Diseño y análisis de un procesador tolerante a fallos transitorios compatible con ARM a nivel de instrucciones

TRABAJO FIN DE GRADO

Andrés Gamboa Meléndez

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

Junio 2015

Documento maquetado con TEXIS v.1.0.

Este documento está preparado para ser imprimido a doble cara.

Diseño y análisis de un procesador tolerante a fallos transitorios compatible con ARM a nivel de instrucciones

Trabajo fin de grado
Grado en Ingeniería de Computadores

Versión 1.0

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

Junio 2015

Resumen

...

...

...

Abstract

...

...

...

Capítulo 1

Introducción

Resumen: En este capítulo se realiza una introducción al trabajo realizado durante el proyecto. Se plantea el problema, se enumeran los objetivos del trabajo y se define la estructura de este documento.

1.1. Introducción

Esta monografía es el resultado del estudio e investigación realizados para la asignatura «Trabajo de Fin de Grado» del Grado en Ingeniería de Computadores que se ha llevado a cabo en el departamento de «Arquitectura de Computadores y Automática (DACYA)» de la Universidad Complutense de Madrid (UCM), bajo la dirección del Dr. José Miguel Montañana Aliaga.

El trabajo se centra en el desarrollo e implementación de un microprocesador tolerante a fallos transitorios, con un diseño que le permita ser compatible con las instrucciones ARM. La tolerancia a fallos aplicada ha sido el «modelo de replicado triple de módulos(TMR)» [8].

1.2. Motivación

Hoy en día, el uso de la tecnología y la informática se extienden a nivel mundial, con aplicación a cada vez, un mayor número de campos. La tecnología está cada vez más presente en nuestras vidas, ya no se concibe un hogar o puesto de trabajo sin un ordenador sobre la mesa. El uso de los dispositivos electrónicos de carácter personal va en aumento, convirtiéndose en elementos imprescindibles en nuestros hogares (figura 1.1). Las estadísticas publicadas por el Instituto Nacional de Estadística (INE) [4], muestran que en España más del 95 % de los hogares posee al menos un teléfono móvil, normalmente teléfonos inteligentes, y más del 70 % posee un ordenador personal, lo que es

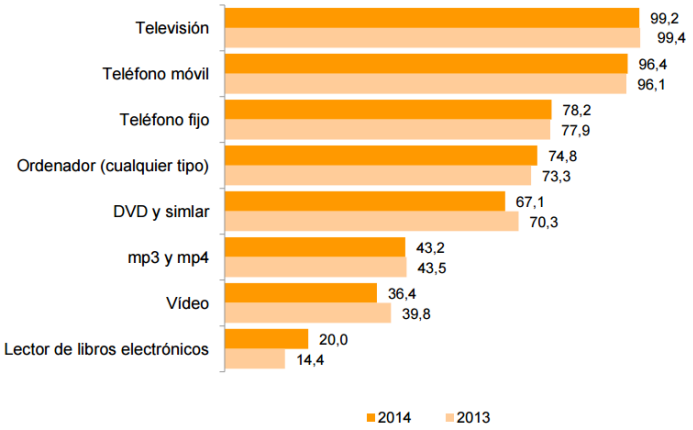


Figura 1.1: Equipamiento de los hogares en algunos productos tecnológicos.

un indicativo de la necesidad y dependencia tecnológica existente en estos tiempos.

Estos dispositivos, y muchos otros presentes en nuestra vida cotidiana, son sistemas con componentes micro-electrónicos. Poseen un microprocesador que es el «cerebro» y responsable de dirigir el sistema ejecutando los programas.

Los microprocesadores, como todos los sistemas, son susceptibles de sufrir fallos y producir errores a varios niveles (se explican en el capítulo 3), errores que provocan comportamientos erráticos y no deseados.

Con un tamaño cada vez más pequeño, los sistemas electrónicos resultan más sensibles a los efectos de los «ruidos de transmisión»[1] producidos por la radiación y los rayos cósmicos.

Las radiaciones cósmicas puede provocar fallos en cualquier sistema electrónico, dañando el mismo permanente o temporalmente, por ello, los sistemas de los satélites que orbitan alrededor de la tierra o de los aviones que se desplazan a gran altura deben ser mucho más robustos que los sistemas que trabajan a nivel del suelo. En la figura 1.2, se representa la variación de la radiación con respecto a la longitud y latitud terrestres. Se observa que hay un mayor flujo de radiación cuanto mayor es la distancia al ecuador. Fundamentalmente debido a la menor protección que proporciona la atmósfera frente a la radiación y los rayos cósmicos provenientes del espacio.

Los fallos, en el terreno de la medicina, pueden provocar consecuencias fatales e incluso la pérdida de vidas humanas. Tal puede ser el caso en los sistemas biomédicos encargados de asistir a la vida de una persona, como un marcapasos o un equipo de respiración asistida.

En el sector del transporte, vehículos y aeronaves controlados por siste-

[1] Interferencias en la señal que tiende a enmascarar la información transmitida.

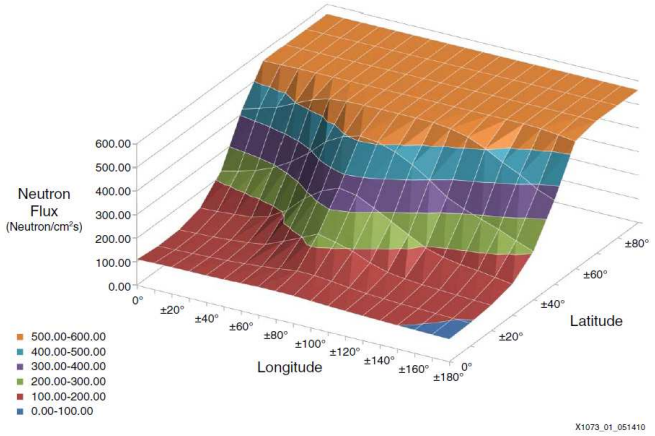


Figura 1.2: Flujo de neutrones a 40.000 pies de altitud [11].

mas electrónicos pueden sufrir fallos y causar accidentes con pérdidas humanas y económicas.

En el campo aeroespacial si un sistema falla de forma irrecuperable, causará la pérdida del sistema completo con un coste económico muy elevado.

El día 7 de octubre de 2008, un avión Airbus A330-303 que había despegado de Singapur con destino Perth, Australia, sufrió dos descensos rápidos de 650 y 400 pies. Tras las investigaciones, las autoridades australianas llegaron a la conclusión que la «Air Data Inertial Reference Unit» (ADIRU) podría haber sufrido un fallo provocado por radiaciones cósmicas [13].

Para garantizar el correcto funcionamiento de estos sistemas existen múltiples técnicas de tolerancia a fallos, técnicas que ayudan a detectar los fallos y a recuperar el sistema antes de que causen errores. Aplicando una o varias de estas técnicas se obtiene un sistema robusto y capaz de funcionar ante los fallos inducidos por la radiación u otros agentes externos.

1.3. Planteamiento del problema

Este trabajo tiene en cuenta que el motor principal de muchos sistemas es el procesador o microprocesador. El procesador de un sistema es su *cerebro*, más concretamente es el encargado de ejecutar las instrucciones que componen los programas.

Si no se toman medidas de prevención y tolerancia, este *cerebro* puede ver alterado su comportamiento por efectos externos, provocando errores de ejecución del programa. Errores que a su vez pueden ser causa de un comportamiento no deseado alterando los datos, modificando el funcionamiento del propio procesador o de otros componentes del sistema como las memorias o los controladores entrada/salida.

Con este trabajo se pretende ofrecer un medio para evitar estas situaciones concediendo un grado extra de fiabilidad a los sistemas basados en microprocesadores. Para ello se quiere diseñar e implementar un microprocesador sencillo capaz de ejecutar un conjunto reducido de instrucciones (RISC), al que posteriormente se le aplicarán las técnicas de tolerancia a fallos, aumentando así su capacidad de detectar e incluso recuperarse de los fallos.

1.4. Objetivo

Una vez planteado el problema señalado en el apartado anterior, el objetivo del presente trabajo es diseñar un procesador con un grado de fiabilidad mayor que un procesador convencional.

El proyecto se ha divido en cuatro tareas dedicadas a la implementación del microprocesador y a la aplicación de la tolerancia a fallos.

1 **Implementación del procesador.**

Se ha implementado el procesador segmentado en 5 etapas. Para ello se ha partido de la arquitectura DLX vista en las asignaturas de computadores de nuestro grado.

2 **Ruta de control**

Se ha rediseñado la ruta de control y parte de la ruta de datos. El nuevo juego de instrucciones empleado, completamente distinto al que utiliza un DLX convencional, obliga a cambiar la ruta de control. Simulando unos pequeños programas se comprueba que el procesador es capaz de decodificar y ejecutar las nuevas instrucciones.

3 **Diseño de tolerancia a fallos.**

Una vez implementado el procesador completo y comprobado su funcionamiento se diseña y se incorpora la tolerancia a fallos. Para ello se **triplican** los módulos que pueden causar mayor número de fallos y se insertan **votadores** de mayoría.

4 **Diseño del sistema de inserción de fallos.**

Para finalizar se ha diseñado un sistema externo de inserción de fallos. Este sistema es capaz de alterar los valores de las salidas de los módulos triplicados, para comprobar después como afecta esto al funcionamiento del procesador.

1.5. Estructura del documento***

Capitulo 1 *Introducción*:

En el presente capitulo 1 se realiza la introducción al proyecto propuesto y realizado para el trabajo de fin de grado que se desarrolla en este documento.

Capitulo 2 *Introducción al procesador*:

En el capitulo 2 se realiza una introducción al diseño de un procesador y sus características.

Capitulo 3 *Introducción a los fallos y su tolerancia*:

En el capitulo 3 se introducen los fallos, y se definen técnicas de tolerancia frente a estos.

Capitulo 4 *Desarrollo de procesador*:

En el capitulo 4 se describe el diseño y arquitectura final del microprocesador y sus componentes.

Capitulo 5 *Proporcionando tolerancia a fallos transitorios*:

En el capitulo 5 se describe cómo se ha proporcionado la tolerancia a fallos y qué técnicas se han utilizado.

Capitulo 6 *Resultados*:

En el capitulo 6 se muestran los resultados obtenidos de las síntesis, implementaciones y simulaciones realizadas.

Capitulo 7 *Análisis de los resultados*:

En el capitulo 7 se analizan los datos expuestos en el capítulo anterior.

Capitulo 8 *Conclusiones*:

En el capitulo 8 se describen las conclusiones tras analizar los resultados.

Capítulo 2

Introducción al procesador

«La idea detrás de los computadores digitales puede explicarse diciendo que estas máquinas están destinadas a llevar a cabo cualquier operación que pueda ser realizada por un equipo humano.»

Alan Turing

Resumen: En este capítulo se define con detalle lo que es un procesador y su importancia en el mundo de hoy en día. También se introduce la arquitectura ARM.

2.1. Procesador

El Diccionario de la Real Academia Española (DRAE) define el procesador como la «Unidad Central de Proceso (CPU), formada por uno o dos chips». Figura 2.1.

La CPU es el circuito integrado encargado de acceder a las instrucciones de los programas informáticos y ejecutarlas. Para poder ejecutar un programa, el procesador debe realizar las siguientes tareas:

1. Acceder a las instrucciones almacenadas en memoria.
2. Analizar las instrucciones y establecer las señales de control internas.
3. Ejecutar operaciones sobre datos.
4. Almacenar los resultados en memoria.

A continuación se definen los elementos fundamentales para constituir un procesador.

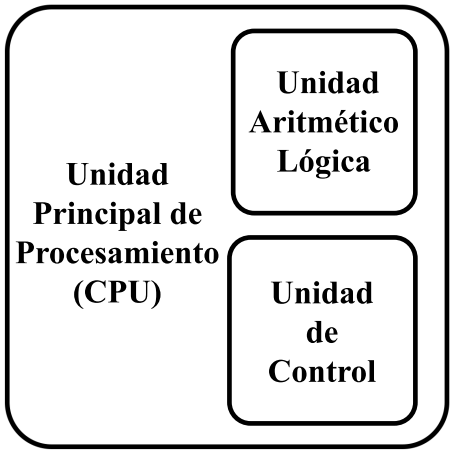


Figura 2.1: Procesador

2.1.1. Arquitectura

Un procesador está formado por una serie de módulos conectados entre sí, siendo la arquitectura del mismo la que define el diseño de los módulos que lo componen y de qué manera se conectan entre ellos.

La arquitectura del procesador diseñada por Von Neumann separa los componentes del procesador en módulos básicos. La CPU es el verdadero núcleo de los computadores, donde se realizan las funciones de computación y control. En la CPU se concentran todos los componentes, la memoria y los elementos de entrada/salida [9].

Según el juego de instrucciones que sea capaz de ejecutar un procesador, su arquitectura puede clasificarse como:

1. *Reduced instruction set computer (RISC).* Utiliza un repertorio de instrucciones reducido, con instrucciones de tamaño fijo y poca variedad en su formato.

2. *Complex instruction set computer (CISC).* Utiliza un repertorio de instrucciones muy amplio, permite realizar operaciones complejas entre las que se encuentran las de realizar cálculos entre los datos en memoria y los datos en registro.

2.1.2. Repertorio de instrucciones

El repertorio de instrucciones define todas las operaciones que el procesador es capaz de entender y ejecutar. Este juego de instrucciones incluye las operaciones aritmético-lógicas que pueden aplicarse a los datos, las operaciones de control sobre el flujo del programa, las instrucciones de lectura y

escritura en memoria, así como todas las instrucciones propias que se hayan diseñado para el procesador.

2.1.3. Memoria

Los procesadores tienen una serie de registros donde se almacenan temporalmente los valores con los que está trabajando. El conjunto de estos registros se conoce como «banco de registros». Los registros de propósito general son muy limitados, por lo que el procesador necesita disponer de apoyo externo donde alojar la información, para ello tiene acceso a una memoria externa.

El modo de acceso a la memoria externa divide las arquitecturas en dos tipos, conocidas con los nombres de Von Neumann y Harvard. La arquitectura Von Neumann utiliza una única memoria para almacenar tanto los datos como las instrucciones. La arquitectura Harvard, sin embargo, separa la memoria de datos de la memoria de instrucciones. Figura 2.2.

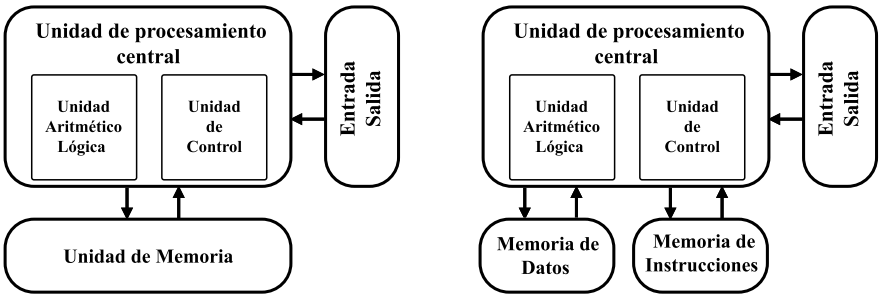


Figura 2.2: Arquitectura Von Neumann y Arquitectura Harvard

2.1.4. Segmentación

La segmentación consiste en dividir el procesador en etapas, de modo que en cada una de ellas se ejecute una parte de la instrucción. Al dividir las instrucciones se consigue que cada etapa procese de forma independiente una parte de las mismas.

Las instrucciones van avanzando de etapa en etapa hasta que se terminen de procesar. De este modo se pueden tener en el procesador varias instrucciones ejecutándose de forma simultanea en distintas etapas, lo que resulta en un aumento significativo del rendimiento del procesador.

En la tabla 2.1 podemos ver cómo se procesan una serie de instrucciones en 5 etapas (IF, ID, EX, MEM, WB). Se observa cómo cada instrucción va ocupando una única etapa en cada ciclo de reloj, cómo cambian de una a otra al ser procesadas, permitiendo la ejecución de la siguiente instrucción.

	Ciclo de reloj								
Número de instrucción	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i + 1		IF	ID	EX	MEM	WB			
i + 2			IF	ID	EX	MEM	WB		
i + 3				IF	ID	EX	MEM	WB	
i + 4					IF	ID	EX	MEM	WB

Tabla 2.1: Segmentacion simple de 5 etapas

Reducción de ciclos por segmentación

La segmentación proporciona la ventaja de poder lanzar una instrucción por cada ciclo de reloj. Característica que aumenta el rendimiento del procesador al obtener un menor número total de ciclos por instrucción para un mismo programa. Para conocer los ciclos por instrucción que necesita un programa se utiliza la formula:

$$\text{Ciclos por instrucción (CPI)} = \frac{\text{Número de ciclos total}}{\text{Número de instrucciones}} \tag{2.1}$$

A modo de ejemplo, veamos que sucede al ejecutar un programa de 3 instrucciones sobre un procesador que emplee 5 ciclos de reloj en ejecutar cualquier instrucción, pero en un caso no segmentado, y en otro caso segmentado en 5 etapas de 1 ciclo cada una:

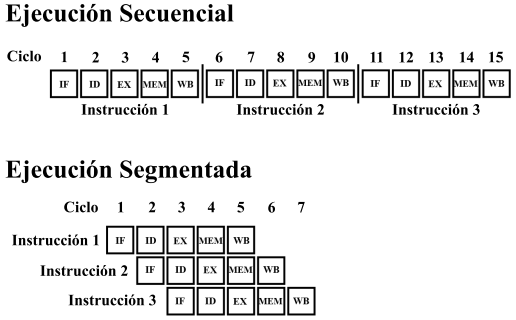


Figura 2.3: Ejecución secuencial comparada con ejecución segmentada

Como podemos ver en la figura 2.3, el procesador no segmentado tarda 15 ciclos en ejecutar las 3 instrucciones y aplicando la formula anterior se obtiene que el valor de CPI es 5. Al ejecutar el mismo programa en el procesador segmentado, este tarda 5 ciclos en ocupar las 5 etapas del procesador. A partir de ese momento con cada ciclo de reloj se completa una instrucción, completándose la ejecución del programa en 7 ciclos de reloj. El nuevo valor

de CPI es de 2,33. Así pues, la segmentación ha reducido el número de ciclos por instrucción de este programa a menos de la mitad.

Inconvenientes de la segmentación

Un programa es un conjunto de instrucciones que ejecutadas en orden realizan una tarea específica. Al permitir ejecutar una instrucción sin terminar las anteriores pueden aparecer conflictos, denominados «riesgos de segmentación» y pueden ser de los siguientes tipos: [9]

1. *Riesgos estructurales.* Surgen cuando 2 o más instrucciones necesitan acceder a los mismos recursos.

2. *Riesgos de datos.* Surgen cuando la ejecución de una instrucción depende del resultado de una instrucción anterior, y este todavía no se ha escrito en el registro correspondiente. A su vez pueden ser:

   - *Lectura después de escritura (RAW).* Una instrucción intenta leer un dato antes de que se escriba en el registro.
   - *Escritura después de lectura (WAR).* La *instrucción i+1* escribe el resultado en el registro antes de que la *instrucción i* haya leído el dato del mismo registro. Esto solo ocurre con instrucciones que realicen una escritura anticipada como por ejemplo, las instrucciones de auto-incremento de direccionamiento.
   - *Escritura después de escritura (WAW).* Ocurre cuando las escrituras se realizan en orden incorrecto. Por ejemplo, cuando en un mismo registro, la *instrucción i+1* escribe su resultado antes de que lo haga la *instrucción i.*
   - *Lectura después de lectura (RAR).* Realmente no es un riesgo como tal, ya que no se modifica ningún dato.

3. *Riesgos de control.* Surgen a consecuencia de las instrucciones que afectan al registro del contador de programa (PC).

2.2. ARM

La arquitectura ARM fue originalmente desarrollada por Acorn Computer Limited, entre los años 1983 y 1985.

Actualmente la arquitectura ARM es el conjunto de instrucciones más extensamente utilizado en unidades producidas. Esto se debe a su amplio uso en los sectores de telefonía móvil, sistemas de automoción, computadoras industriales y otros dispositivos.

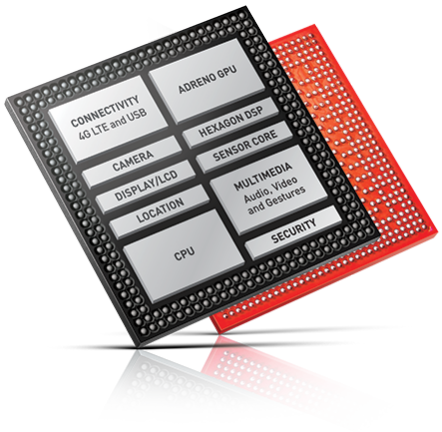


Figura 2.4: Procesador Qualcomm Snapdragon 810

Lo que hace que esta arquitectura sea tan popular es la simpleza de sus núcleos, utilizan un número relativamente pequeño de transistores, permitiendo añadir funcionalidades específicas en otras partes del mismo chip [3]. Por ejemplo, uno de los procesadores de ultima generación de la empresa Qualcomm, el «Qualcomm Snapdragon 810» esta compuesto por una CPU con 8 núcleos ARM, una unidad de procesamiento gráfico, controladores de pantalla, conectividad y cámara entre otros, todo ello en un único chip [21]. Figura 2.4.

Además de requerir poco espacio, los dispositivos ARM están diseñados con el objetivo de minimizar el consumo de energía, haciéndolo apropiado para sistemas móviles empotrados[1] que dependen de una batería.

Por último, la arquitectura ARM es altamente modular, es decir, sus componentes como pueden ser la memoria caché o los controladores, se construyen como módulos independientes y opcionales..

Todo ello no impide que la arquitectura ARM resulte muy eficiente y proporcione un alto rendimiento.

2.2.1. Arquitectura ARM

La arquitectura ARM [19] deriva de la arquitectura RISC con las características propias de esta:

- Banco de registros uniforme.
- Instrucciones de tamaño fijo.

[1] Sistema de computación diseñado para realizar una o pocas funciones dedicadas.

- Las instrucciones de procesamiento operan sobre los datos almacenados en los registros.
- Modos de direccionamiento simples.

La arquitectura ARM añade algunas características adicionales para proporcionar un equilibrio entre el rendimiento, el tamaño del código, el consumo y el silicio requerido, estas son:

- Actualización de flags en la mayoría de instrucciones.
- Ejecución condicional de instrucciones.
- Auto-incremento y auto-decremento para el direccionamiento.
- Instrucciones de carga y almacenamiento múltiple.

La arquitectura ARM contiene un banco de registros con 31 registros de propósito general. De estos sólo son visibles 16, a los cuales puede acceder cualquier instrucción. Los otros registros se utilizan para acelerar el procesado. Tres de los 31 registros tienen un uso especial y son el «puntero de pila (SP)», el «registro de enlace (LR)» y el «contador de programa (PC)».

2.2.2. Repertorio de instrucciones ARM

El repertorio de instrucciones se divide en seis categorías:

- **Salto**

  Además de permitir que las instrucciones aritmético-lógicas alteren el flujo de control, almacenando sus resultados en el registro PC, se incluye una instrucción estándar capaz de aplicar un salto de hasta 32MB hacia delante o hacia atrás.

  Otra instrucción de salto permite almacenar el valor del contador de programa en un registro para poder volver al mismo punto al finalizar el desvío. Esto es útil cuando se quiere llamar a una subrutina.

  También es posible lanzar instrucciones de salto que realizan un cambio de juego de instrucciones, en caso de necesitar lanzar subrutinas en alguno de los otros juegos de instrucciones compatibles con la arquitectura, tal es el caso de Thumb o Jazelle.

- **Procesamiento de datos**

  El procesamiento de datos se realiza mediante instrucciones aritmético-lógicas, operaciones de comparación, instrucciones sobre múltiples datos, instrucciones de multiplicación y operaciones diversas.

Las instrucciones aritmético-lógicas, como su nombre describe, ejecutan operaciones aritméticas o lógicas sobre dos operandos. El primer operando siempre será un registro, mientras que el segundo puede ser un inmediato, o un segundo registro. El resultado se almacena en un registro.

Como se ha comentado anteriormente, las operaciones de comparación aplican una operación aritmético-lógica. Sin embargo no escriben el resultado en un registro, actualizan los flags de condición.

- **Transferencia de registros de estado**

  Estas instrucciones son capaces de transferir contenidos entre los registros especiales CPSR y SPSR, y los registros de propósito general.

  Al escribir en el registro CPSR se consigue establecer los valores de los bits de condición, habilitar o deshabilitar interrupciones, cambiar el estado y el modo del procesador, y cambiar el modo de acceso a memoria entre «little endian» o «big endian».

- **Carga y almacenamiento**

  Las instrucciones de carga y almacenamiento permiten transmitir datos entre los registros de propósito general y la memoria externa.

  Se pueden cargar o almacenar los registros de forma individual, un solo dato por instrucción, o de forma colectiva, un bloque de datos con una sola instrucción.

- **Co-procesador**

  Las instrucciones de co-procesador comunican el procesador principal con un co-procesador auxiliar para transmitir instrucciones o datos.

  Existen tres clases de este tipo de instrucciones: Procesado de datos, comienza el trabajo específico del co-procesador. Transferencia de instrucciones, envía o recibe datos del procesador a la memoria. Transferencia de registro, envía o recibe datos entre los registros del microprocesador y el co-procesador.

- **Excepciones**

  Las instrucciones de excepción generan interrupciones en el programa. Las instrucciones «Interrupción software» normalmente se utilizan para realizar peticiones al sistema operativo. Mientras que las instrucciones «Punto de interrupción software» generan excepciones abortando la ejecución del programa.

Los procesadores ARM son capaces de procesar instrucciones de tres repertorios diferentes. El repertorio ARM [26], el set Thumb/ Thumb-2 [1] y las instrucciones Jazelle [26].

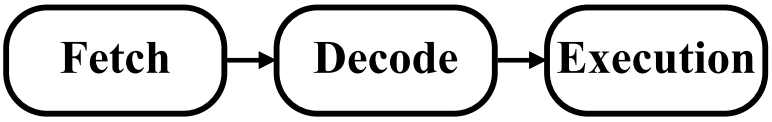


Figura 2.5: Segmentación ARM

2.2.3. Segmentación ARM

La evolución de los ARM ha significado un aumento en la cantidad de etapas en las que se divide el procesador. La familia «ARM7TDMI» consta de 3 etapas, mientras que la familia «ARM9TDMI» se divide en 5 etapas, y la familia «Cortex» se compone de 13 etapas.

La arquitectura del «ARM7TDMI», como se ha comentado, está segmentada en 3 etapas para aumentar la velocidad de flujo de entrada de las instrucciones en el procesador. Permite realizar varias operaciones al mismo tiempo y operar de forma continua. [18]

Las tres etapas en las que se divide la segmentación del «ARM7TDMI» son: (Figura 2.5)

1. **Búsqueda de instrucción**

   Se accede a la memoria para extraer la instrucción.

2. **Decodificación**

   Los registros utilizados son extraídos de la instrucción.

3. **Ejecución**

   Los valores de los registros se extraen del banco de registros, se realizan las operaciones, y se almacenan los resultados en el banco de registros.

Mientras se ejecuta una instrucción, la siguiente es decodificada y una tercera es traída de memoria.

2.2.4. Memoria ARM

Se utiliza una arquitectura Von-Neumann con un único bus de 32 bits para acceder tanto a las instrucciones como a los datos.

El único tipo de instrucciones con acceso a memoria son las instrucciones de carga y almacenamiento. Puede transmitir datos de 8, 16 o 32 bits, alineados cada 1, 2 y 4 bytes respectivamente. [18]

2.3. Field-Programmable Gate Arrays (FPGA)

Las FPGAs, del inglés Field-Programmable Gate Arrays, consisten en bloques lógicos con conexiones programables para realizar diferentes diseños

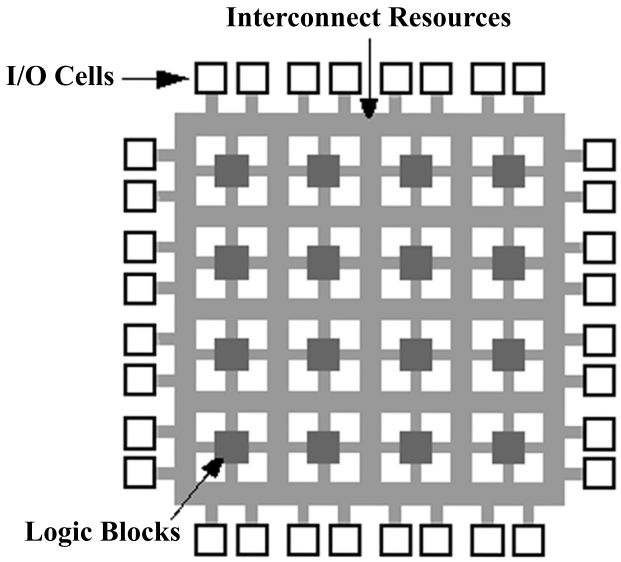


Figura 2.6: Arquitectura de una FPGA [2].

[22]. Figura 2.6.

Los bloques pueden ser tan simples como un transistor o tan complejos como un microprocesador. Las FPGAs comerciales suelen tener bloques basados en:

- Parejas de transistores.
- Puertas lógicas simples.
- Multiplexores.
- Look-up tables (LUT).
- Estructuras de puertas AND-OR.

Debido a su naturaleza re-configurable, las FPGAs conllevan un mayor coste en área, retardos y consumo de energía: requieren un área 20 veces mayor, consume 10 veces más energía y trabaja 3 veces más lenta [17]. Estas desventajas son minimizadas por la ventaja de permitir que el sistema funcione de forma casi inmediata.

Las ventajas de este tipo de dispositivos residen en su gran versatilidad, flexibilidad, su alta frecuencia de trabajo, su capacidad de procesamiento en paralelo, y a su bajo precio en comparación con los «Circuitos Integrados para Aplicaciones Específicas (ASICs)»[2].

[2]Circuitos integrados hechos a la medida para un uso particular.

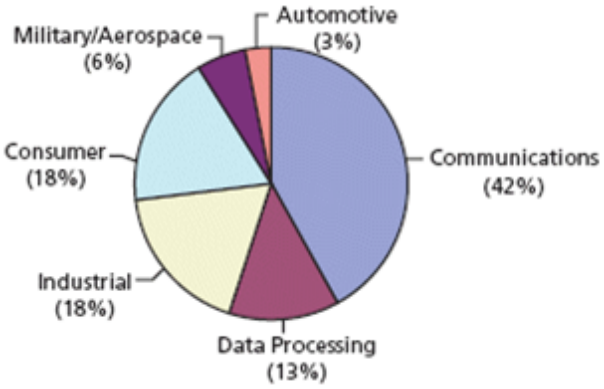


Figura 2.7: Distribución del uso de FPGAs en el año 2008.

Es por ello que las FPGAs se utilizan en todo tipo de sectores desde el procesamiento de datos y las comunicaciones hasta el sector de la automoción, el sector militar y el sector aeroespacial. Ha sido el sector de comunicaciones, como vemos en la figura 2.7, el que más uso hacía de esta tecnología en 2008.

2.3.1. Nexys 4

Para el desarrollo de este proyecto se ha utilizado la placa de prototipado «Nexys 4» [5], figura 2.8. Ésta placa se basa en la FPGA Artix-7 de Xilinx, proporcionando a la misma acceso a memorias externas, puertos de entrada/-salida (USB, ethernet, etc), y una serie de sensores y periféricos integrados (acelerómetro, sensor de temperatura, micrófono, etc).

La FPGA Artix-7 (XC7A100T-1CSG324C) [29], está optimizada para la lógica de altas prestaciones y ofrece más recursos y mejor rendimiento que sus predecesoras. Esta FPGA consta de:

- 15,850 bloques lógicas (cada porción contiene 6 LUTs y 8 biestables).
- 4,860 Kb de bloques RAM.
- 6 lineas de reloj.
- Velocidad interna de reloj superior a 450MHz.
- 240 bloques DSP.
- Conversor analógico-digital integrado (XADC).

Algunos perifericos incluidos en la placa son: 16 interruptores, 16 LEDs, puerto USB-UART, lector de tarjeta microSD, salida de audio, salida VGA, acelerómetro, sensor de temperatura, 16MB de memoria CellularRAM y puerto ethernet 10/100.

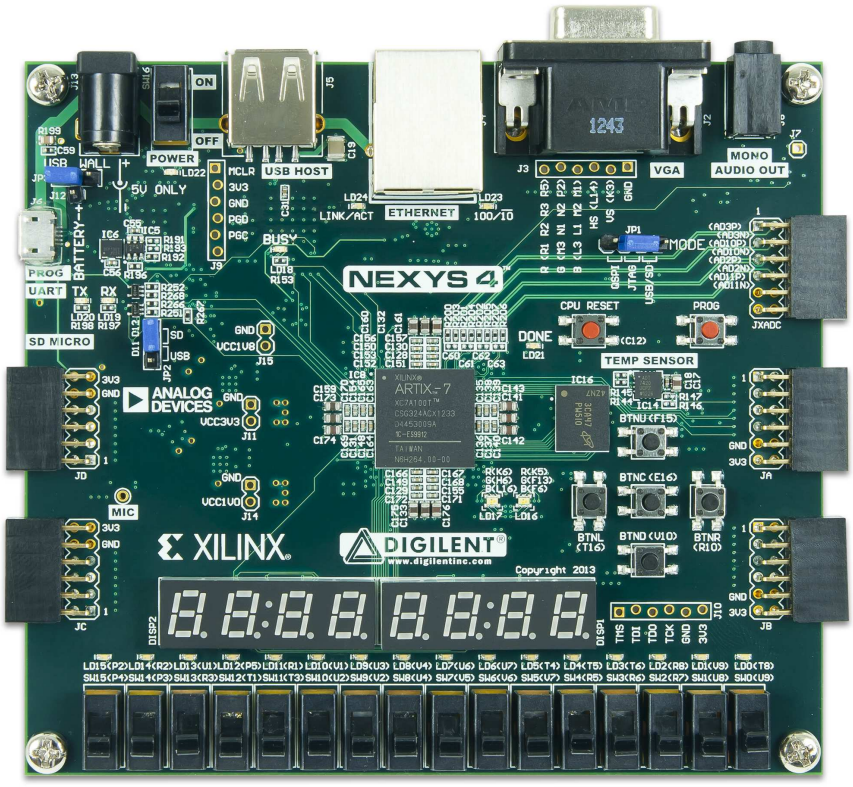


Figura 2.8: Placa de prototipado Nexys 4

Capítulo 3

Instroducción a los fallos y su tolerancia

«La verdadera ciencia enseña, sobre todo, a dudar y a ser ignorante.»

Ernest Rutherford

Resumen: En este capítulo se define qué es un fallo y qué tipos de fallos pueden ocurrir en los sistemas electrónicos. Además se introducen algunas técnicas de tolerancia a fallos y ejemplos de procesadores tolerantes a fallos.

3.1. Introducción a los fallos

Un fallo ocurre cuando un sistema no ha funcionado correctamente. Se pueden encontrar desde fallos en la definición de requisitos que se propagan hasta la fase de producción, hasta fallos producidos en el sistema por agentes externos, como la radiación. En un sistema electrónico pueden ocurrir fallos que se clasifican en *«soft errors»* o *fallos transitorios* y *«hard errors»* o *fallos permanentes*.

Cuando el fallo ocurrido afecta a los elementos de memoria alterando sus valores, lo que incluye tanto a los datos como a las instrucciones, se conoce como *«soft error»* o *fallo transitorio*. Sin embargo, si el fallo daña o altera el funcionamiento del chip, se conoce como *«hard error»* o *fallo permanente*.

En esta sección no se contemplan los fallos que se producen a partir de una mala implementación, únicamente se centra en los fallos producidos por agentes externos que no se pueden evitar en las fases de diseño, y que afectan al hardware, dañando sus componentes o alterando los valores de las señales con las que trabaja.

3.1.1. Causas

En general, estos fallos se conocen como *«Single-Event Effects (SEEs)»*. Se deben al choque de una partícula de energía contra un elemento del circuito integrado. Figura 3.1.

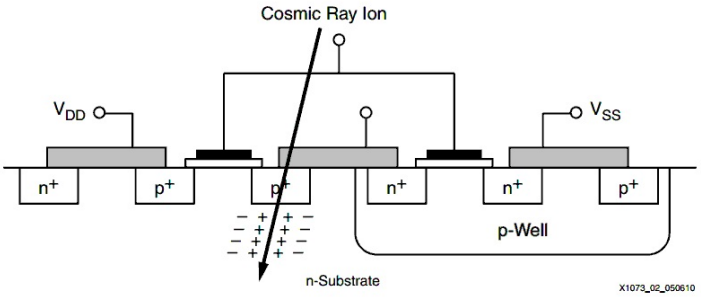


Figura 3.1: Single Event Upset en una FPGA [11].

Las partículas de energía pueden ser:

- Los **rayos cósmicos**: Si poseen suficiente carga pueden depositar energía suficiente para invertir un bit en un elemento de memoria, en una puerta lógica, o en una sección del circuito. Estos rayos pueden tener un origen galáctico o solar.

- Los **protones y neutrones de alta energía**: Bien sean de origen radiactivo o solar, pueden provocar una reacción radiactiva ionizando elementos en el chip y provocando un SEE.

Los rayos cósmicos y las partículas solares reaccionan con la atmósfera provocando un efecto de lluvia de partículas. La atmósfera actúa a modo de filtro contra estas partículas. Este efecto se distribuye de manera diferente alrededor de la tierra debido a la densidad de la atmósfera, variando la proporción de partículas que llegan a nivel de suelo y las que quedan bloqueadas.

Como ya se adelantó en la introducción, los efectos varían según la latitud, la longitud y la altitud. Figura 3.2. Al entrar en contacto con la atmósfera las partículas colisionan contra estas y pierden energía. Cuanto menor sea la densidad, mayor será el número de partículas que llega al nivel del suelo manteniendo su energía, mayor será el numero de partículas que puedan colisionar contra un chip y en consecuencia mayor la probabilidad que se produzca un fallo. Para más información sobre lluvias de partículas véase el trabajo de W.K. Melis[20].

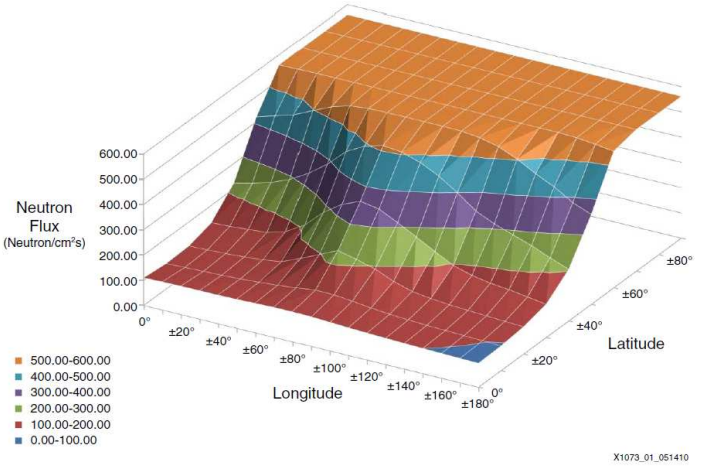


Figura 3.2: Flujo de neutrones a 40.000 pies de altitud [11].

3.1.2. Tipos de fallos

Los fallos se clasifican los fallos en dos tipos: Fallos transitorios o no destructivos, y fallos permanentes o destructivos.

Fallos Transitorios

Los *fallos transitorios*, también llamados *«soft errors»*, son aquellos que cambian el estado del dispositivo o celda sin afectar a su funcionalidad.

Los principales tipos de fallos transitorios son [14]:

- **Single-Event Upset (SEU)**

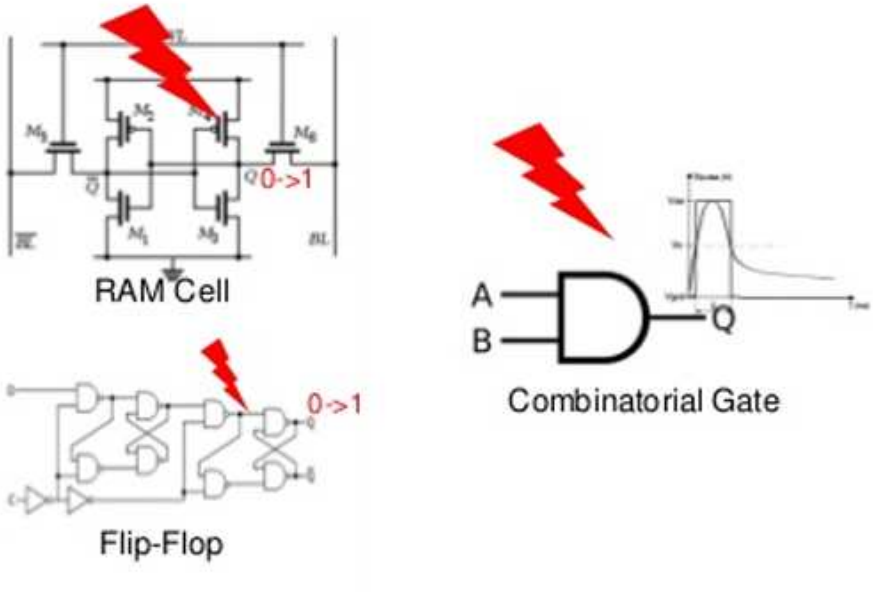
  Aquellos fallos que afectan a los elementos del chip invirtiendo su valor: memoria, celdas de memoria o registros. En un microprocesador se pueden corromper los datos del banco de registro, o los datos y las señales de control entre las etapas de segmentación. Figura 3.3a.

- **Single-Event Functional Interrupt (SEFI)**

  Fallos que producen una perdida temporal de la funcionalidad del dispositivo, provocando un mal funcionamiento detectable, que no requiere reiniciar el sistema para recuperar su funcionalidad. Normalmente se asocia con un SEU en los registros de control.

- **Single-Event Transient (SET)**

  Picos de energía provocados por una partícula en un nodo de un circuito integrado. Pueden propagarse y almacenarse en un biestable si se produce en un flanco de reloj. Figura 3.3b.



(a) Single-Event Upset (SEU) (b) Single-Event Transient (SET)

Figura 3.3: Fallos Transitorios

El sistema sufre las consecuencias como un cambio de valor en un bit. Si se produce un fallo de este tipo en una celda de memoria o en un registro de un microprocesador se corromperá el dato almacenado. Si afecta a un biestable en cualquier etapa de la segmentación, puede alterar el comportamiento de la instrucción, siendo más o menos grave según el lugar donde se produzca el fallo.

Fallos Permanentes

Los *fallos permanentes* o *«hard errors»* son los que afectan a la funcionalidad del dispositivo y lo dañan permanentemente. Pueden producir cambios en el diseño que impiden el correcto funcionamiento del módulo o circuito que lo sufre. [14]

Los principales tipos de fallos permanentes son:

- **Single-Event Latch-up (SEL)**

  Corto-circuito en un transistor que provoca el mal funcionamiento del mismo. En algunos casos pueden ser reparados reiniciando el sistema.

- **Single-Event Hard Errors (SHE)**

  Este fallo se identifica por causar que las celdas afectadas no puedan cambiar de estado.

Existen otros tipos de fallos permanentes, *Single-Event Burnout (SEB)* y *Single-Event Gate Rupture (SEGR)*, que destruyen el transistor a nivel físico.

Los fallos permanentes, una vez detectados, únicamente pueden solucionarse sustituyendo el chip o modificando la configuración interna del propio chip. Véase el apartado 3.2.3.

3.2. Tolerancia a Fallos

La tolerancia a fallos se define como la capacidad de un sistema para funcionar correctamente, incluso si se produce un fallo o anomalía en el sistema.

En ocasiones se producen fallos que no llegan a propagarse por el sitema y no producen errores en su funcionamiento, algo que ocurre cuando los cambios sufridos en un sistema debidos a un fallo, se ven enmascarados. Pueden deberse a alguna de las siguientes razones:

- **Enmascarado lógico**

  Se evita el error en una puerta lógica, gracias a que el valor del dato no es necesario para estimar la salida. En la figura 3.4 vemos que el valor de la señal invertida es indiferente para calcular el resultado ya que el resultado de una puerta «or» es «1» siempre que una de sus entradas sea «1».

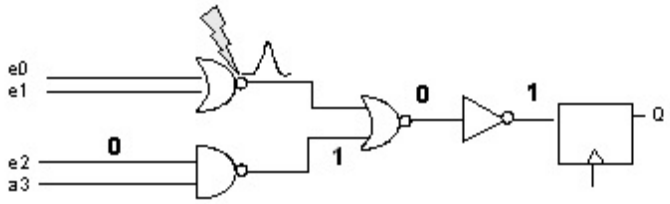


Figura 3.4: Fallo enmascarado por una puerta lógica.

- **Enmascarado eléctrico**

  El fallo producido pierde intensidad en el recorrido lógico y no tiene efecto al llegar al elemento de memoria donde se almacenaría. Figura 3.5.

- **Enmascarado temporal**

  El fallo se propaga con suficiente energía hasta el biestable, sin embargo, ocurre fuera de la de la ventana crítica de tiempo y la señal puede estabilizarse a su valor correcto antes de almacenarse en el biestable Figura 3.6.

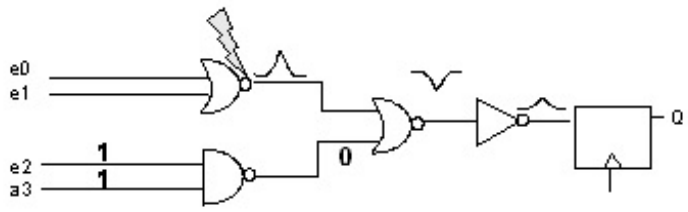


Figura 3.5: Fallo enmascarado eléctricamente.

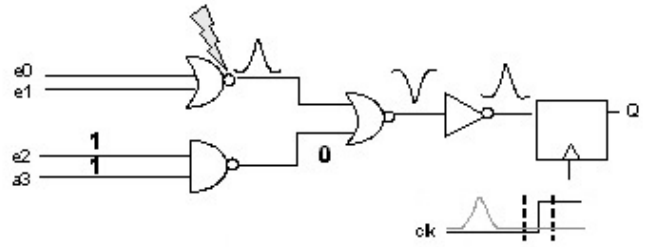


Figura 3.6: Fallo enmascarado por ventana de tiempo.

Dependiendo de la aplicación del sistema, se distinguen diferentes grados de tolerancia:

- **Tolerancia completa (fail operational)**

  El sistema puede seguir funcionando sin perder funcionalidad ni prestaciones.

- **Degradación aceptable (failsoft)**

  El sistema continua funcionando parcialmente hasta la reparación del fallo.

- **Parada segura (failsafe)**

  El sistema se detiene en un estado seguro hasta que se repare el fallo.

La tolerancia a fallos hardware se resuelve principalmente aplicando la redundancia en una o varias de sus modalidades:

3.2.1. Redundancia en la información

La redundancia de datos se basa en mantener varias copias de todos los datos en diferentes ubicaciones junto a códigos de detección y corrección de errores. La replicación de datos consigue que la perdida o daño de una memoria no implique la perdida de los datos que almacena, mientras que los

códigos de detección y corrección permiten comprobar los datos en busca de errores y corregir los datos si fuese necesario.

El ejemplo más claro de este tipo de tolerancia es el conocido como *conjunto redundante de discos independientes* o *redundant array of independent disks (RAID)*. Las diferentes clases de RAID proporcionan un acceso a los datos rápido y transparente para el sistema operativo [27]. Por ejemplo:

- **RAID 1:** Se basa en la utilización de discos adicionales sobre los que se realiza una copia de los datos que se están modificando.
- **RAID 5:** Reparte la información en bloques con bits de paridad, que se guardan en diferentes discos.

3.2.2. Redundancia en el tiempo

La redundancia en el tiempo es efectiva contra los fallos transitorios. Consiste en ejecutar parte de un programa o el programa completo varias veces. Los fallos transitorios, como se ha explicado anteriormente, se producen en zonas aleatorias del chip, siendo poco probable que aparezca el mismo error en el mismo lugar.

Aunque este tipo de redundancia requiere una menor cantidad de hardware y de software, obliga a ejecutar varias veces el programa, con lo que se produce una reducción en el rendimiento del sistema.

Algunas técnicas de redundancia en el tiempo se basan en «puntos de control» o «checkpoints». Consisten en almacenar los datos con los que se está trabajando cada cierto tiempo, se crea así un «punto de control». eUna vez se detecta un error se recurre al último «checkpoint» en lugar de tener que reiniciar el programa completo [15].

3.2.3. Redundancia en el hardware

La redundancia hardware se basa en la inserción de módulos extra para la detección y corrección de los fallos. Aunque su objetivo es el de reducir el número de fallos que provocan errores, la inserción de módulos extra implica un aumento en la complejidad del sistema, paradógicamente, con ello aumenta la posible aparición de nuevos fallos.

La tolerancia con hardware redundante se clasifica en:

- **Tolerancia estática:** Se hace uso de varias unidades que realizan la misma función en paralelo.
- **Tolerancia dinámica:** Consiste en mantener una unidad en funcionamiento y varias de repuesto para sustituirla si fuera necesario.
- **Tolerancia híbrida:** Combinan tolerancia estática con tolerancia dinámica.

Algunas técnicas se detallan a continuación [16].

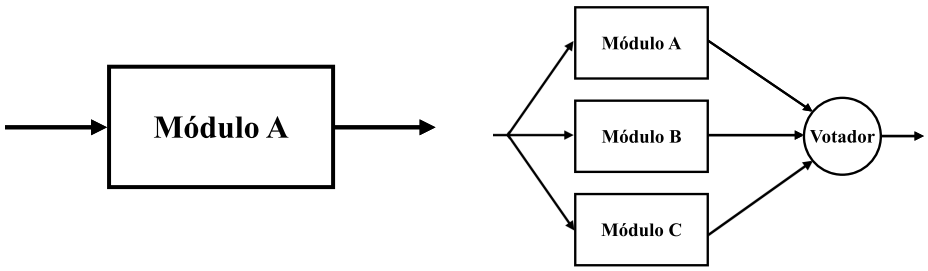
Redundancia modular

La redundancia modular consiste en replicar N veces el bloque al que se desea aplicar la tolerancia, siendo N un número impar, y a través de una votación de mayoría de las salidas extraer el valor correcto del módulo. Al aplicar la redundancia modular es posible corregir los fallos producidos en $\frac{N}{2}$ de los módulos redundantes.

El votador de mayoría es un componente de lógica combinacional que determina el valor más repetido en sus entradas. Actúa recibiendo tanto las salidas del bloque original como de cada una de las réplicas y determinando cual es el valor más repetido. De este modo los fallos quedan enmascarados.

Este método es conocido como *«N-Modular Redundancy (NMR)»*, y el uso más común de esta técnica es la *«Triple Modular Redundancy (TMR)»*, con $N = 3$.

En la figura 3.7 se observa el resultado de aplicar la TMR a un bloque.



(a) Módulo original (b) Módulo con TMR

Figura 3.7: Aplicando Triple Modular Redundancy (TMR)

Esta técnica permite evitar los fallos producidos dentro de los bloques, sin embargo inserta un nuevo punto crítico. Si el votador, un circuito combinacional, se ve afectado por un SET, este puede propagarse y afectar a los siguientes bloques, generando un error en la ejecución.

Re-configuración

La re-configuración de un sistema consiste en cambiar su implementación en el momento deseado. Por ejemplo, cuando nuestro sistema empieza a fallar debido a que parte del chip se ha dañado, en vez de eliminar el chip y sustituirlo por otro, se puede configurar el mismo circuito de manera que se eviten las zonas dañadas.

En el apartado 2.3 de este mismo capitulo se han introducido las FPGAs, sistemas re-programables que permiten al diseñador re-configurar su

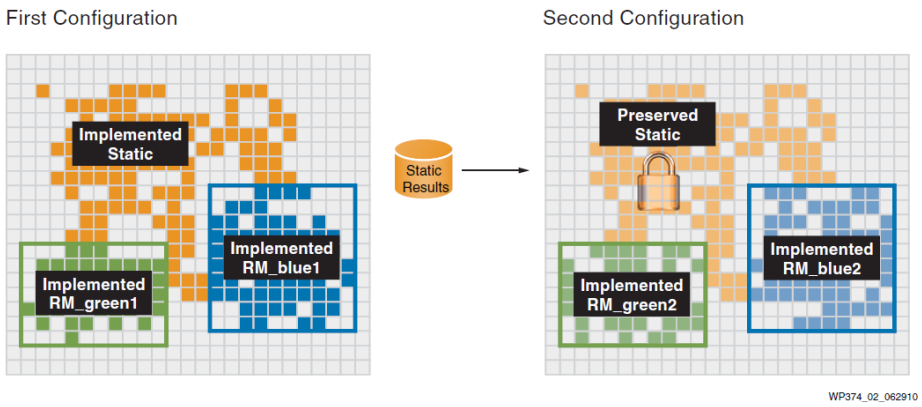


Figura 3.8: Tolerancia dinámica parcial [25].

estructura para realizar diferentes tareas, o la misma con una nueva implementación u organización de los componentes.

En la imagen 3.8 podemos ver un ejemplo de re-configuración, concretamente de re-configuración en un caso de tolerancia dinámica parcial.

Hay dos formas de aplicar la re-configuración a un sistema [7]:

- **Re-configuración estática**

  Aquella que requiere detener el sistema completamente e iniciarlo con la nueva configuración.

- **Re-configuración dinámica**

  Aquella que en tiempo de ejecución es capaz de sustituir parcial o completamente el diseño del sistema.

3.3. Tolerancia en microprocesadores

La importancia de aplicar estas técnicas a los microprocesadores viene condicionada por la utilización final que se haga de los mismos. Inicialmente, los microprocesadores nacen sin un uso específico, es precisamente su empleo final el que determina la necesidad de emplear técnicas de tolerancia. Tal puede ser el caso en misiones espaciales o como cotroladores de sistemas vitales.

La NASA utiliza microprocesadores para sus misiones espaciales. Los incluye en sus sistemas de asistencia a la vida y sistemas de experimentación. Tradicionalmente la NASA ha utilizado técnicas de tolerancia estática, como las técnicas de NMR por su buena confiabilidad [24]. En estos casos se hacen más necesarias por la mayor tasa de fallos que se da fuera de la atmósfera terrestre.

Algunas técnicas aplicadas a microprocesadores, sin modificar su diseño interno, requieren de un sistema externo conectado al microprocesador, que sea capaz de comprobar los valores internos, detectar los fallos y recuperar el sistema, relanzando las instrucciones o recuperando los valores correctos. Para prevenir los casos en los que los fallos se producen en el circuito de comparación y detección, se recurre principalmente a técnicas de NMR en estos circuito, evitando tener que modificar la estructura interna del microprocesador, se replica el sistema de comprobación, que normalmente tendrá una implementación más simple que el propio microprocesador. [28]

Otras técnicas utilizadas sobre microprocesadores evitan modificar o añadir sistemas auxiliares para comprobar y corregir los fallos. Se centran en la *«tolerancia en el tiempo»* esto es, duplicando el programa y lanzando ambos en el mismo procesador de forma simultanea. Se propone esta técnica por considerar que las técnicas de *«tolerancia hardware»* son demasiado intrusivas para el diseño, insuficientes para cubrir los fallos lógicos o demasiado costosa para la computación de proposito general[23].

Existen procesadores tolerantes a fallos en el mercado, tales como el «LEON3FT», una implementación tolerante a fallos de la tercera versión del procesador «LEON», el «IBM S/390 G5» o el «Intel Itanium» [6].

En concreto el «LEON3FT», fue diseñado para misiones militares y espaciales. Tiene cuatro modos de tolerancia que dependen de la tecnología utilizada y de la cantidad de bloques RAM disponibles. El modo de tolerancia se selecciona a la hora de sintetizar el diseño y estos modos pueden ser: [12]

- **Biestables resistentes a la radiación o TMR.**

  Se utilizan registros compuestos de biestables reforzados para resistir la influencia de la radiación o se utiliza la técnica TMR.

- **Paridad de 4-bits con reinicio.**

  Se utiliza un código *«checksum»* de 1 bit por cada byte, 4 bits por palabra. Se reinicia la cola de segmentación para corregir los fallos.

- **Paridad de 8-bits sin reinicio.**

  Se utiliza un código *«checksum»* de 8 bits por palabra y permite corregir 1 bit por byte, puede llegar a corregir 4 bits por palabra. La corrección se realiza sin reiniciar la cola de segmentación.

- **Código BCH de 7 bits con reinicio.**

  Se utiliza un código *«BCH checksum»* de 7 bits, que permite detectar fallos en 2 bits y corrige 1 bit por palabra. La cola de segmentación se reinicia al aplicar la corrección.

El procesador «IBM S/390 G5» duplica la cola de segmentación hasta la etapa de escritura, lanzando la misma instrucción dos veces. En la etapa de escritura se comparan los resultados , en caso de discrepancia no se escribe el resultado y se reinicia la ejecución desde la instrucción fallida. La ventaja proporcionada por este método reside en que el tiempo de propagación de las señales no se ve afectado por la inserción la lógica del votador. En caso contrario, el reiniciar la cola de segmentación puede costar miles de ciclos de reloj. [6]

Por último, la implementación de Intel en el «Intel Itanium» incluye una combinación de códigos de correción de errores y códigos de paridad en las memoria caché y TLB. [6]

Capítulo 4

Procesador

...

...

Resumen: En este capitulo se presenta el diseño del procesador implementado: su arquitectura,

4.1. Introducción

Basado en la arquitectura de los procesadores DLX estudiados durante el grado en ingeniería de computadores [10], se ha diseñado e implementado un procesador con arquitectura RISC. Se trata de un procesador con un ancho de palabra de 32 bits y una segmentación en 5 etapas.

La implementación ha sido adaptada para poder ejecutar instrucciones del repertorio ARM. En concreto, se permite ejecutar un subconjunto del juego de instrucciones THUMB-2 que es utilizado principalmente por los procesadores de la gama ARM CORTEX M.

Para el desarrollo de este proyecto se ha utilizado la tecnología de las FPGAs, en concreto la placa «Nexys 4» que hace uso de la FPGA «Artix 7» de Xilinx.

Para la implementación se ha utilizado el lenguaje de diseño hardware VHDL junto al software «ISE Design Suite 14.4» de Xilinx y el software «ModelSim PE Student Edition» de Altera para las simulaciones. Con ello se ha conseguido probar el diseño y la implementación de forma rápida y realizar las correcciones necesarias.

4.2. Arquitectura del procesador

El microprocesador es una implementación modificada de la arquitectura DLX para permitir ejecutar instrucciones del repertorio ARM. A continuación se describen sus características completas.

4.2.1. Estructura

El procesador se compone de los siguientes elementos (figura 4.1):

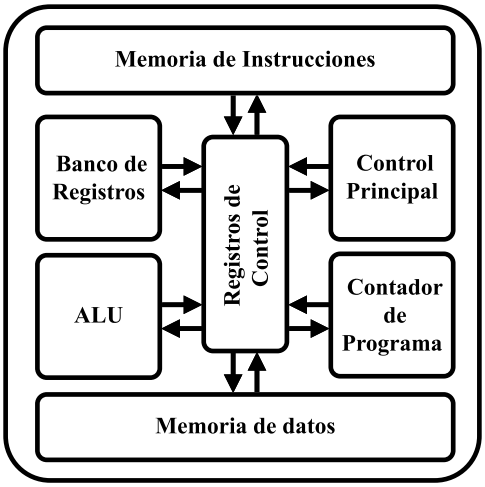


Figura 4.1: Estructura del procesador diseñado

- **Banco de registros:** Dispone de 16 registros (R0, R1, ..., R15) de propósito general con un tamaño de 32 bits. Estos registros se pueden utilizar tanto para guardar datos leídos de memoria como enviar los valores a memoria. Igualmente se puede trabajar con los valores que tengan almacenados ejecutando operaciones sobre ellos. El registro R15 es accesible de forma limitada puesto que el identificador de este registro se utiliza para diferenciar unas instrucciones de otras.
- **Contador de programa (PC):** Este registro especial almacena la dirección de memoria de la instrucción que debe ejecutarse a continuación. Se incrementa automáticamente en 4 cada ciclo y solo se puede alterar este mecanismo por medio de instrucciones de control.
- **Memoria de instrucciones:** Memoria que almacena el código de programa, al cual accede el procesador para analizar y ejecutar las instrucciones.
- **Memoria de datos:** Esta memoria conserva los valores de los datos que son accesibles por el procesador mediante instrucciones de carga y

almacenamiento.

- **Control principal:** Analiza las instrucciones para extraer la información necesaria que permita ejecutarles adecuadamente. Esta información se propaga medante señales de control al resto de componentes.

- **Unidad Aritmético-Lógica (ALU)**: Es el componente encargado de realizar las operaciones aritméticas o lógicas sobre los operandos.

- **Registros de control:** Almacena las señales de control, y las señales internas entre las diferentes etapas de la segmentación.

4.2.2. Repertorio de instrucciones

El procesador implementado es capaz de ejecutar 3 tipos de instrucciones:

- Accesos a memoria

- Procesamiento de datos

  a). Operaciones con dos registros

  b). Operaciones con un registro y un inmediato

- Operaciones de control

A continuación se explican brevemente los diferentes tipos de instrucciones. Más adelante se expondrán las instrucciones con más detalle, explicando los campos de cada una.

Accesos a memoria

Las instrucciones de acceso a memoria son necesarias cuando se requiere cargar (load) un dato desde la memoria al banco de registros, o almacenar (store) el valor de un registro en la memoria.

Aunque es posible acceder a las direcciones de memoria direccionadas por media palabra. En esta implementación se está obligado a cargar valores de tamaño 4 bytes (tamaño de palabra), siendo por tanto recomendable utilizar direcciones de memoria que sean múltiplos de 4.

Para el cálculo de la dirección efectiva de carga o almacenamiento se ha implementado un único modo de direccionamiento. Registro base Rn + imm12", es decir, la dirección base se obtiene del registro Rn, y se suma un inmediato de 12 bits extraído de la instrucción.

Procesamiento de datos

Las instrucciones de procesamiento realizan cálculos aritméticos y lógicos. Se aplican sobre dos operandos y el resultado (si existe) se almacena en un registro.

Dependiendo de la instrucción los operandos pueden ser:

- **Operaciones con dos registros:**

  Los datos de trabajo se extraen de dos registros.

  Al utilizar el registro R15 se deben tener en cuenta ciertas restricciones ya que se utiliza para diferenciar unas operaciones de otras. Por ejemplo, si el código de operación es "0010", el registro origen Rn es R15 ("1111") entonces la operación ejecutada será la operación "MOVE". Si el registro Rn es cualquier otro, se ejecutará una "Ó lógica"(operación or).

- **Operaciones con un registro y un inmediato:**

  El conjunto de operaciones con inmediato se limita a cuatro. Se permite mover un inmediato a la mitad más significativa, o a la menos significativa, de un registro. Y se permite sumar o restar un inmediato al valor de un registro.

Operaciones de control

Las operaciones de control intervienen en la ejecución normal del programa y se utilizan para modificar el valor del registro del contador de programa. Esta operación se conoce como «instrucción de salto».

Existen dos tipos de instrucciones de salto. La primera es el salto incondicional y permite sumar un entero al valor del contador de programa y almacenar el resultado en el mismo.

La segunda operación de control es el salto condicional. Previamente a un salto condicional se debe ejecutar una operación de comparación para actualizar los flags de comparación de la ALU. Los flags se comparan a la condición de salto y en caso de coincidir, se efectúa el salto. Si no se ejecuta la comparación, el estado de los flags es desconocido y el procesador se comportará de manera no controlada.

4.2.3. Segmentación

El microprocesador ha segmentado en 5 etapas, en cada una de las cuales realiza una parte fundamental en la ejecución de las instrucciones. Las etapas en las que se divide el procesador son:

1. **Búsqueda de instrucción (IF):**

La primera etapa es la encargada de cargar las instrucciones de memoria y transmitirlas a la siguiente, simultáneamente se calcula la dirección de la siguiente instrucción.

2. **Decodificación de instrucción(ID):**

   En la etapa de decodificación se analiza la instrucción y se obtienen los datos necesarios para realizar las operaciones correctamente.

3. **Ejecución (EX)**

   En esta etapa se realizan los cálculos aritméticos o lógicos sobre los datos obtenidos del banco de registro y del circuito de extensión de signo.

4. **Acceso a Memoria (MEM)**

   En la etapa de memoria se realizan intercambios de datos con la memoria principal.

5. **Escritura en registros (WB)**

   Es la etapa final del procesador en la que se escriben los resultados calculados por la ALU o los datos cargados de memoria en el banco de registros.

4.2.4. Memoria

El diseño del procesador hereda el sistema de memoria de la arquitectura Harvard, es decir, tiene acceso a dos memorias diferentes:

- **Memoria de instrucciones:**

  Memoria ROM donde se almacenan todas las instrucciones del programa a ejecutar.

- **Memoria de datos:**

  Memoria RAM accesible en modo lectura y en modo escritura para almacenar los datos con los que trabaja el programa.

4.3. Implementación

La implementación del proyecto se ha realizado utilizando la herramienta «ISE Design Suite» de Xilinx y el lenguaje de diseño hardware «VHDL».

En la figura 4.2 se muestra el diseño completo del microprocesador segmentado. En color negro se muestra la ruta de datos y los componentes. En color azul se destaca la ruta de control y el módulo de control principal. Y en color rojo aparecen los registros de control.

En esta sección se describen los componentes principales del procesador descritos en la figura 4.1.

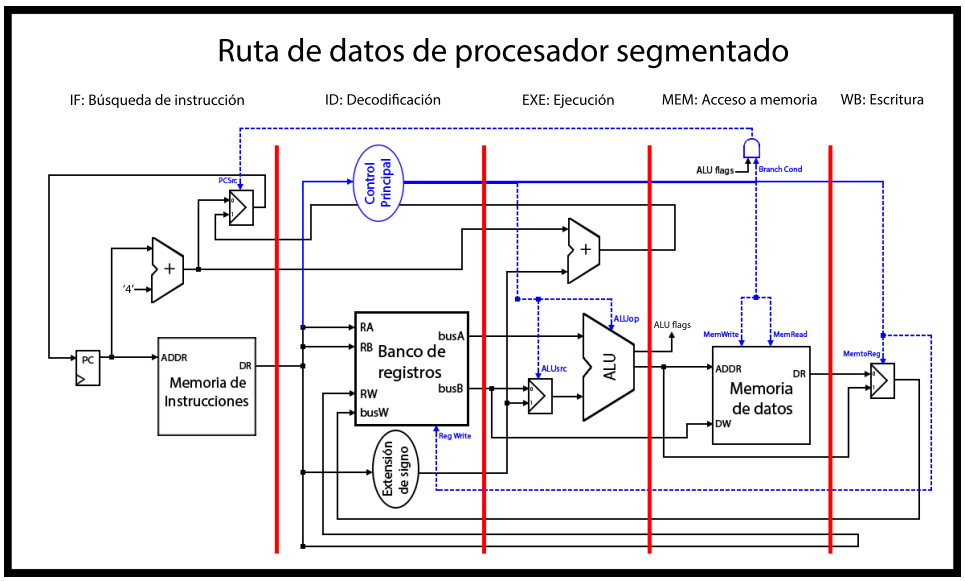


Figura 4.2: Diseño completo del procesador segmentado.

4.3.1. Banco de registros

El banco de registros es el componente de memoria con el que opera principalmente el procesador. De este obtiene los datos con los que realiza la mayoría de las operaciones.

Internamente se compone de 16 registros que almacenan valores de 32 bits. El acceso a estos se codifica en 4 bits de modo que el registro accedido con el valor «1010» es el registro «R10».

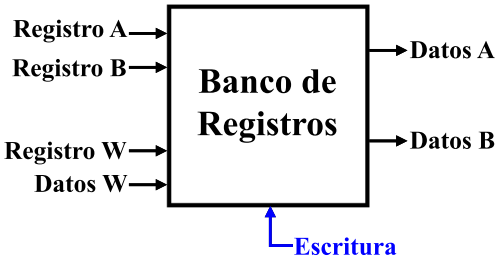


Figura 4.3: Banco de registros.

Sus señales externas son (figura 4.3):

- **Señales de entrada**
  - **Registro A:** Registro origen del primer operando de la instrucción.

- **Registro B:** Registro origen del segundo operando de la instrucción.
- **Registro W:** Registro donde se almacenará el resultado de la instrucción.
- **Datos W:** El valor que se almacenará en el registro W.
- **Escritura:** Habilita la escritura en el registro W.

▪ **Señales de salida**

- **Datos A:** Valor del registro A, primer operando de la instrucción.
- **Datos B:** Valor del registro B, segundo operando de la instrucción.

4.3.2. Contador de programa

El contador de programa es un registro común de 32 bits encargado de almacenar la dirección de memoria que indica donde se encuentra la siguiente instrucción del programa. Su valor se actualiza en cada ciclo de reloj.

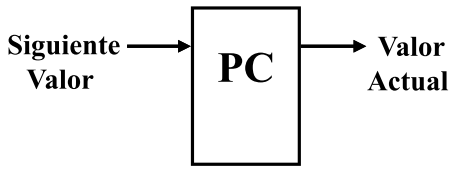


Figura 4.4: Contador de programa.

Sus señales externas son (figura 4.4):

▪ **Señales de entrada**

- **Siguiente Valor:** El valor de la siguiente instrucción. Puede ser calculado de forma secuencial, o por medio de un salto efectivo.

▪ **Señales de salida**

- **Valor actual:** Dirección origen de la instrucción que debe ejecutarse.

4.3.3. Unidad Aritmético-Lógica (ALU)

La unidad aritmético-lógica aplica ciertas operaciones sobre los datos extraídos previamente del banco de registros y de la instrucción. Así mismo, y dependiendo del resultado, puede modificar el valor de los flags de comparación que se modifican solo si la instrucción así lo requiere.

Las señales externas son (figura 4.5):

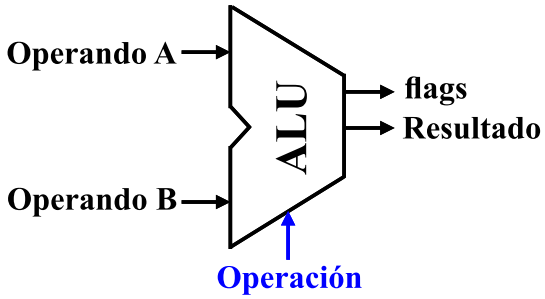


Figura 4.5: Unidad aritmético-lógica.

- **Señales de entrada**
  - **Operando A:** Primer operando de la operación.
  - **Operando B:** Segundo operando de la operación.
  - **Operación:** Operación que debe aplicarse sobre los operadores.
- **Señales de salida**
  - **flags:** Indican si el resultado de la operación cumple ciertas condiciones[1].
  - **Resultado:** Valor de aplicar la operación a los operandos.

4.3.4. Control principal

El control principal del microprocesador es el encargado de analizar la instrucción y establecer las señales de control que permitirán a los módulos realizar la función adecuada. Es un módulo combinacional, por lo tanto analiza y establece los valores de las señales de control dentro de un único ciclo de reloj.

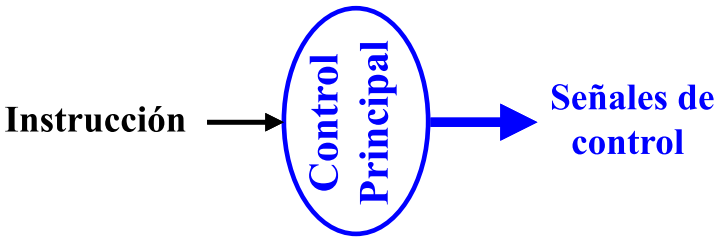


Figura 4.6: Control principal.

[1] Si el resultado de la operación es igual a cero se activa el flag correspondiente.

La instrucción es la una única entrada para este módulo, sin embargo, las señales de control son varias y se transmiten de etapa a etapa de la segmentación hasta que son consumidas. A continuación se explican por el orden en el que son utilizadas(figura 4.2):

1. **Ejecución (EX)**
   - **ALUsrc:** Establece el origen del segundo operando de la instrucción, este puede ser un registro o la propia instrucción.
   - **ALUop:** Asigna a la ALU la operación que debe aplicar a los operandos.
2. **Acceso a Memoria (MEM)**
   - **Branch Cond:** Junto a los flags de la ALU, establece la señal **PCSrc** encargada de efectuar un salto en el código del programa.
   - **MemWrite:** Indica al módulo «memoria de datos» si debe acceder a memoria en modo escritura[2].
   - **MemRead:** Indica al módulo «memoria de datos» si debe acceder a memoria en modo lectura[2].
3. **Escritura en registros (WB)**
   - **MemtoReg:** Establece el origen de los datos que deben almacenarse en el banco de registros, puede ser la ALU o la memoria de datos.
   - **RegWrite:** Indica al banco de registros si el resultado de la instrucción debe almacenarse en un registro.

4.3.5. Registros de control

Los registros de control son todos aquellos que almacenan la información entre las etapas , se muestran como líneas rojas en la figura 4.2.

Esta colección de registros son fundamentales para que exista la segmentación. Separan las etapas de forma que los cambios en una de ellas no se propaguen y no interfieran con las demás etapas.

Para que esto no suceda se inserta una serie de registros entre, por ejemplo, los componentes de las etapas de decodificación y ejecución. Éstos registros se actualizan al final de cada ciclo de reloj, conservando los datos de forma estable para que se puedan utilizar correctamente en la siguiente etapa. Así se permite que ambas etapas trabajen de forma independientemente.

[2] El modo lectura y el modo escritura en memoria son incompatibles, solo debe activarse una de estas señales.

Figura 4.7: Registros de control

4.3.6. Memoria de instrucciones

La memoria de instrucciones es una memoria de solo lectura (ROM) Este tipo de memorias, como su nombre indica, solo permite la lectura de sus datos, y no permite que se modifiquen. Este módulo permite que se consulte el valor de la dirección en un único ciclo de reloj.

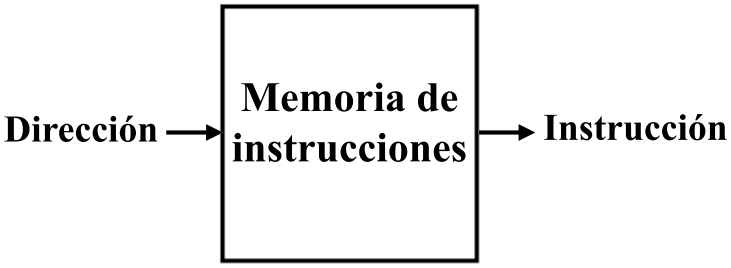


Figura 4.8: Memoria de Instrucciones.

Las señales externas de esta memoria son (figura 4.9):

- **Señales de entrada**
  - **Dirección:** Dirección de la instrucción a la que se accede.
- **Señales de salida**
  - **Instrucción:** Instrucción leída de memoria.

4.3.7. Memoria de datos

El módulo de la memoria de datos es el encargado de almacenar aquellos datos que no son inmediatamente necesarios para ejecutar las instrucciones.

Son de un mayor tamaño que el banco de registros y, por lo general, más lentos a la hora de transmitir la información.

Este módulo se ha implementado como un banco de registros de mayor tamaño que el módulo con el mismo nombre. Su acceso se completa en un mismo ciclo de reloj.

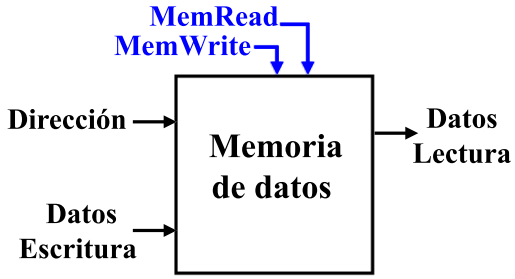


Figura 4.9: Memoria de datos.

Las señales externas de esta memoria son (figura 4.9):

- **Señales de entrada**
  - **Dirección:** Dirección de acceso a memoria, ya sea en modo lectura o escritura.
  - **Datos Escritura:** Datos que deben almacenarse en la memoria.
  - **MemWrite:** Indica que los datos deben escribirse en memoria[3].
  - **MemRead:** Indica que los datos deben leerse de memoria[3].
- **Señales de salida**
  - **Datos Lectura:** Datos que se han leído de la dirección indicada de memoria.

4.4. Formato de instrucciones

En esta sección se exponen el formato de las instrucciones que es capaz de ejecutar el microprocesador con todos sus campos, y el significado de estos.

4.4.1. Accesos a Memoria

Estas instrucciones permiten al microprocesador acceder a los valores almacenados en la memoria de datos así como almacenar datos en ella. Las

[3] El modo lectura y el modo escritura en memoria son incompatibles, solo debe estar activa una de estas señales.

instrucciones de carga o almacenamiento se identifican por los 7 bits más significativos de la instrucción, estos deben ser «1111100».

Se ha implementado un único tipo de instrucción de acceso a memoria. Éste, dependiendo del valor de sus campos, se utiliza para cargar de o almacenar datos en memoria.

La instrucción permite realizar transferencias de datos entre el banco de registros y la memoria de datos del microprocesador. Permite aplicar un desplazamiento de hasta 4KB al valor base del registro.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Formato general	1	1	1	1	1	0	0									
Rn + imm12								S	1	Size		L	Rn			

Tabla 4.1: Instrucciones de acceso a memoria (bits 31..16)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Formato general																
Rn + imm12	Rd				imm12											

Tabla 4.2: Instrucciones de acceso a memoria (bits 15..0)

Los campos de la instrucción «Rn + imm12» representados en las tablas 4.1 y 4.2 son los siguientes:

- **Extensión de signo (S):** Indica si se debe extender el signo del valor inmediato (S=1).
- **Tamaño (Size):** Indica el tamaño del valor que debe cargar de o almacenar en memoria. Puede cargar datos de tamaño 1, 2 o 4 bytes. No se utiliza.
- **Cargar/Almacenar (L):** Este bit indica si la operación debe leer un dato de memoria (L=1) o escribirlo (L=0).
- **Dirección (Rn):** Indica el registro con la dirección base de acceso a memoria.
- **Dato (Rt):** Indica el registro dónde se debe almacenar el dato en caso de carga, o el registro cd dónde debe extraerse el dato en caso de almacenamiento.
- **Desplazamiento (imm12):** Es el desplazamiento que debe aplicarse a la dirección base para obtener la dirección efectiva.

4.4.2. Procesamiento de datos

Las operaciones se distinguen según el tipo de operandos que se apliquen. El operando A siempre es obtenido de un registro. Mientras que el operando

B puede ser el valor de un segundo registro o puede formar parte de la instrucción.

Operaciones con dos registros

Las operaciones que hacen uso de dos registros son aritméticas (sumar, restar y mover), lógicas (and, or y or exclusiva) y de comparación que activan diferentes flags (Negativo, Cero). En el caso de una comparación no se modifican los registros. Las instrucciones de operación con dos registros se identifican por los 7 bits más significativos. Éstos deben ser «1110101».

Se ha implementado un único tipo de instrucción de procesamiento con dos registros. Dependiendo del valor de sus campos se aplicará una operación u otra sobre los operandos.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Formato general	1	1	1	0	1	0	1									
Data Processing								OP				S	Rn			

Tabla 4.3: Instrucciones de procesamiento de datos con dos registros (bits 31..16)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Formato general																
Data Processing	SBZ	imm3			Rd				imm2		type		Rm			

Tabla 4.4: Instrucciones de procesamiento de datos con dos registros (bits 15..0)

Los campos de la instrucción «Data Processing» representados en las tablas 4.3 y 4.4 son los siguientes:

- **Código de operación (OP):** Indica la operación que debe realizarse en la fase de ejecución sobre los operandos.
- **Activar flags (S):** Indica si deben activarse los flags de salto al ejecutar la operación.
- **Registro origen A (Rn):** Indica el registro origen del primer operando.
- **Should be Zero (SBZ):** Este campo debe tener un valor de 0.
- **Inmediato (imm3:imm2):** Indica el desplazamiento que debe aplicarse al segundo operando. No se utiliza.
- **Registro destino (Rd):** Indica el registro destino donde se almacenará el resultado de la operación.

- **Tipo de desplazamiento (type):** Indica el tipo de desplazamiento aplicado. No se utiliza.
- **Registro origen B (Rm):** Indica el registro origen del segundo operando.

Las operaciones implementadas junto con su respectiva codificación se muestran en la tabla 4.5.

Operación	Código	Restricciones
ADD	1 0 0 0	
AND	0 0 0 0	
CMP	1 1 0 1	(Rd==«1111»,S==1)
EOR	0 1 0 0	
MOV	0 0 1 0	(Rn==«1111»)
ORR	0 0 1 0	
SUB	1 1 0 1	

Tabla 4.5: Operaciones con dos registros

Operaciones con un registro y un inmediato

Las operaciones que hacen uso de un registro y un inmediato únicamente pueden ser aritméticas (sumar, restar y mover). Estas instrucciones se dividen en dos tipos según el tamaño del inmediato utilizado. Para identificar este tipo de instrucciones se utilizan los 5 bits más significativos, que deben ser «11110» y el bit 15 que debe valer «0».

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Formato general	1	1	1	1	0											
Add, Subtract, plain 12-bit immediate						i	1	0	OP	0	OP2		Rn			
Move, plain 16-bit immediate						i	1	0	OP	1	OP2		imm4			

Tabla 4.6: Instrucciones de procesamiento de datos con un registro y un inmediato (bits 31..16)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Formato general	0															
Add, Subtract, plain 12-bit immediate		imm3			Rd				imm8							
Move, plain 16-bit immediate		imm3			Rd				imm8							

Tabla 4.7: Instrucciones de procesamiento de datos con un registro y un inmediato (bits 15..0)

Los campos de las instrucciones con inmediato, representados en las tablas 4.6 y 4.7 son:

- **«Add, Subtract, plain 12-bit immediate»**
  - **Código de operación (OP:OP2):** Indica la operación que debe aplicarse en la fase de ejecución sobre los operandos.
  - **Registro origen (Rn):** Indica el registro origen del primer operando.
  - **Inmediato (i:imm3:imm8):** Contiene el inmediato que se utiliza como segundo operando.
  - **Registro destino (Rd):** Indica el registro destino donde se almacenará el resultado de la operación.
- **«Move, plain 16-bit immediate»**
  - **Código de operación (OP:OP2):** Indica la operación que debe aplicarse en la fase de ejecución sobre los operandos.
  - **Inmediato (imm4:i:imm3:imm8):** Contiene el inmediato que se utiliza como segundo operando. Utilizado en las operaciones mover.
  - **Registro destino (Rd):** Indica el registro destino donde se almacenará el resultado de la operación.

Las operaciones implementadas junto con su respectiva codificación se muestran en la tabla 4.8.

Operación	Código
ADD	0 0 0
SUB	1 1 0
MOVT	1 0 0
MOV	0 0 0

Tabla 4.8: Operaciones con un registro y un inmediato

4.4.3. Operaciones de control

También conocidas como instrucciones de salto, estas instrucciones son aquellas capaces de alterar el contador de programa. Para identificar este tipo de instrucciones se utilizan los 5 bits más significativos, que deben ser «11110» y el bit 15 que debe tener un valor de «1».

Se han implementado dos tipos de instrucciones de salto: salto incondicional, y salto condicional. El salto incondicional se realiza siempre que esté presente la instrucción. La operación de salto condicional sólo se efectúa cuando coinciden las condiciones de la instrucción con los flags previamente calculados de la ALU.

En el caso del salto incondicional se permite realizar un salto de 16MB por el código. El salto condicional, debido a necesitar un campo que indique la condición, puede realizar un salto de 1MB.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Formato general	1	1	1	1	0											
Branch						S	offset[21:12]									
Conditional Branch						S	cond				offset[17:12]					

Tabla 4.9: Instrucciones de control (bits 31..16)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Formato general	1															
Branch		0	I1	1	I2	offset[11:1]										
Conditional Branch		0	J1	0	J2	offset[11:1]										

Tabla 4.10: Instrucciones de control (bits 15..0)

Los campos de las instrucciones de control representados en las tablas 4.9 y 4.10 son:

- **Salto (Branch)**
  - **Extensión de signo (S):** Indica si se debe extender el signo del desplazamiento (S=1).
  - **Desplazamiento (offset):** Contiene el inmediato que se suma al registro PC para calcular la dirección efectiva del salto. Los campos «I1» e «I2» son respectivamente los bits 23 y 22 del desplazamiento.
- **Salto condicional (Conditional Branch)**
  - **Extensión de signo (S):** Indica si se debe extender el signo del desplazamiento (S=1).
  - **Condición de salto (cond):** Indica la condición necesaria para que el salto deba realizarse.
  - **Desplazamiento (offset):** Contiene el inmediato que se suma al registro PC para calcular la dirección efectiva del salto. Los campos «J1» e «J2» son respectivamente los bits 19 y 18 del desplazamiento.

Capítulo 5

Aplicando tolerancia a fallos

...

...

Resumen: En este capitulo se explica cómo se ha aplicado la tolerancia a fallos en el microprocesador.

5.1. Introducción

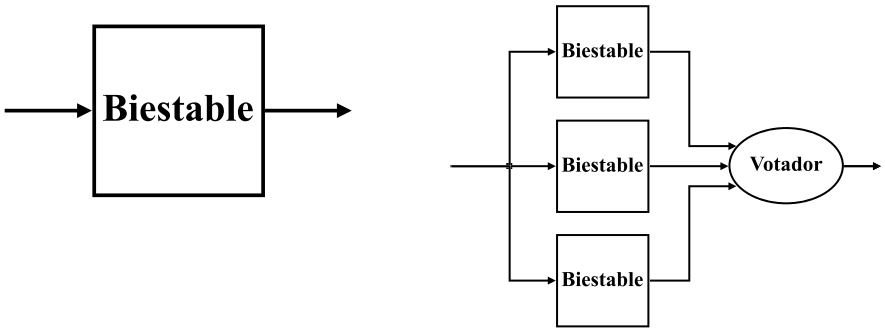
Los fallos más comunes en los sistemas electrónicos son los conocidos como fallos transitorios, estos no dañan el sistema de forma permanente pero provocan un cambio de valor en los elementos de memoria (SEU) o interferencias en las conexiones internas (SET), que si no se estabilizan a tiempo pueden llegar a propagarse hasta una celda de memoria y almacenarse provocando el mismo efecto que un SEU. Con esta información se ha implementado un método para paliar y reducir los efectos tanto de los SEU como de los SET.

5.2. Redundancia modular

La técnica utilizada para paliar los fallos transitorios ha sido la redundancia modular (NMR) con un valor de $N = 3$, conocida como redundancia modular triple (TMR). Consiste en triplicar cada componente de memoria y conectar «votadores» a sus salidas. Los «votadores» permiten enmascarar una cantidad de fallos igual a $\frac{N}{2}$. En nuestro caso significa que el sistema podrá tolerar un fallo en cada conjunto de módulos triplicados.

Para proporcionar al sistema tolerancia frente a los SEU al sistema, se han sustituido todos los biestables por un conjunto compuesto por tres biestables

más un votador. Si el componente inicial era el representado en la figura 5.1a, aplicando lo anterior obtendremos el conjunto representado en la figura 5.1b.



(a) Biestable original. (b) Conjunto de biestables con votador.

Figura 5.1: Sustitución de biestable.

5.3. El Votador

El votador es un circuito combinacional, y su único objetivo es filtrar los valores de entrada, para dar un valor de salida igual al de la mayoría de sus entradas. Esta propiedad hace del votador el componente principal utilizado para otorgar la capacidad de tolerar los fallos transitorios de tipo SEU al sistema.

Como se vé en la tabla 5.1, el valor de salida de un votador es igual al valor que más se repite en sus entradas. Con este método puede enmascararse un fallo en cualquiera de los módulos que alimentan sus entradas.

Entradas			Salida
A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tabla 5.1: Tabla de verdad del votador

La implementación del votador puede realizarse de diferentes formas siempre que cumpla con las restricciones de la tabla 5.1. El votador de este proyecto se ha diseñado siguiendo la fórmula lógica 5.1, utilizando 4 puertas lógicas: 3 puertas «AND» de dos entradas y 1 puerta «OR» de tres entradas. Como se puede observar en la figura 5.2, se introduce únicamente un retardo de 2 puertas lógicas, haciendo que las puertas lógicas «AND» funcionen en paralelo.

$$Z = (A * B) + (B * C) + (A * C) \tag{5.1}$$

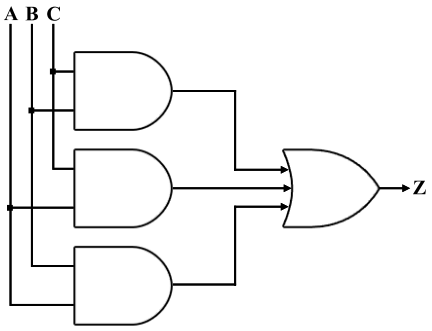


Figura 5.2: Diseño de votador con puertas lógicas

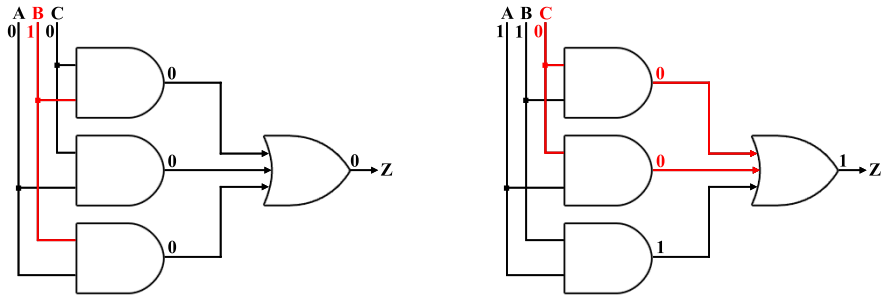
Para exponer cómo funciona el votador se muestran las figuras 5.3a y 5.3b. En la figura de la izquierda observamos cómo la entrada B es diferente al resto, eso quiere decir que el módulo origen de esa señal ha sufrido un fallo. Se observa cómo tal fallo queda enmascarado por las puertas «AND» por su funcionalidad $(0 * 1 = 1 * 0 = 0)$. En el caso de la figura derecha, el fallo ocurre en la entrada C, en este caso queda enmascarado por la puerta «OR»$(0 + 0 + 1 = 1)$.

5.3.1. Implementación

La implementación de este módulo se divide en dos secciones:

- **Registros triplicados**

  Al aplicar la la redundancia tipo TMR a los registros, estos deben triplicarse. Para ello se define una constante «N_Tolerancia» para establecer el número de replicas que se desea implementar, en nuestro caso este número es 3. Después se declara un conjunto de tres registros de un tamaño variable determinado por una variable. Esto facilita utilizar el componente para datos tanto de tamaño 32 como de tamaño 1. A continuación se declara un proceso por el cual se actualizan los tres registros al mismo tiempo y con los mismos datos.



(a) Fallo en entrada B (0 ->1) (b) Fallo en entrada C (1 ->0)

Figura 5.3: Ejemplos de fallos en entradas.

```
constant N_Tolerancia : integer := 3;
type tipo_conjunto_registros is array (0 to N_Tolerancia
    -1) of STD_LOGIC_VECTOR (tamaño-1 downto 0);
signal registros : tipo_conjunto_registros;

[...]

p_regs: process (clk, rst)
begin
   if rst='0' then
    for i in 0 to N_Tolerancia-1 loop
            registros(i) <= (others=>'0');
       end loop;
   elsif rising_edge(clk) then
       for i in 0 to N_Tolerancia-1 loop
            registros(i) <= dato_entrada;
       end loop;
   end if;
end process;
```

- **Votador**

  Aplicando la fórmula 5.1 a los registros triplicados, el votador obtiene el valor con un mayor número de repeticiones.

```
dato_salida <= (registros(0) and registros(1)) or
               (registros(0) and registros(2)) or
               (registros(1) and registros(2));
```

5.4. Aplicación

La segmentación del procesador requiere una cantidad importante de biestables para almacenar todos los datos que deben transmitirse de una

etapa a la siguiente. Debido al enorme aumento del número de lugares críticos donde puede ocurrir un SEU, se ha visto necesario aplicar la técnica TMR especialmente en esta parte del microprocesador.

Esta técnica se ha aplicado a todos los registros de control para evitar que un fallo no controlado produzca un error en medio de la ejecución de una instrucción en cualquiera de sus etapas. Por ejemplo: cambiando la dirección de un salto, cambiando el dato que debe almacenarse en memoria o habilitando la escritura en registro cuando no debería almacenarse el resultado de la instrucción.

Sin incluir las memorias de datos e instrucciones que normalmente son externas, el componente «registros de control» es el mayor elemento de memoria dentro del microprocesador. Un fallo en este componente altera de forma imprevisible la ejecución de las instrucciones que estén en ejecución dentro del procesador en ese momento, evitar este problema es el objetivo de este trabajo y por ello nos centraremos en modificar este componente para ofrecer un mayor grado de confiabilidad. En su caso, también se podría aplicar este método a otros componentes como el banco de registros y el registro «contador de programa».

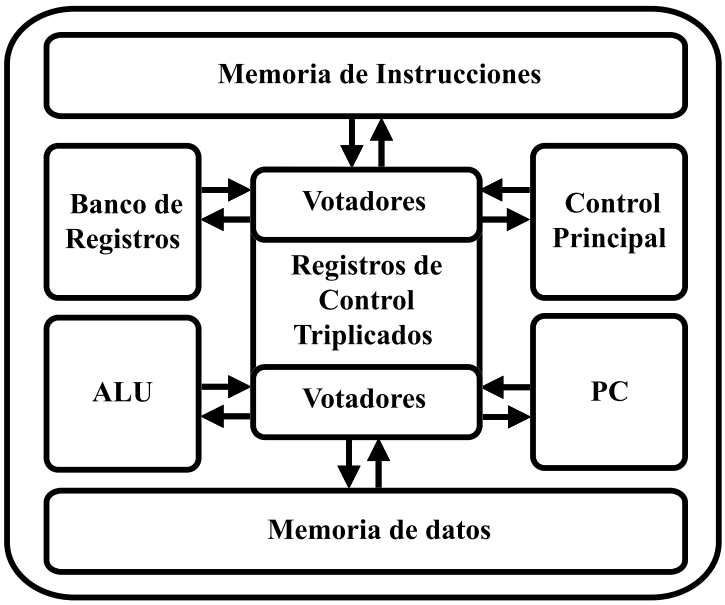


Figura 5.4: Procesador con TMR en los registros de control.

El resultado de aplicar este método sobre los registros de control se puede observar en la figura 5.4. Obtenemos un componente de mayor tamaño, pero que proporciona una mayor fiabilidad en que las instrucciones se ejecutarán correctamente.

Capítulo 6

Resultados

...

...

Resumen: En este capitulo se exponen los resultados obtenidos al implementar el procesador e introducir el sistema de tolerancia a fallos en el mismo.

6.1. Introducción

La herramienta software «PlanAhead» de Xilinx es capaz de sintetizar e implementar el diseño VHDL. Se utilizará esta herramienta para analizar los cambios que ocurren en el procesador al insertar los módulos de tolerancia a fallos. A continuación se analizan ambos diseños.

6.2. Procesador pre-TMR

6.2.1. Área

6.2.2. Tiempos

6.2.3. Simulaciones

Simulación sin fallos

Simulación con fallos

6.3. Procesador post-TMR

6.3.1. Área

6.3.2. Tiempos

6.3.3. Simulaciones

Simulación sin fallos

Simulación con fallos

Capítulo 7

Análisis de los resultados

...

...

Resumen: En este capitulo se exponen los resultados obtenidos al implementar el procesador e introducir el sistema de tolerancia a fallos en el mismo.

7.1. Introducción

Capítulo 8

Conclusiones

...

...

Resumen: En este capitulo se

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

[1] R. Brinkgreve, W. Swolfs, and E. Engin. *ARM Architecture Reference Manual Thumb-2 Supplement.* 2011.

[2] S. Brown and J. Rose. Architecture of FPGAs and CPLDs: A tutorial. *IEEE Design and Test of Computers*, 13(2):42–57, 1996.

[3] C. T. Bustillos. Simulador arm en el ámbito docente. 2012.

[4] I. N. de Estadistica. Penetración de ordenador en hogares. 2014.

[5] S. Flash. Nexys4 FPGA Board Reference Manual Ethernet connector. pages 1–29, 2013.

[6] J. Gaisler. A portable and fault-tolerant microprocessor based on the SPARC V8 architecture. *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 409–415, 2002.

[7] J. C. González Salas. *Filtro adaptativo tolerante a fallos.* PhD thesis, 2014.

[8] S. Habinc. Functional Triple Modular Redundancy (FTMR). *Design and Assessment Report, Gaisler Research*, pages 1–56, 2002.

[9] J. L. Hennessy and D. A. Patterson. *Arquitectura de Computadores: Un enfoque cuantitativo.* Mcgraw Hill Editorial, 1993.

[10] J. L. Hennessy and D. a. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach.* Number 0. 2006.

[11] A. C. Hu and S. Zain. NSEU Mitigation in Avionics Applications. 1073:1–12, 2010.

[12] O. Ieee-std. LEON3 7-Stage Integer Pipeline. (March), 2010.

[13] A. O. Investigation. ATSB TRANSPORT SAFETY REPORT Aviation Occurrence Investigation AO-2008-070 Final. (October), 2008.

[14] Jedec. Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Error in Semiconductor Devices: JESD89A. *JEDEC Sold State Technology Association*, pages 1–85, 2006.

[15] A. Kadav, M. J. Renzelmann, and M. M. Swift. Fine-grained fault tolerance using device checkpoints. *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS '13*, page 473, 2013.

[16] H. Kirrmann. Fault Tolerant Computing in Industrial Automation. *Lecture notes ABB Corporate ResearchETH*, 2005.

[17] I. Kuon, R. Tessier, and J. Rose. FPGA Architecture: Survey and Challenges. *Foundations and TrendsÂ® in Electronic Design Automation*, 2(2):135–253, 2007.

[18] A. R. M. Limited. ARM7TDMI-S. (Rev 3), 2000.

[19] a. R. M. Limited. ARM Architecture Reference Manual. pages 1–1138, 2007.

[20] W. K. Melis. *Reconstruction of High-energy Neutrino-induced Particle Showers in KM3NeT .* PhD thesis, 2014.

[21] C. Mobile. Streaming 4K Ultra HD video at home and on the go. pages 0–1.

[22] J. Rose, A. E. Gamal, and A. Sangiovanni-Vincentelli. Architecture of Field-Programmable Gate Arrays.

[23] E. Rotenberg. AR-SMT: a microarchitectural approach to fault tolerance in microprocessors. *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352)*, 1999.

[24] D. J. Sorin and S. Ozev. Fault Tolerant Microprocessors for Space Missions. *Memory*, pages 1–4.

[25] U. States. Reduce Cost and Board Space. 374:1–8, 2011.

[26] I. S. Summary, T. C. Field, M. Long, S. D. Transfer, U. Instruction, and I. S. Examples. ARM Instruction Set. pages 1–60.

[27] J. M. Torrecillas. RAID - Tolerancia a Fallos.

[28] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. *Proceedings of the International Conference on Dependable Systems and Networks*, (July):411–420, 2001.

[29] Xilinx. Xilinx Artix-7 Fpgas: a New Performance Standard for Power-Limited, Cost-Sensitive Markets.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes