o se ha terminado el tiempo para la respuesta) o bien existe un error en el identificativo de la transacción (existen transacciones divididas pendientes, pero el identificativo no coincide con ninguna de ellas).

La notificación del error es opcional puesto que este no es un error de protocolo paralelo del Futurebus+.

En los registros del esclavo se capturará la dirección de la transacción que se marcó como no existente, en el registro del CSR ERROR_HI debe de estar activos el bit de:

ERROR_SUMMARY

y en el registro del CSR ERROR_LO debe de estar activo el bit de:

NONEXISTENT_TRANSACTION_ID

El esclavo (peticionario) es el que da cuenta del error.

Error de ancho de datos equivocado en una transferencia dividida: El módulo que responde a una transacción de lectura dividida usa un ancho de datos diferente del que se especificó en el campo de orden de la transacción de petición.

El esclavo (peticionario inicial) que detecta el error activará BE* en la fase de conexión y también dará cuenta del error activando TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En el registro del CSR del esclavo ERROR_HI se activarán los bits de:

ERROR_SUMMARY	PROTOCOL_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

11.2.2 Errores en la fase de datos

Error de paridad en los datos: En las transacciones de lectura, el maestro detecta el error cuando se produce un error de paridad en cualquiera de las líneas AD[]* o D[]* en la fase de datos. Si la transacción es de escritura, será el esclavo el que detecte el error. La paridad se comprueba sólo en los bytes válidos y únicamente de los datos que se reciben.

El maestro al detectar el error debe pasar a la fase de desconexión mediante la desactivación de AS*. Si es el esclavo el que detecta el error, si no tiene activada la opción de notificación del mismo, puede activar ED* o continuar con los datos erróneos. Si por el contrario si que está activada esta opción, activará BE* en la fase de conexión para dar cuenta del error y también se activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y el byte que produjo el error. Además en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DATA_PHASE	DATA/ADDR_PARITY_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

En el esclavo se debe de capturar la dirección del bus y el byte que produjo el error. Además ERROR_HI debe de contener:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de paridad en las órdenes: Este error se produce si existe un error de paridad durante la fase de datos en las líneas de orden (CM[7..0]*). Aunque estas líneas sólo son significativas durante el primer paso de la fase de datos, la paridad debe de ser correcta durante toda la fase.

Cuando el esclavo detecta el error, si no esté activado el bit de PARITY_ERROR_REPORTING_ENABLE no dará cuenta de él. En caso contrario activará BE* en el paso de datos actual para dar cuenta del error y también se activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se debe de capturar la dirección del bus. Además ERROR_HI debe de contener:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de operación ilegal o error de capacidades: El esclavo que detecta el error activará la línea BE* durante la fase de conexión y notificará su existencia activando TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	PROTOCOL_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de temporización en una transacción: Antes o al comenzar una transacción, el maestro debe de inicializar el temporizador de la transacción. Un error de este tipo se detecta cuando el temporizador termina antes de que finalice la fase de desconexión. La duración del temporizador por defecto es de 128μs ± 10%, sin embargo se puede cambiar escribiendo en el registro del CSR TRANSACTION_TIMEOUT.

Al detectar el error, el maestro debe de activar RE* para producir una inicialización del bus. En algunos sistemas se entiende este error como un error fatal, por lo que se producirá una inicialización (reset) del sistema.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DATA_PHASE	TRANSACTION_TIMEOUT
CA[2..0]*	ST[7..0]*
CM[7..0]*	

Error de paso: Se produce cuando el maestro detecta la señal BE* activada en la fase de datos, significando que otro módulo en el sistema notifica que ha detectado un error en esta fase.

El maestro al detectar BE* entrará en la fase de desconexión mediante la desactivación de AS*.

En los registros del maestro se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En los registros del esclavo debe de haber la suficiente información para averiguar por qué se generó este error de paso.

11.2.3 Errores en la fase de desconexión

Error de paridad en los datos: Los módulos detectan el error cuando se produce un error de paridad en cualquiera de las líneas AD[]* durante la fase de desconexión. Si la transacción es de lectura no se debe de comprobar la paridad en los datos desconectados, puesto que estas líneas permanecen en un estado indefinido en la fase de desconexión que le sigue a una transacción de lectura conectada.

El esclavo el que detecta el error, si no tiene activada la opción de notificación, no dará cuenta de él. Si por el contrario si que está activada esta opción, activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino. Además en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se debe de capturar la dirección del bus y además ERROR_HI debe de tener activos los bits de:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de paridad en las órdenes: Este error se produce si existe un error de paridad durante la fase de desconexión en las líneas de orden (CM[7..0]*).

Cuando el esclavo detecta el error, si no esté activado el bit de PARITY_ERROR_REPORTING_ENABLE no dará cuenta de él. En caso contrario activará TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se deben capturar los datos de la desconexión. Además ERROR_HI debe de contener:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error de temporización en una transacción: Antes o al comenzar una transacción, el maestro debe de inicializar el temporizador de la transacción. Un error de este tipo se detecta cuando el temporizador termina antes de que finalice la fase de desconexión. La duración del temporizador por defecto es de 128μs ± 10%, sin embargo se puede cambiar escribiendo en el registro del CSR TRANSACTION_TIMEOUT.

Al detectar el error, el maestro debe de activar RE* para producir una inicialización del bus. En algunos sistemas se entiende este error como un error fatal, por lo que se producirá una inicialización (reset) del sistema.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	TRANSACTION_TIMEOUT
CA[2..0]*	ST[7..0]*
CM[7..0]*	

Error de longitud en modo empaquetado: Se produce el error cuando no se recibe el número de datos esperado en modo empaquetado.

El esclavo notificará el error activando TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

Si el maestro detecta el error, en sus registros se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	LENGTH_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

Si es el esclavo el que detecta el error, en sus registros se capturará los datos de desconexión y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	DISCON_PHASE
LENGTH_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

Error interno o retransmitido: Cuando ocurre un error interno en un esclavo, o cuando se produce un error en una transacción dividida al otro lado de un puente de bus, si se traducen

en la corrupción de datos, deben ser transmitidos al bus. Entre este tipo de errores podemos tener: un error de dirección no existente, un error de conflicto o un error de transacción.

En las transacciones conectadas el esclavo debe de notificar el error activando TE* hasta el final de la fase de conexión de la siguiente transacción de bus. Si es una transacción dividida el maestro lo notifica mediante las líneas AD[2..0]* en la fase de desconexión de una transacción de respuesta dividida.

En las transacciones conectadas, en los registros del maestro se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo se capturará la dirección de destino y el registro del CSR ERROR_HI contendrá:

ERROR_SUMMARY	UNIT_ESPECIFIC_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En las transacciones divididas, el maestro (el que responde a la transacción dividida) debe de capturar la dirección de destino y los datos de la desconexión que muestran el tipo de error. Además el registro del CSR ERROR_HI contendrá activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	UNIT_ESPECIFIC_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

El esclavo (peticionario original) no debe de mostrar ningún error en el registro del CSR ERROR_HI o bien contendrá activos los bits de:

ERROR_SUMMARY	DISCON_PHASE
CA[2..0]*	ST[7..0]*
CM[7..0]*	

Error de transacción: Cuando el maestro detecta activada la línea TE* en la fase de desconexión indica que otro módulo en el sistema ha detectado un error. Deberá entonces notificar el error enviando una interrupción.

En los registros del maestro se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el esclavo, los registros de error del CSR deben contener la suficiente información para indicar por qué se activó la línea TE*.

Error de exceso de datos de respuesta: Se produce el error cuando el módulo que responde a una transacción de respuesta divida devuelve más datos de los que se especifican en la fase de desconexión de la transacción.

Para este error no se produce ninguna notificación hardware, sino que el esclavo debe de activar ED* para terminar con la transacción. Por tanto no es posible diagnosticar el error.

En los registros del esclavo se capturará la dirección de destino y en el registro del CSR ERROR_HI deben estar activos los bits de:

ERROR_SUMMARY	PROTOCOL_ERROR
CA[2..1]*	ST[7..0]*
CM[7..0]*	

Error de operación ilegal: El esclavo detecta el error cuando en una transacción dividida, mediante los datos de la desconexión se le indica que no hay restricción de longitud de datos (DL = 0).

El esclavo notificará la existencia del error activando TE* hasta el final de la fase de conexión de la siguiente transacción de bus.

Tanto en los registros del maestro como en los del esclavo se capturará la dirección que ha fallado y los datos de la desconexión (DATA_HOLD CSR).

En el registro del CSR ERROR_HI del esclavo estarán activos los bits:

ERROR_SUMMARY	DISCON_PHASE
PROTOCOL_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

En el registro del CSR ERROR_HI del maestro estarán activos los bits:

ERROR_SUMMARY	DISCON_PHASE
CA[2..0]*	ST[7..0]*
CM[7..0]*	

11.2.4 Otro tipo de errores

Error de temporización en una transacción dividida: Se produce este error cuando esperando la respuesta de una transacción dividida se sobrepasa el valor almacenado en el registro de SPLIT_TIMEOUT.

Este error no es notificado mediante ninguna señal de estado del protocolo paralelo.

En el maestro, en el registro del CSR ERROR_HI estarán activos los bits:

ERROR_SUMMARY	MASTER
SPLIT_TIMEOUT	

Errores de arbitraje del bus: Estos errores de arbitraje funcionan de forma asíncrona con respecto a las transacciones que se producen en el bus de protocolo paralelo. Los dos tipos posibles de error son:

- Error de temporización en el arbitraje
- Error de comparación o de paridad en el arbitraje

Si se detecta el error en el bus de arbitraje se activan las señales AC1* y AC0* en la fase 3 de la competición.

En el maestro, en el registro del CSR ERROR_HI estarán activos los bits:

ERROR_SUMMARY MASTER

y en el registro del CSR ERROR_LO estarán activos bien los bits:

ARB_ERR ARB_COMPARISON_OR_PARITY_ERR

bien los bits

ARB_ERR ARB_TIMEOUT_ERR

12 *Extensiones de Tolerancia a Fallos del Futurebus+*

12.1 Introducción

El IEEE 896.9 describe las extensiones de tolerancia a fallos del Futurebus+, que son en todo momento compatibles con los protocolos usados. El objetivo principal de la especificación es asegurar que un sistema basado en Futurebus+ que necesita ser tolerante a fallos puede ser implementado con un impacto mínimo en el rendimiento y en el coste. Los objetivos que se deben cumplir son los siguientes:

- Ningún dato erróneo se transmita por el bus sin ser detectado
- Detectar como mínimo los errores de un único bit
- Los errores deben ser registrados y notificados
- El bus tiene la capacidad de detenerse si se detecta un fallo

En este estándar se describen los mecanismos de bajo nivel utilizados para la detección de fallos, la recuperación y la notificación de los errores de FB+ al nivel superior y un rango de aplicaciones tolerantes a fallos: fallo silencioso (*fail-silent*), fallo seguro (*fail-safe*) y fallo operativo (*fail-operate*).

- **Fail-stop:** Este modelo representa a un sistema tolerante a fallos en donde se debe de proteger la información para que no se corrompa debido a un error. Es necesario por tanto la detección de fallos, la notificación del error y un método para detener el sistema en un estado definido.

- **Fail-safe:** Este modelo representa a los sistemas en donde los fallos pueden ser aceptados sólo si son benignos. Un fallo benigno se define como aquel que produce unas consecuencias

del mismo orden de magnitud (en términos de coste) que los beneficios producidos por el servicio producido en la ausencia del fallo. El término de *seguro* se refiere al hecho de que el sistema falla de tal manera que evita las consecuencias catastróficas en el mismo sistema o en su entorno. Se definen dos tipos de sistemas seguros: los *Fail-Safe-Stop* en donde al detectar el fallo se accionan los mecanismos necesarios para comprobar que el sistema está en un estado que no produce ninguna catástrofe y entonces lo paran y los *Fail-Safe-Continue* que difieren de los anteriores en que al asegurarse de que el sistema está en un estado que no produce ninguna catástrofe, se produce la reconfiguración del mismo y entonces se continúa con la operación normal.

Estos sistemas cumplen con los requisitos de los Fail-Stop puesto que la información debe estar protegida para asegurar que el sistema termina en un estado seguro. Este modelo tiene las mismas necesidades de detección, recuperación y de notificación de error que el modelo anterior, pero requiere la presencia de un bus alternativo de acceso a los registros del CSR.

- **Fail-Operate:** Este sistema debe de funcionar aún con la presencia de errores. Esto se puede lograr mediante mecanismos hardware o software, dependiendo de las necesidades de rendimiento del sistema. Las necesidades de fiabilidad del sistema determinarán cuantos errores pueden ser tolerados y cuanta redundancia se necesita para cada elemento. La fiabilidad depende especialmente del entorno y del tiempo de misión.

  Estos sistemas cumplen con los requisitos de los dos anteriores ya que los fallos se enmascaran y el sistema continua su operación como si el fallo no hubiera ocurrido (con la excepción de las tareas realizadas en segundo plano para el aislamiento del fallo y la recuperación). Aparte de la necesidad de un bus alternativo de acceso a los CSRs se necesitan recursos replicados para soportar el enmascaramiento de los fallos.

12.2 Detección de errores

La especificación base de Futurebus+ incluye muchas características de tolerancia a fallos: paridad de byte, protocolos bien definidos que se pueden monitorizar para ver si funcionan correctamente, registro de los síntomas de error, facilidad de diagnóstico, reintento e inserción en vivo. Sin embargo estas características aún no son suficientes para implementar un bus FB+ tolerante a fallos. Es necesario para ello que se sigan los siguientes criterios:

1. La información que se transfiere entre diferentes módulos debe de tener restricciones temporales.
2. Todos los errores simples que ocurren en el bus, incluyendo también los transceivers y los controladores, deben ser detectables y observables.
3. Se debe de suministrar un otro bus como vía de comunicación alternativa.
4. Debe de haber cierta redundancia para poder hacer una recuperación.

No todas las señales de FB+ están protegidas por paridad puesto que múltiples módulos pueden activar bits simultáneos en los campos de estado y de capacidades. En este punto se describen las facilidades añadidas para verificar la integridad de las señales de estado, de capacidades, de dirección geográfica, de reset y de sincronización. Con estas extensiones FB+ permite el diseño de módulos fail-stop, con reparación o enmascaramiento de fallos, a través del reintento o con módulos duplicados. Si la reparación del módulo no es factible la información se hace visible para permitir al sistema operativo reparar o enmascarar los fallos.

Las señales que incorpora este estándar al FB+ se incluyen en la Tabla 8.5.

Líneas de información	
RF*	Reflected
TED*	Transaction Error Duplicate
SLD*	Select Duplicate
Líneas de inicialización/reset	
RED*	Reset Duplicate
Dirección geográfica	
GAP*	Geografical Address Parity

Tabla 8.5: Nuevas señales de la extensión tolerante a fallos de FB+

Mecanismos de detección de fallos

Se debe de incorporar información redundante para proteger las señales activas por nivel. Para ello se introducen líneas de paridad. Para proteger las señales de estado ST[] y de capacidades CA[] se introducen las señales de duplicación y de reflexión. El maestro en la fase de conexión muestrea estas señales y en las siguientes fases de datos, su valor debe de ser reflejado hacia los esclavos. Para ello se utilizan las líneas de órdenes que ya están protegidas por paridad.

La reflexión no cubre todos los fallos posibles para las señales de error de transacción TE* y para la señal de selección SL*. La primera se activa en la fase de desconexión, por lo tanto no puede ser reflejada en la misma transacción y la segunda se activa en todas las transacciones donde exista una dirección válida, por lo tanto no se puede detectar el fallo en la que esté siempre activada. Como consecuencia estas señales, más la señal de reset, deben estar duplicadas (TED*, SLD* y RED*) y funcionar a la vez que sus señales correspondientes. Se trata de una detección de error mediante una redundancia temporal.

Las señales activas por flanco no necesitan ser protegidas por paridad, ya que están estrechamente ligadas unas con otras mediante el protocolo de sincronización y por lo tanto se pueden monitorizar. Para detectar los errores de protocolo se comprobará el estado de ciertas señales de sincronización en el momento en el que se produce una transición de otra señal de sincronización. Comparando estas señales con el estado de alguna señal que controla el módulo se puede detectar el error. De esta manera se pueden detectar errores de protocolo paralelo, de posesión del bus, del arbitraje centralizado y de mensajes arbitrados.

Como FB+ es asíncrono se deben usar temporizadores para detectar las situaciones en las que una señal nunca llega. De este modo se obtiene una protección a un nivel superior. La expiración de un temporizador hace que se recoja el error en uno de los registros del CSR. En la Tabla 8.6 podemos ver cuales son las funciones que están protegidas por este método:

Función protegida	Error
Arbitraje	ARB_TIMEOUT_ERROR
Transacción	TRANSACTION_TIMEOUT
Reset	RESET_TIMEOUT
Transacción dividida	SPLIT_TIMEOUT
Petición de bus	REQUEST_TENURE_TIMEOUT
Mensaje de arbitraje	ARB_MSG_TIMEOUT

Tabla 8.6: Funciones de FB+ protegidas por temporizador de guardia

12.3 Manejo de errores

A continuación se presentan varias tablas con todos los posibles errores que pueden ser detectados. La señal de Local_Report debe de ser activada por el módulo que detecta el error para indicar que se ha entrado en un estado de error. El bit de FTE_ENABLE en el registro MAINT_CONTROL del CSR indica si el módulo debe de realizar la comprobación del error o no. Independientemente del estado de este bit, todos los módulos diseñados con la extensión para tolerancia a fallos (FTE - Fault Tolerant Extension) deben generar las señales duplicadas y las replicadas

Clase de detección de error por paridad en:

las direcciones
la lectura de los datos
la escritura de los datos
los datos de la desconexión
la orden de la fase de conexión
la orden de la fase de datos
la orden de la fase de desconexión
la etiqueta de la fase de conexión
la etiqueta de la fase de lectura de datos
la etiqueta de la fase de escritura de datos
la etiqueta de la fase de desconexión
la dirección geográfica inicial
los mensajes arbitrados

Clase de errores detectados por la reflexión de la señal:

Ocupado
Terminación de datos
Envío múltiple
Bandera de fase de conexión de la transacción
Bandera de fase de datos de la transacción
Intervención en la conexión
Intervención en los datos
Espera
Velocidad de la transacción empaquetada
Transacción forzada
Respuesta dividida

Clase de errores detectados por duplicación:

Fase
Transacción
Seleccionado
Reset

Clase de errores detectados por observación del protocolo de bus:

Por parte del maestro en la fase de conexión
Por parte del maestro en la fase de datos
Por parte del maestro en la transacción de desconexión
Por parte del esclavo participante en la transacción de desconexión
Por parte de otro módulo observador en la transacción de desconexión
Por parte del maestro en el protocolo
Por parte del esclavo en el protocolo
Operación ilegal en la fase de conexión
Operación ilegal en la fase de datos

Operación ilegal en la fase de desconexión
Por parte del maestro en la longitud
Por parte del esclavo en la longitud
Comparación en el arbitraje
Condición de arbitraje
Secuencia lógica de arbitraje
Por parte del módulo en la concesión de la petición
Por parte del árbitro central en la concesión de la petición
Interno al nodo en la fase de datos de una transacción no dividida
Dirección no existente
Más datos divididos de los pedidos
Respuesta dividida inesperada / Identificativo de transacción no existente
Ancho de datos erróneo en una respuesta dividida
Escritura a un CSR de sólo lectura
Escritura a un CSR protegido contra escritura
Por parte del maestro en la secuencia de las direcciones
Por parte del esclavo en la secuencia de las direcciones
Por parte del maestro en la secuencia de los datos
Por parte del esclavo en la secuencia de los datos

Clase de errores detectados por expiración del contador

Excedido el nivel de retransmisiones por busy
Excedido el nivel de retransmisiones por error
Temporización del arbitraje
Temporización de las transacciones divididas
Temporización en la fase de conexión
Temporización en la fase de datos
Temporización en la fase de desconexión
Temporización para la obtención del bus
Temporización en el mensaje de arbitraje
Temporización en el reset

12.4 Notificación del error

Después de la detección, los errores deben ser registrados en los CSR del módulo. La activación del bit de ERROR_SUMARY en el registro ERROR_HI indica que el registro de errores se ha detenido. Para la notificación del error podemos distinguir dos tipos de bits diferentes:

1. Los bits de estado de la cuenta de errores, que al activarse producen la activación de ERROR_SUMARY.
2. Los bits de captura de la cuenta de errores, cuyo estado se congela como resultado de la activación de ERROR_SUMARY.

Los demás bits tienen un valor indefinido cuando se produce un error. Cuando se desactiva el bit de ERROR_SUMARY debido a la desactivación de los bits de estado de la cuenta de errores, dicha cuenta ya no está bloqueada y puede recoger nuevos estados.

Los bits de estado de la cuenta de errores son campos específicos en los registros ERROR_HI y ERROR_LO e identifican cada uno de los errores definidos en el punto anterior.

Cualquier módulo esclavo indicará la detección de un error al FB+ mediante la activación y el mantenimiento de BE* una vez se detecta el error hasta la fase de desconexión. En este caso, el módulo esclavo que participa en la transacción puede activar TE* y TED* hasta la siguiente fase de conexión. Cualquier esclavo que detecte un error de reflexión puede indicarlo mediante la activación

de la señal de error de transacción y su duplicado (TE* y TED*) antes del final de la fase de desconexión de la transacción en la que se observó el error.

Existen tres errores especiales que puede detectar el maestro: Error de paridad en los datos de lectura, error de secuencia de datos y error de longitud. Debe de notificarlos activando TE*/TED* para indicar a los esclavos participantes que la transacción no ha terminado con éxito. Esto previene errores de consistencia de los datos en las caches de los esclavos participantes en la transacción, haciendo que los subsiguientes reintentos todavía necesiten que los esclavos suministren los datos o que los recojan.

12.5 Recuperación frente al error

Los detalles de la recuperación frente al error, notificado a través de las señales TE*/TED*, pueden verse en la Figura 8.8, e incluyen:

- Recuperación de la transacción actual
- Mecanismos de reintento de las transacciones
- Mecanismos configurables de contención de errores

Cuando el maestro detecta el error, el módulo registra el error, normalmente activa TE*/TED*, procede a la fase de desconexión de la transacción o reinicializa el bus mediante un BUS_INIT, y bien intenta retransmitir la transacción si puede, bien deja de ser maestro de bus para cualquier transacción posterior.

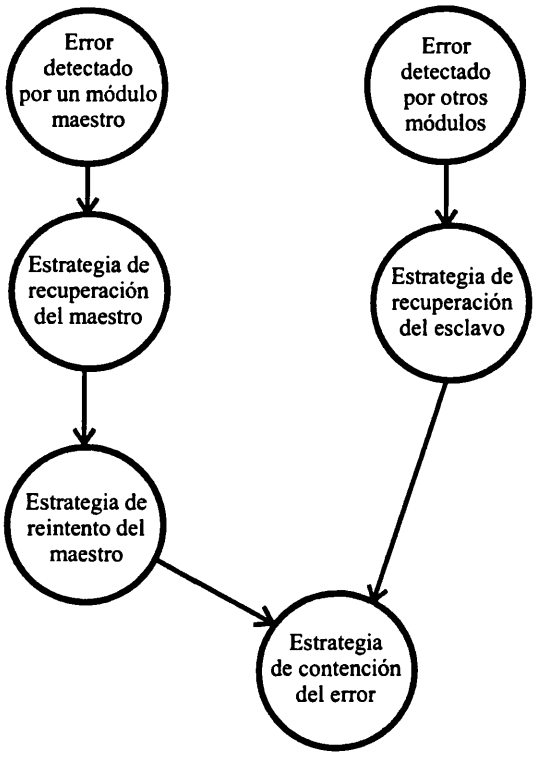


Figura 8.8: Mecanismos de recuperación

Cuando por el contrario es el esclavo el que detecta el error, el módulo registra el error, activa TE* (y posiblemente BE*), termina con los protocolos hasta la fase de conexión de la siguiente transacción y la termina. Los datos no se considerarán válidos hasta que la transacción se termine con éxito.

Anexo III

En este anexo se recogen los acrónimos, abreviaturas y términos ingleses que mayormente se han utilizado durante la memoria de investigación. Se incluye una pequeña descripción de su significado para ponerlo dentro del contexto de la tolerancia a fallos.

AE (Aplication Execution): Define parte del software implementado en el módulo de proceso que se encarga de ejecutar la aplicación del usuario.

AHDL (Altera Hardware Description Languaje): Es un lenguaje de descripción de hardware, muy parecido al VHDL que se utiliza para definir la lógica de los circuitos programables de Altera.

ATPG (Automated Test Pattern Generation): Son los programas de ordenador que toman como datos las especificaciones de un circuito digital y generan de forma automática los patrones de test de ese circuito.

Backoff: Ciclo especial del procesador 486 que consiste en repetir el ciclo anterior.

Backplane: Parte de la estructura física de un bus donde se encuentran los conectores.

Boundary Scan: Técnica de comprobación periférica que incorporan algunos integrados que permite, a través del puerto de acceso para test, comprobar su funcionamiento.

Bus Bridge: Dispositivo que hace de interface entre dos buses, que pueden ser idénticos o no.

CSR (Command and Status Registers): Registros de estado y de órdenes definidos en Futurebus+.

CSR Bus: Bus que conecta al CSR con los procesadores y el Host Bus.

Dependability: Garantía de funcionamiento de un sistema tolerante a fallos

DPU (Data Procesing Unit ó Dual Procesing Unit): Módulo de procesamiento de datos dual, en donde existen elementos replicados que trabajan al unísono.

Drivers: Dispositivos especiales que conectan las señales de los módulos al bus del sistema.

E/S: Hace referencia a todos las operaciones o circuitos que tienen que ver con la Entrada o la Salida de un sistema informático.

EPLD (Erasable Programmable Logic Device): Dispositivos reprogramables basados PALs con registros.

ESM (Even-driven State Machine): Parte del software implementado en los módulos de proceso del sistema FASST que hace de interfaz entre el software y el hardware. Es una máquina de estados dirigida por eventos.

Fail-stop: Propiedad de algunos sistemas tolerantes a fallos por la cual cuando se detecta la avería se detienen sin contaminar con datos erróneos otras partes del sistema.

FASST (Fault-tolerant Architecture with Stable Storage Technology): Acrónimo del Proyecto ESPRIT P-5212. Arquitectura tolerante a fallos con tecnología de almacenamiento estable.

FB+: Acrónimo del bus Futurebus+ definido en el estándar del IEEE 896

Graceful degradation: Reconfiguración del sistema en presencia de averías sin necesidad de pararlo y a costa de perder prestaciones.

HB: Abreviación de Host Bus.

Host Bus: Parte del bus definido en Futurebus+ que conecta a los procesadores con la memoria local y con los integrados de Futurebus+.

I'm alive: Mensajes que envían los módulos de proceso de un multiprocesador para indicar a las demás placas que están funcionando de forma correcta.

JTAG (Join Test Action Group): Comité encargado de definir el estándar de comprobación periférica.

LMC (Local Memory Controler): Parte de un dispositivo programable que maneja la memoria local.

Lock-step: Modo de funcionamiento de los elementos duplicados que se caracteriza porque funcionan a la vez, gobernados por la misma señal de reloj y con las mismas señales de entrada.

MTBF (Mean Time Between Failures): Tiempo medio que transcurre entre dos averías sucesivas de un sistema.

MTTF (Mean Time To Failure): Tiempo medio que transcurre desde que empieza a funcionar un dispositivo y se produce la primera avería.

MTTR (Mean Time To Repair): Tiempo medio que se tarda en reparar un sistema.

Off-line: Fuera de línea, es decir, con el sistema apagado o sin realizar las tareas normales.

On-line: En línea. La actividad que se realiza en línea se desarrolla en paralelo con las aplicaciones normales que ejecuta el sistema cuando está funcionando.

PCB (Printed Circuit Board): Placa de circuito impreso.

Pegado-a: Modelo de fallos que se utiliza en la inyección que se basa en poner en una línea un estado lógico alto o bajo de forma permanente (pegado-a-0 ó pegado-a-1).

Pinout: Descripción del significado de cada pata de un circuito integrado.

Rack: Carcasa donde se encuentran los conectores de un bus y a veces la fuente de alimentación.

RTC (Real Time Clock): Dispositivo que mantiene la hora del día mediante una batería.

SD (Stable Disk): Disco estable que se utilizó en el sistema FASST.

SEC-DED (Single error Detection - Double Error Correction): Códigos redundantes por los que se codifican los datos de forma que detectan todos los errores de 2 bits y además corrigen los errores de 1 bit.

Steady-State: Estado estacionario en una Cadena de Markov. En este estado el tiempo es infinito.

STM (Stable Transactional Memory): Memoria de almacenamiento estable que se utilizó en el sistema FASST.

Tasa de Fallos: Se define como el número de fallos que tienen lugar por unidad de tiempo. Se representa con la letra griega lambda.

TFB2002 DPU (Data Processing Unit): Integrado de la familia de Futurebus+ que procesa los datos.

TFB2010 ABC (Arbitration Bus Controller): Integrado de la familia de Futurebus+ que implementa un árbitro distribuido.

Transceivers: Dispositivos similares a los drivers pero que manejan señales en ambos sentidos.

VLSI (Very Large Scale Integration): Son los circuitos integrados con un número de transistores muy elevados.

Bibliografía

[1] G. Fabregat. "Diseño de un Módulo de Proceso Tolerante a Fallos. Inclusión en una Arquitectura Multiprocesador". Tesis Doctoral. Departamento de Informática y Electrónica. Universitat de València. Febrero 1996.

[2] V. Cerverón. "*Algoritmos de Soporte para Tolerancia a Fallos. Aplicación a la Comunicación Punto a Punto*". Tesis Doctoral. Departamento de Informática y Electrónica. Universitat de València. 1996.

[3] 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 de 1992.

[4] W. Carter, W. Bouricius. "*A survey of fault tolerant computer architecture and its evaluation*". Computer, Vol. 4, No. 1, Enero 1971, pag 9-16.

[5] Engineer Research Association. "*High Speed Computing Devices*". McGraw-Hill, 1950.

[6] P. Gil. "Garantía de funcionamiento: Conceptos básicos y terminología". Informe interno. DISCA. UPV. Mayo 1996.

[7] J. Laprie. "*Dependability: basics concepts and associated terminology*". Internal Report Nº 90055. LAAS: Laboratoire d'Automatique et d'Analyse des Systemes, Tolouse Cedex. March 1990.

[8] M. Weik. "*A survey of domestic electronic digital computing systems*". Report Nº 971. Comerce Department. Ballistic Research Laboratories, Aberdeen Providing Ground. Ms. December 1955.

[9] M. Banâtre, G. Muller, B. Rochat and P. Sanchez; "Design Decisions for the FTM: A General Purpose Fault-Tolerant Machine."; Proceedings of the 21st Fault-Tolerant Computing Symposium; June 1991.

[10] P.A. Lee, T. Anderson, "Fault Tolerance. Principles and Practice", Second Edition, Springer-Verlag Wirn New York, 1990.

[11] B. Coghlan and J. Jones; "The Stable Disk - An Introduction."; FASST Project (ESPRIT P5212) Deliverable; September 1992.

[12] M. Banâtre and P. Jouber. "Cache Management in a Tightly Coupled Fault-Tolerant Multiprocessor." Proceedings of the 20th Fault-tolerant Computing Symposium. June 1990.

[13] R. E. Ahmed, R. C. Frazier and P. N. Marinos; "Cache-Aided Rollback Error Recovery (CARER) Algoritms for Shared-Memory Multi-processors."; Proceedings of the 20th Fault-Tolerant Computing Symposium; June 1990.

[14] B. W. Johnson. "Fault-tolerant Microprocessor-based Systems". IEEE MICRO. December 1984.

[15] J. C. Laprie, J. Arlat, C. B'eounes and K. Kanoun. "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures". IEEE Computer. July 1990.

[16] R. D. Schlichting and F. B. Schneider; "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems."; ACM Transactions on Computing Systems; August 1983.

[17] R. Martínez, C. Pérez, F. Pardo, J. Boluda, "A fail-stop 486 processor module: The design of the comparators", Aplicaciones en computación y telemática avanzadas, ISBN 84-370-1416-6. Universidad de Valencia, 1993.

[18] IEEE 896.1 "Futurebus+ Logical Layer Specification", 1991

[19] IEEE Std 896.2, "Futurebus+ Physical Layer and Profile Specifications", 1991

[20] "Futurebus+. P896.3: System Design Guide". IEEE, 1990

[21] IEEE P896.3, "Futurebus+ Recommended Practices", Draft No. 4, Jan. 7, 1992

[22] IEEE Std 1212, "Control and Status Register (CSR) Architecture for Microcomputer Buses", 1991

[23] R. Balakrishnan. "*The Proposed IEEE 896 Futurebus-A Solution to the Bus Driving Problem*". IEEE Micro. August 1984. Pág 23-27.

[24] Lui Sha, Ragunathan Rajkumar, John P.Lehoczky, "Real-time Computing with IEEE Futurebus+", IEEE Micro, June 1991, pp 23-100

[25] Victor P.Nelson, Auburn University, "Fault-Tolerant Computing: Fundamental Concepts", Computer, July 1990, pp 19-25

[26] F.Pardo, J.A.Boluda, R.Martínez, and C.Pérez, "Futurebus+ aplicación en sistemas tolerantes a fallos", Mundo Electrónico, pp 44-49, March 1994

[27] "High-Performance I/O Bus Architecture: a Handbook for IEEE Futurebus+ Profile B". IEEE Standards Press. 1994

[28] G. Fabregat, C. Pérez, J.A. Boluda, R.J. Martínez and R. Munt; "The FASST DPU Technical Description."; FASST Project (ESPRIT P5212) Internal Report; July 1994.

[29] R.J. Martínez, C. Pérez, G. Fabregat; "The FASST comparators."; FASST Project (ESPRIT P5212) Internal Report; May 1993.

[30] Texas Instruments Inc.; "Futurebus+ Interface Family. Protocol, Arbitration and Backplane Transceiver Logic."; Data Manual; March 1994.

[31] P. A. Bernstein; "Sequoia: A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing."; IEEE Computer; February 1988.

[32] FASST Consortium; "FASST: Fault Tolerant Architecture with Stable Storage Technology."; FASST Project (ESPRIT P5212) Technical Annex; 1990.

[33] GNU's gdb (v.4.10) texinfo on-line help.

[34] "GNU General Public License.", GNU's gdb (v.4.10) documentation.

[35] J. Ousterhout, F. Douglis. "*Beating the I/O Bottleneck: A case for Log-Structured File Systems*". Internal Report UCB/SD 88/467. University of California.

[36] P. Gil, V. Santonja, R. Ors, J.J. Serrano. "*Validación teórica de Sistemas Tolerantes a Fallos*". DISCA. Servicio de publicaciones Universidad Politécnica de Valencia. SPUPV-94.482.

[37] B. Johnson. "*Design and Analysis of Fault-Tolerant Digital Systems*". Addison-Wesley Publishing Company, Reading, Mass., 1989.

[38] T. Arnold. "*The concept of coverage and its effect on reliability model of a repairable system*". IEEE Transactions on Computing, vol. C-22. Marzo 1973.

[39] D. Siewiorek y R. Softwareartz. "*The Theory and Practice of Reliable System Design*". Digital Press, 1982.

[40] W. Bouricius, W. Carter y P. Schneider. "*Reliability modeling techniques for self-repairing computer sytems*". Proceedings of the 24th ACM Annual Conference. 1969.

[41] D. Pradhan. "*Fault-Tolerant Computer System Design*". Prentice-Hall Ptr. 1996.

[42] R. Sahner, K. Trivedi. "*SHARPE: Symbolic Hierarchical Automated Reliability and Performance Evaluator*". Feb 1992.

[43] J. Dugan, K. Trivedi. "*Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems*". IEEE Transactions on Computers. Jun 1989.

[44] K. Trivedi, R. Geist, M. Smotherman and J. Dugan. "*Hybrid modeling of fault-tolerant systems*". Computer Electronics Engineering, vol 11, no. 2 y 3. 1985.

[45] R. Conn, P. Merryman, K. Whitelaw. "CAST - A complementary analytic-simulative technique for modeling fault-tolerant computing systems". Proc. AIAA Computer Aerospacial Conference. Los Angeles, California. Nov. 1977.

[46] J. Dugan, K. Trivedi, R. Geist y V. Nicola. "*Extended stochastic Petri nets: Applications and analysis*", Performance '84. E. Gelembe. Ed. Amsterdam: North Holland, 1984.

[47] S. Makam, A. Avizienis. "*ARIES 81: A reliability and lifecycle evaluation tool for fault-tolerant systems*". Proceedings of the 12nd Fault-Tolerant Computer Symposium. June 1982.

[48] J. Stiffler, L. Bryant. "*CARE III phase III report-Mathematical description*". Contract Report 3566, NASA, Nov. 1982. Fault Tolerant Computing Simposium, Kyoto, Japan. October 1980.

[49] J. McGough. "*Effects of near-coincident faults in multiprocessor systems*". Proceedings of the 5th IEEE/AIAA Digital Avionics Systems Conference. Nov 1983.

[50] D. Siewiorek y R. Softwareartz. "*Reliable Computer Systems. Design and Evaluation*". Digital Press, 1992.

[51] M. Shooman. "*Operational Testing and Software Reliability Estimation During Program Development*". IEEE Symposium on Computer Software Reliability, 1973.

[52] Z. Jelinsky y P. Moranda. "*Applications of a Probability Based Method to a Code Heading Experiment*". Proceedings IEEE Symposium Computer Software Reliability, 1973.

[53] R. Wolverton y G. Shick. "*Assessment of Software Reliability*". TRW-SS-73-04. Los Angeles, CA. 1974.

[54] T. Thayer, M. Lipow y E. Nelson. "*Software Reliability*". New York: North-Holland, 1978.

[55] B. Littlewood. "*A reliability Model for Markov Structured Software*". IEEE Conference on Reliable Software. 1975.

[56] A. Trivedi y M. Shooman. "*A Many State Markov Model for the Estimation and Prediction of Computer Software Performance*". Proceedings of the International Conference on Reliable Software. IEEE Computer Society. 1975.

[57] N. Schneidewind. "*Analysis of Error Processes in Computer Software*". Proceedings of the International Conference on Reliable Software. IEEE Computer Society. 1975.

[58] E. Nelson. "*A Statistical Basis for Software Reliability Assessment*". TRW, 1973.

[59] A. Fitzsimmons y T. Love. "*A Review and Evaluation of Software Science*". ACM Computing Surveys 10. March 1978.

[60] A. Costes, C. Landrault y J. Laprie. "*Reliability and Availability Models for Maintained Systems Featuring Hardware Failures and Design Faults*". IEEE Transactions on Computers. June 1978.

[61] United States Department of Defense. "*Military Standardization Handbook: Reliability Prediction of Electronic Equipment*". MIL-HDBK-217, 1965.

[62] D.L. Snyder. "*Random Point Processes*". New York. Wiley, 1975.

[63] C. Gear. "*The Automatic Integration of Stiff Ordinary Differential Equations*". Information Processing '68. North Holland, Amsterdam, 1969.

[64] M. Silva. "*Las Redes de Petri en la Automática y la Informática*". Editorial AC. Madrid. 1988.

[65] J. Dugan, V. Nicola, R. Geist, K. Trivedi. "*Extended Stochastic Petri Nets: Applications and Analysis*". Performance '84. North-Holland. Amsterdam, 1985.

[66] S. Bavuso, P. Peterson, D. Rose. "*Care III Model Overview and User's Guide*". NASA Technical Memorandum Number 85810. June, 1984.

[67] M. Hsueh, T. Tsai, R Iyer. "*Fault Injection. Techniques and Tools*". Computer. IEEE. April 1997.

[68] R. Iyer. "*Experimental Evaluation*". Fault Tolerant Computing. Special Issue. FTCS-25 Silver Jubilee. California. June 1995.

[69] J. Melsa, D. Cohen. "*Decision and Estimation Theory*". New York. McGraw-Hill, 1978.

[70] M. Cukier, J. Arlar, D. Powell. "*Frequentist and Bayesian Coverage Estinations for Stratified Fault-Injection*". Six International Working Conference on Dependable Computing for Critical Applications. Grainau, Germany. March 1997.

[71] I. Miller, J. Freund. "*Probability and Statistics for Engineers*". Englewood Cliffs, New York. Prentice-Hall, 1965.

[72] R. Saleh, A. Newton. "*Mixed-Mode Simulation*". Kluwer Academic Publishers, 1990.

[73] F. Yang. "*Simulation of Faults Causing Analog Behavior in Digital Circuits*". Ph. D. diss., Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign. May, 1992.

[74] W. Rogers, J. Abraham. "*CHIEFS: A Concurrent Hierarchical and Extensible Fault Simulator*". Proceedings of the IEEE International Test Conference, 1985. Pag 710-716.

[75] G. Choi, R. Iyer, V. Carreno. "*Fault Behavior Dictionary for Simulation of Device-Level Transients*". Proceedings of the International Conference on Computer Aided Design. November, 1993.

[76] G. Ries, R. Iyer. "*Evaluating the Impact of Transient Faults on Software Behavior: Case Study of a Commercial High-Speed Network*". Six International Working Conference on Dependable Computing for Critical Applications. Grainau, Germany. March 1997.

[77] A. Dupuy, J. Schardwareartz, Y. Yemini, D. Bacon. "*NEST: A network Simulation and Prototyping Testbed*". Comunnications of the ACM, vol. 33, nº 10. October 1990. Pág 64-74.

[78] K. Gosoftwareami, R. Iyer. "*DEPEND: A Simulation-Based Environtment for System Level Dependability Analisys*". IEEE Transactions on Computers. Vol 46, Nº 1. January 1997.

[79] J. Clark, D. Pradhan. "*REACT: A Synthesis and Evaluation Tool for Fault-Tolerant Multiprocesor Architectures*". Proceedings of the Annual Reliability and Maintainability Symposium. January 1993, Atlanta, Georgia. Pág 428-435.

[80] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson. "*Fault Injection into VHDL Models: The MEFISTO Tool*". Proceedings of the 24th International Fault-Tolerant Computing Symposium. 1994.

[81] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. Fabre, J. Laprie, E. Martins, D. Powel. "*Fault Inyection for Dependability Validation: A Methodology and Some Applications*". IEEE Transactions on Software Enginerring, vol. 16, nº 2. February 1990. Pág 166-182.

[82] C. Yount, D. Siewiorek. "*A Methodology for the Rapid Injection of Transient Hardware Errors*". IEEE Transactions on Computers, vol. 45, nº 8. August 1996. Pág 881-891.

[83] J. Lala. "*Fault Detection, Isolation and Reconfiguration in FTMP: Methods and Experimental Results*" Proceedings of the 5th AIAA/IEEE Digital Avionics Systems Conference (DASC). 1983.

[84] Z. Segal, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, T. Link. "*FIAT - Fault Injection Based Automated Testing Environtment*". Proceedings of the 18th Fault-Tolerant Computing Symposium., Tokyo, Japan. June 1988. Pág. 102-107.

[85] G. Kanawati, N. Kanawati, J. Abraham. "*FERRARI: A Tool for the Validation of System Dependability Properties*". Proceedings of the 22nd Fault-Tolerant Computing Symposium., Boston, MA. July 1992. Pág. 336-344.

[86] L. Young, R. Iyer, K. Gosoftwareami, C. Alonso. "*Hybrid Monitor Assisted Fault Injection Environtment*". Proceedings of the Third IFIP Working Conference on Dependable Computing for Critical Aplications, Mondello, Sicily. September 1992. Pág 163-174.

[87] W. Kao, R. Iyer. "*DEFINE: A Distributed Fault Injection and Monitor Environtment*". The 1994 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems. June 1994.

[88] H. Rosenberg, K. Shin. "*Software Fault Injection and its Application in Distributed Systems*". Proceedings of the 23rd Fault-Tolerant Computing Symposium. Toulouse, Francia. June 1993. Pág. 228-227.

[89] U. Gunneflo, J. Karlsson, J. Torin. "*Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation*". Proceedings of the 19th Fault-Tolerant Computing Symposium. June 1989. Pág. 340-347.

[90] J. Karlsson, U. Gunneflo, J. Torin. "*The Efects of Heavy-ion Induced Single Event Upsets in the MC6809E Microprocessor*". Proceedings of the 4th International Conference on Fault-Tolerant Computing Systems. GI/ITG/GMA, Baden, Germany. 1989.

[91] J. Hansen, D. Siewiorek. "*Models for Time Coalescence in Event Logs*". Proceedings of the 22nd Fault-Tolerant Computing Symposium., Boston, MA. July 1992. Pág. 221-227.

[92] P. Gil, J. Baraza, D. Gil, J. Serrano. "*High Speed Fault Injector for Safety Validation of Industrial Machinery*". EWDC-8 (8th European Workshop on Dependable Computing). 1997.

[93] J. Arlat, Y. Crouzet, J. Laprie. "*Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems*". FTCS-19, Chicago IL, IEEE 1989, pág. 348-355.

[94] H. Madeira, J. Silva. "*Experimental Evaluation of the Fail Silent Behavior in Computers Without Error Masking*". FTCS-24, Austin, Texas, IEEE 1994, pág. 350-359.

[95] M. Schuette, J. Shen, D. Siewiorek, Y. Zhu. "*Experimental Evaluation of Two Concurrent Error Detection Schemes*". FTCS-16, Vienna, Austria, IEEE 1986, pág. 138-143.

[96] D. Powell, M. Cukier, J. Arlat. "*On Stratified Sampling for High Coverage Estimations*". Dependable Computing - Proceedings of the EDCC-2. Second European Dependable Computing Conference. Taormina, Italy. Oct. 1996.

[97] E. Fuchs. "*Validating the Fail-Silence Assumption of the MARS Architecture*". Six International Working Conference on Dependable Computing for Critical Applications. Grainau, Germany. March 1997.

[98] G. Zorpette. "*The Power of Paralelism*". IEEE Spectrum. September, 1990. Pág 28-33.

[99] R. Harper, J. Lala. "*Fault-Tolerant Parallel Processor*". Journal of Guidance, Control and Dynamics. Vol. 14, nº 3. 1991. Pág 554-563.

[100] O. Serlin. "*Fault-Tolerant Systems in Commercial Applications*". IEEE Computer. August, 1984. Pág 19-30.

[101] D. Johnson. "*The Intel 432: A VLSI Architecture for Fault-Tolerant Computer Systems*". IEEE Computer. August, 1984. Pág 40-48.

[102] D. Pradhan. "*Fault-Tolerant Computing: Theory and Techniques*". Vol. I. Prentice-Hall. 1986.

[103] B. Janssens. W. Fuchs. "*Experimental Evaluation of Multiprocessor Cache-Based Error Recovery*". 1991 International Conference on Parallel Processing. August 1991. Pág I-505 a I-508.

[104] N. Bowen, K. Pradhan. "*Virtual Checkpoints: Architecture and Performance*". IEEE Transactions on Computers, vol 41. May 1992. Pág 516-525.

[105] N. Bowen, K. Pradhan. "*Processor and Memory Based Checkpoint and Rollback Recovery*". Computer. February 1993. Pág 22-31.

[106] C. Dimmer. "*The Tandem Non-Stop System*". Resilient Computing Systems. 1985.

[107] D. Pradhan, N. Vaidya. "*Roll Forward Checkponting Scheme; A Novel Fault-Tolerant Architecture*". IEEE Transactions on Computers. Vol 43. October 1994. Pág 1163-1174.

[108] B. Randell. "*System Structure for Software Fault Tolerance*". IEEE Transactions on Software Engineering. SE-1. June 1975. Pág 220-232.

[109] Tandem Computers Incorporated. "*NonStop Cyclone / R*". Tandem Product Report. 1993.

[110] Sequoia Systems Incorporated. "*Series 400 Technical Summary*". June 1992.

[111] Stratus Computer Incorporated. "*XA/R Series 300*". Stratus Product Report. 1992.

[112] FASST Project Consortium. "*The FASST Architecture: Overall Requirements and Specifications*". January 1992.

[113] C. Frankenfeld, C. Morin. "*FASST recovery protocol and stable memory*". FASST Technnical Report. September 1992.

[114] B. Coghlan, J. Jones. "*TSMx200 Stable Disk integration into FASST Recovery Protocol*". FASST Technical Report. September 1992.

[115] R. Ors, J. Serrano, V. Santonja, P. Gil, A. Pérez, S. Rodriguez. "*FASST Architecture: Queuing Models and General Performance*". FASST Technical Report. 1993.

[116] M. Jöhnk, P. McGrath, C. Morin, W. Schwartz, A. Thomas. "*FASST Microkernel Specification*". FASST Technical Report. September 1992.

[117] A. Pérez, S. Rodriguez, L. Muñoz, A. García, M. Liébana y L. Prieto. "*Real Time Support Executive Implementation*". ESPRIT Project P5212. Technical Report. Universidad Politécnica de Madrid, 1994.

[118] IEEE 1194.1. "*Electrical Characteristics of Backplane Transceiver Logic Interface Circuits*".

[119] J. Gray. "*A Census of Tandem System Availability Between 1985 and 1990*". IEEE Transactions on Reliability. Vol. 39, nº 4. Oct 1990. Pág 409-418.

[120] D. Siewiorek, V. Kini, H. Mashburn, S. McConnel, M. Tsao. "*A Case Study of C.mmp, Cm, and C.vmp: Part I - Experience with Fault Tolerance in Multiprocessor Systems*". Proceedings of the IEEE. Vol. 66, nº 10. Oct 1978. Pág 1178-1199.

[121] S. McConnel, D. Siewiorek, M. Tsao. "*The Measurement and Analysis of Transient Errors in Digital Computer Systems*". Proceedings of the 9th International Symposium on Fault Tolerant Computing. 1979. Pág 67-70.

[122] R. Iyer, D. Rossetti. "*Effect on System Workload on Operating System Reliability: A Study on IBM 3081*". IEEE Transactions on Software Engineering. Vol SE-11, Nº 12. Dec 1985. Pág 1438-1448.

UNIVERSITAT DE VALÈNCIA FACULTAD DE CIÈNCIES FÍSIQUES

Reunit el Tribunal que subscriu, en el dia de la data, acordà d'atorgar, per unanimitat, a aquesta Tesi Doctoral d'En/ Na/ N' RAFAEL J. MARTINEZ DURA la qualificació d APTO CUM LAUDE por unanimidad

València a 29 d Septiembre de 1997

El Secretari,

El President,

UNIVERSITAT DE VALÈNCIA
Facultat de Ciències Físiques