Capítulo 4

Titulo por definir

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

Ernest Rutherford

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 tratan dos arquitectura más concretas, la arquitectura DLX y la arquitectura ARM.

A continuación se define qué es un fallo y qué tipos de fallos pueden ocurrir en los sistemas. Además se explican algunas técnicas de tolerancia a fallos.

Para terminar, se justifica la importancia de la tolerancia en los sistemas y concretamente porqué es necesaria la tolerancia en los microprocesadores.

4.1. 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 4.1.

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.

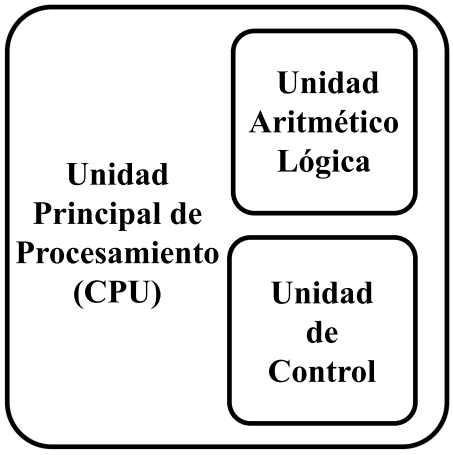


Figura 4.1: Procesador DRAE

4. Almacenar los resultados en memoria.

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

4.1.1. 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, y contiene todos los componentes la memoria y los elementos de entrada y salida [Hennessy y Patterson (1993)].

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 tales como realizar cálculos entre los datos en memoria y los datos en registro.

3. *Simple instruction set computer (SISC).* Utiliza un repertorio de instrucciones enfocado al procesamiento paralelo.

4.1.2. 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 puede aplicar 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.

4.1.3. 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». Estos registros de propósito general son muy limitados. Por ello el procesador necesita de apoyo externo para alojar la información, para lo que tiene acceso a una memoria externa.

El acceso a la memoria externa divide las arquitecturas en dos tipos. La arquitecturas Von Neumann utiliza una única memoria para almacenar tanto los datos como las instrucciones. Las arquitecturas Harvard, sin embargo, separan la memoria de datos de la memoria de instrucciones. Figura 4.2.

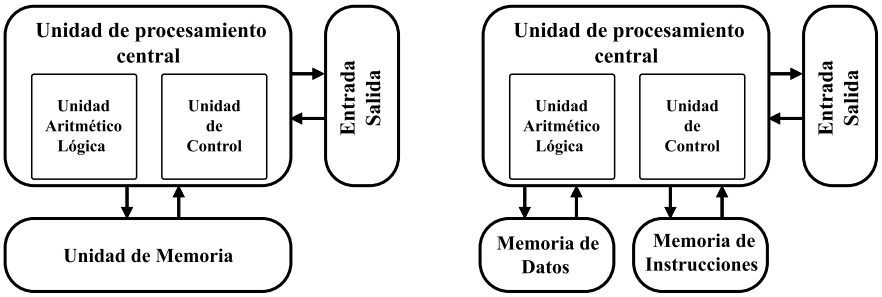


Figura 4.2: Arquitectura Von Neumann y Arquitectura Harvard

4.1.4. Segmentación

La segmentación es una técnica de implementación, que no siendo imprescindible, aumenta el rendimiento del procesador. Permite que haya varias instrucciones en ejecución al mismo tiempo en el mismo procesador. El procesador es dividido en etapas y en cada una de ellas se realiza una parte del trabajo completo de la instrucción de forma secuencial.

La segmentación permite que en cada ciclo de reloj se busque una instrucción y se comience su ejecución, con ello se consigue reducir el número de ciclos total que necesita el programa. [Hennessy y Patterson (1993)].

En la tabla 4.1 podemos ver cómo se lanzan una serie de instrucciones. Se observa cómo las instrucciones ocupan unicamente una etapa del procesador, y cómo avanzan por el procesador dejando libre la etapa anterior para 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 4.1: Segmentacion simple de 5 etapas

Ventajas de la segmentación

La segmentación proporciona la ventaja de poder lanzar una instrucción por cada ciclo de reloj. Esta característica 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.

$$\text{Ciclos por instrucción (CPI)} = \frac{\text{Número de ciclos total}}{\text{Número de instrucciones}} \tag{4.1}$$

A modo de ejemplo, veamos que sucede al ejecutar un programa de 3 instrucciones sobre un procesador que tarde 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.

Como podemos ver en la figura 4.3, el procesador no segmentado tarda 15 ciclos en ejecutar las 3 instrucciones y utilizando la formula anterior se obtiene un valor de CPI es 5. Al ejecutar el mismo programa en el procesador segmentado, este tarda 5 ciclos en llenar las 5 etapas del procesador. A partir de ahora cada ciclo de reloj termina una instrucción, completandose 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.

Riesgos de la segmentación

Además de las ventajas vistas en el apartado anterior, la segmentación también implica unos riesgos a la hora de ejecutar las instrucciones. Estos riesgos implican que las instrucciones deban esperar un número de ciclos para poder continuar su ejecución, retrasando la entrada de instrucciones en

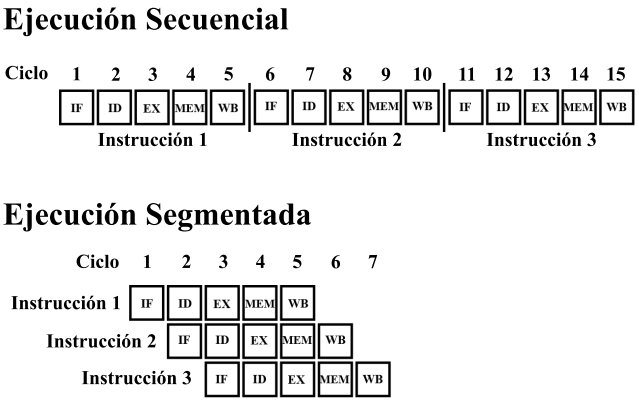


Figura 4.3: Ejecución secuencial comparada con ejecución segmentada

el microprocesador. Estos riesgos pueden ser de los siguientes tipos[Hennessy y Patterson (1993)]:

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

2. *Riesgos de datos.* Surgen cuando 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 instrucciones de auto-incremento de direccionamiento.
   - *Escritura después de escritura (WAW).* Ocurre cuando las escrituras se realizan en orden incorrecto. Por ejemplo la *instrucción i+1* escribe su resultado antes de que lo haga la *instrucción i*, ambas escriben en el mismo registro.
   - *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).

4.1.5. DLX

El microprocesador DLX fue diseñado por John Hennessy y David A. Patterson, diseñadores de las arquitecturas MIPS y Berkeley RISC respectivamente. Es un procesador sencillo con arquitectura RISC y proporciona una base fácil de comprender. Se utiliza ampliamente en educación universitaria para explicar las arquitecturas de computadores [Pascual (2011)].

Basado en las máquinas de carga/almacenamiento, el DLX se centra en proporcionar[Hennessy y Patterson (1993)]:

- Un sencillo repertorio de instrucciones de carga/almacenamiento.
- Un diseño de segmentación eficiente.
- Un repertorio de instrucciones fácil de descodificar.

Arquitectura RISC

El microprocesador DLX utiliza una arquitectura RISC con instrucciones de 32 bits. Posee un banco de registros compuesto por 32 registros de propósito general, además de un segundo conjunto de registros que se pueden usar como 32 registros de simple precisión o como 16 registros en punto flotante.

Instrucciones DLX

Todas las instrucciones del repertorio del procesador DLX tienen un tamaño de 32 bits y están alineadas en memoria. En cualquier instrucción los bits [31:26] forman el campo de código de operación que se debe ejecutar.

Se dividen en tres tipos según su formato [Arnau Llombart]:

- *Tipo R.* Instrucciones aritmético-lógicas.
- *Tipo I.* Instrucciones de transferencia.
- *Tipo J.* Instrucciones de bifurcación.

Como podemos observar en la figura 4.4, el formato es muy similar en los tres tipos, lo que reduce la ruta de datos, simplificando su implementación.

Segmentación DLX

El DLX basa su rendimiento en la segmentación y se divide en 5 etapas

- *Búsqueda de la instrucción (IF)*

  Esta primera etapa es la encargada de acceder a memoria y traer la siguiente instrucción.

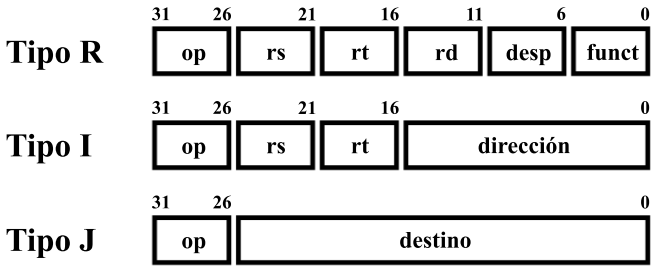


Figura 4.4: Formato de instrucciones DLX

- *Descodificación de la instrucción (ID)*

  En la segunda etapa se descodifica la instrucción cargada en la primera etapa, obteniendo las señales de control. Extrae los operandos del banco de registro o de la propia instrucción.

- *Ejecución y cálculo de direcciones efectivas (EX)*

  La tercera etapa se encarga de ejecutar la instrucción utilizando las unidades funcionales. Las unidades funcionales pueden estar segmentadas y/o duplicadas.

- *Acceso a memoria (MEM)*

  En la etapa de memoria es cuando se ejecutan las operaciones de carga y almacenamiento. Las instrucciones de carga traen datos de la memoria y los almacenan en los registros, mientras que las instrucciones de almacenamiento guardan los datos en memoria.

- *Postescritura (WB)*

  En la última etapa se almacenan los resultados de las instrucciones en los registros.

Como pudimos ver en la figura 4.1, las instrucciones se buscan en cada ciclo de reloj, a menos que surjan riesgos debido a la segmentación. Como vimos en el apartado 4.1.4, la segmentación implica ciertos riesgos. Para solucionar o reducir estos, el DLX implementa las siguientes técnicas[Hennessy y Patterson (2006)]:

1. Duplicar y/o segmentar las unidades funcionales. Con ello se reducen los ciclos de espera debidos a los *riesgos estructurale*s. Se consigue un mayor número de etapas para poder cargar nuevas instrucciones.

2. «Adelantamiento»(forwarding) o «Cortocircuito». Técnica encaminada a resolver los *riesgos de datos*. Se consigue proporcionar un acceso a los resultados de instrucciones previas que todavía no han almacenado los datos en el «banco de registros».

    - *Lectura después de escritura (RAW)*. Debido al cortocircuito implementado, el dato es recibido de las etapas siguientes y no es necesario que se haya escrito en los registros.
    - *Escritura después de lectura (WAR)*. No puede ocurrir debido a que todas las lecturas se realizan al comienzo de la ejecución, en la etapa de descodificación, y las escrituras al final, en la etapa de postescritura.
    - *Escritura después de escritura (WAW)*. Solo se presenta en segmentaciones que escriben en más de una etapa, esta arquitectura no se ve afectada ya que solo escribe en la etapa de postescritura. puede ocurrir de

3. *Riesgos de control*. Si se ejecuta una instrucción de salto, el cambio no se ve reflejado hasta la fase de memoria, esto implica una detención de 3 ciclos. En DLX se utiliza lógica especializada para averiguar si el salto es efectivo y para la calcular la dirección destino de salto en la etapa de descodificación.

Cuando existe un riesgo que no es posible evitar con estas técnicas se aplica un interbloqueo de la segmentación. Esta técnica detecta un riesgo y detiene la ejecución de la instrucción hasta que el riesgo desaparece. El bloqueo se realiza en la fase de decodificación, donde es posible determinar si existe algún riesgo.

Memoria DLX

Todos las referencias a memoria se realizan a través de instrucciones de carga y almacenamiento, cargando los datos en los registros, donde se pueden acceder y trabajar con ellos, y almacenándolos en memoria.

El acceso a memoria es direccionable por bytes en el modo «Big endian» con una dirección de 32 bits almacenada previamente en un registro. Los accesos pueden realizarse a un byte, media palabra o una palabra completa. Además se puede acceder a palabras en doble precisión para almacenarlas en los registros de punto flotante.

4.1.6. ARM

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

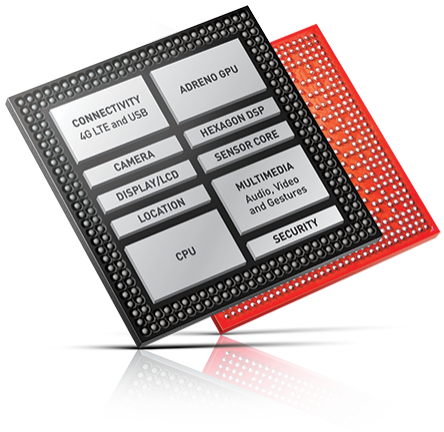


Figura 4.5: Procesador Qualcomm Snapdragon 810

Actualmente la arquitectura ARM es el conjunto de instrucciones más ampliamente 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.

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 Bustillos (2012). 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, controlados de pantalla, conectividad y cámara entre otros, y todo ello en un único chip Mobile. Figura 4.5.

Además de requerir poco espacio, los dispositivos ARM están diseñados con el objetivo de minimizar la energía consumida. Haciéndolo apropiado para sistemas móviles empotrados 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, son opcionales.

Todo esto no impide que la arquitectura ARM sea muy eficiente y proporcione un alto rendimiento.

Arquitectura ARM

La arquitectura ARM Limited (2007) deriva de la arquitectura RISC con las características propias de esta:

- Banco de registros uniforme.

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

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:

- Control adicional sobre la unidad aritmético-lógica.
- Auto-incremento y auto-decremento para el direccionamiento.
- Instrucciones de carga y almacenamiento múltiple.
- Ejecución condicional de instrucciones.

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 registros tienen un uso especial y son el «puntero de pila (SP)», el «registro de enlace (LR)» y el «contador de programa (PC)».

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 como en 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 contenido 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 Summary et al., el set Thumb/ Thumb-2 Brinkgreve et al. (2011) y las instrucciones Jazelle Summary et al..

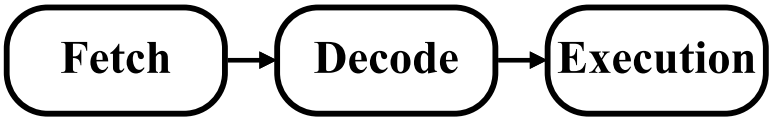


Figura 4.6: Segmentación ARM

Segmentación ARM

La arquitectura ARM 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. Limited (2000)

Las tres etapas en las que se divide la segmentación son: (Figura 4.6)

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.

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. Limited (2000)

4.2. 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.

4.2.1. 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 4.7.

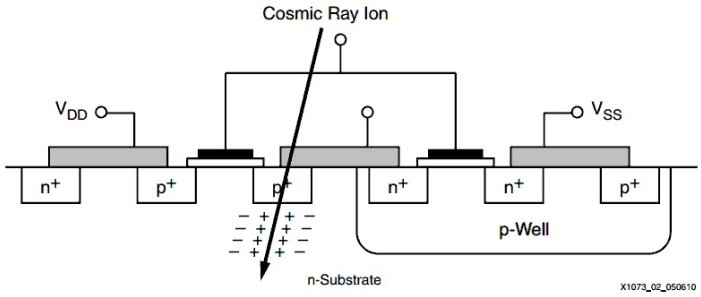


Figura 4.7: Single Event Upset en una FPGA[Hu y Zain (2010)].

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 4.8. 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 Melis (2014).

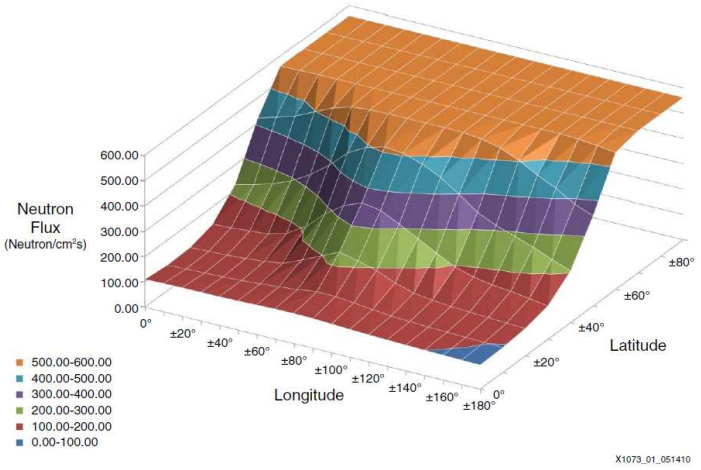


Figura 4.8: Flujo de neutrones a 40.000 pies de altitud[Hu y Zain (2010)].

4.2.2. 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 [Jedec (2006)]:

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

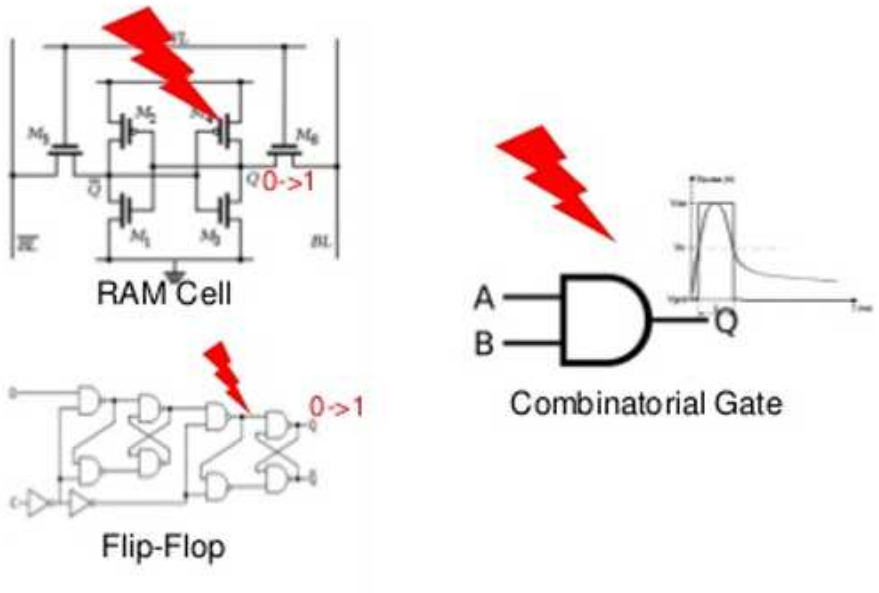
  Aquellos fallos que afectan a los elementos del chip invirtiendo su valor: memoria, celdas de memoria o registros. En un microprocesador se pueden corromper los datos del banco de registro, o los datos y las señales de control entre las etapas de segmentación. Figura 4.9a.

- **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 la 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 4.9b.



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

Figura 4.9: 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. [Jedec (2006)].

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 4.3.3.

4.3. 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 4.10 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».

- **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 4.11.

- **Enmascarado temporal**

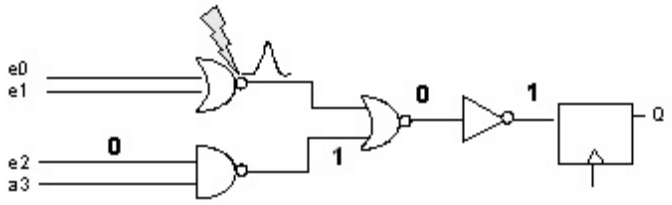


Figura 4.10: Fallo enmascarado por una puerta lógica.

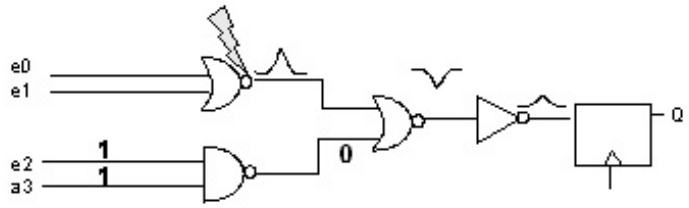


Figura 4.11: Fallo enmascarado eléctricamente.

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 4.12.

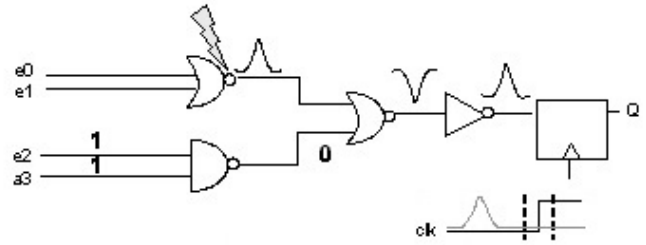


Figura 4.12: Fallo enmascarado por ventana de tiempo.

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:

4.3.1. 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 [Torrecillas]. 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.

4.3.2. 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 Kadav et al. (2013).

4.3.3. 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.

El 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.

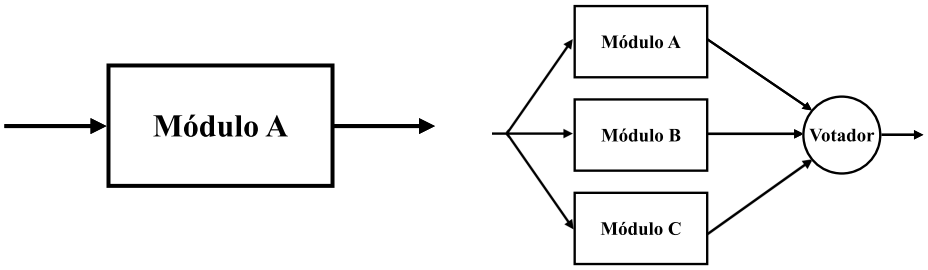
Algunas técnicas se detallan a continuación Kirrmann (2005).

Redundancia modular

La redundancia modular consiste en replicar N veces el bloque, 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 NMR es posible corregir los fallos producidos en N div 2 de los módulos redundantes. El votador que recibe las salidas de los módulos es el encargado de enmascarar los fallos.

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 4.13 se observa el resultado de aplicar la TMR a un bloque.



(a) Módulo original (b) Módulo con TMR

Figura 4.13: 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 com-

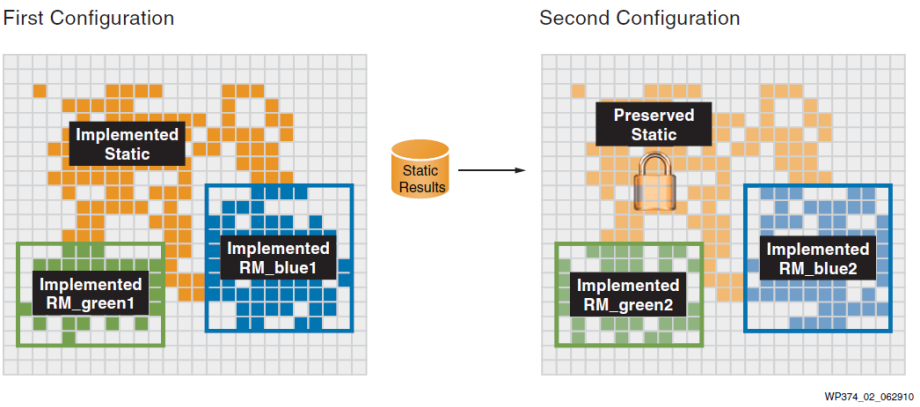


Figura 4.14: Tolerancia dinámica parcial States (2011).

binacional, 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 **??** de este mismo capitulo se han introducido las FPGAs, sistemas re-programables que permiten al diseñador re-configurar su estructura para realizar diferentes tareas, o la misma con una nueva implementación u organización de los componentes.

En la imagen 4.14 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 González Salas (2014):

- **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.

4.3.4. Tolerancia en microprocesadores

...

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

Arnau Llombart, V. Manual DLX. ????

Brinkgreve, R., Swolfs, W. y Engin, E. *ARM Architecture Reference Manual Thumb-2 Supplement*. 2011. ISBN 9781597180948.

Bustillos, C. T. Simulador arm en el ámbito docente. 2012.

González Salas, J. C. *Filtro adaptativo tolerante a fallos*. Tesis Doctoral, 2014.

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

Hennessy, J. L. y Patterson, D. A. *Arquitectura de Computadores: Un enfoque cuantitativo*. Mcgraw Hill Editorial, 1993. ISBN 1558600698.

Hennessy, J. L. y Patterson, D. a. *Computer Architecture, Fourth Edition: A Quantitative Approach*. 0. 2006. ISBN 0123704901.

Hu, A. C. y Zain, S. NSEU Mitigation in Avionics Applications. vol. 1073, páginas 1–12, 2010.

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

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

Kadav, A., Renzelmann, M. J. y Swift, M. M. Fine-grained fault tolerance using device checkpoints. *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS '13*, página 473, 2013. ISSN 15232867.

KIRRMANN, H. Fault Tolerant Computing in Industrial Automation. *Lecture notes ABB Corporate ResearchETH*, 2005.

LIMITED, A. R. M. ARM7TDMI-S. (Rev 3), 2000.

LIMITED, A. R. M. ARM Architecture Reference Manual. páginas 1–1138, 2007.

MELIS, W. K. *Reconstruction of High-energy Neutrino-induced Particle Showers in KM3NeT* .. Tesis Doctoral, 2014.

MOBILE, C. Streaming 4K Ultra HD video at home and on the go. páginas 0–1, ????

PASCUAL, J. M. *Simulador DLX con repertorio multimedia*. Tesis Doctoral, Universidad Complutense de Madrid, 2011.

SADASIVAN, S. An introduction to the arm cortex-m3 processor. 2006.

STATES, U. Reduce Cost and Board Space. vol. 374, páginas 1–8, 2011.

SUMMARY, I. S., FIELD, T. C., LONG, M., TRANSFER, S. D., INSTRUCTION, U. y EXAMPLES, I. S. ARM Instruction Set. páginas 1–60, ????

TORRECILLAS, J. M. RAID - Tolerancia a Fallos. ????