eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Escuela de Ingeniería de Bilbao

Departamento de Tecnología Electrónica

TESIS DOCTORAL.

Caracterización de la tolerancia a fallos de circuitos implementados en FPGAs

Autor: Igor Villalta Bustillo

Director: Unai Bidarte Peraita

Bilbao, Abril 2019

Agradecimientos

Aunque sea imposible devolver todo el apoyo recibido durante mi trayectoria como doctorando, en estas líneas voy a dedicar unas palabras a las personas que han mostrado su apoyo durante este tiempo.

En primer lugar quisiera agradecer al grupo de investigación APERT por haber confiado en mí y haberme dado la oportunidad de realizar esta tesis con ellos.

A mi director Unai Bidarte, por todas las buenas ideas aportadas y por su apoyo unánime durante cada una de las etapas de esta tesis.

A los profesores del área de digital del grupo por su ayuda y su contribución a este trabajo: Armando Astarloa, Jesús Lázaro, Aitzol Zuloaga, Jaime Jiménez, Carlos Cuadrado y José Luis Martín.

A la empresa SoC-e por cederme material determinante para el desarrollo de la tesis, y en especial a David Esteban y a Enekoitz Ormaetxea por su ayuda para la implementación.

Al grupo humano que conforma el grupo de investigación APERT, por el inmejorable ambiente de trabajo del que he podido disfrutar. Además de los anteriormente mencionados, quisiera agradecer a los investigadores Asier, Itxaso, David, Marcelo, Tatiana, Iker, Endika, Markel, Julen, Víctor, Oier, Iraide, Naiara, Ángel y Uli; y a los profesores del área de potencia del grupo: Jon, Iñigo Kortabarria, Iñigo Mtz de Alegría, Edorta, Estefanía e Iñaki.

A mi familia, a mis padres y mi hermano por su apoyo incondicional durante toda la duración de esta tesis. Por animarme y apoyarme en todo momento a pesar de no comprender muy bien su contenido. También me quiero acordar de mi abuela y de todos mis tíos y primos.

El trabajo descrito en esta publicación ha sido financiado por el Departamento de Educación del Gobierno Vasco en base a las ayudas para apoyar las actividades

de grupos de investigación del sistema universitario vasco IT978-16 y también por el Ministerio de Economía y Competitividad de España a través del proyecto de investigación TEC2017-84011 y los fondos FEDER.

Abstract

FPGAs (Field-Programmable Gate Arrays) and SoCs (Systems-on-chip) based on this technology, are electronic devices that can be configured in field, offering the possibility of developing a customized circuit in a short time-to-market with lower design costs than ASICs. Due to the continuous improvements on device performance, safety-oriented specific sectors such as automotive, railway, industrial, avionics aerospace and others, have shown interest in these devices. For these industries, the appliance of methods to estimate the system failure rate is mandatory, due to safety regulations. The problem lies in the fact that FPGAs are susceptible to radiation-induced SEUs (Single Event Upsets) in the configuration memory, a type of error that causes the random modification of one or more bits of this memory, affecting the implemented circuit.

Therefore, reliability-oriented designs using commercial FPGAs must consider applying reliability mechanisms to mitigate SEUs. In addition to this, evaluation techniques are also necessary to corroborate the effectiveness of those strategies. Among the different applicable evaluation procedures, SEU emulation stands out. It consists of programming the device with an intentionally corrupted file, so that erroneous content is stored in the configuration memory, generating an effect analogous to SEU. Different emulation methodologies have been studied in the literature and a series of deficiencies have been observed. On the one hand, internal emulation methods (errors are injected from inside the FPGA itself) have the problem of being self-blocking, since the injected error can affect the emulation system itself. On the other hand, external emulation systems may require major changes at hardware level.

The main objective of this thesis is the development of a SEU emulation mechanism able to be straightforwardly implemented in an already built system, whose unique requirement is containing a SoC FPGA such as Zynq or similar. Moreover, the proposed method intends to solve the deficiencies observed in the literature, taking advantage of the capacities offered by these SoCs. To this end, the error

injection system has been proposed to be placed in the PS in order to prevent blocking injections. Although injections are made from outside the FPGA, they are carried out from inside the chip itself, avoiding the need of hardware modifications. A universal verification scheme has been proposed, so that the test can be adapted to different systems in a straightforward way.

Once the emulation methodology has been described, two more contributions have been realised by this work. First, it has been analyzed the influence of different decisions taken during design process on the failure rate. Here, it has been proven that the failure rate in a concrete design can have fluctuations of up to 50 % when some parameters are modified. On the other hand, having observed that the emulators of SEUs existing in the literature are focused on the study of SBUs (Single Bit Upsets), a procedure has been proposed for the estimation of the failure rate in the presence of MCUs (Multiple Cell Upsets).

Laburpena

FPGAk (Field Programmable Gate Array) eta FPGAetan oinarritutako SoCak (System-on-Chip) eremuan programagarriak diren txipak dira, zirkuitu pertsonalizatuak neurrira garatzeko aukera eskaintzen dutenak. ASICekin (Application Specific Integrated Circuit) konparatuz gero, nabarmendu beharra dago gailu horiek merkaturatze-denbora azkartzen dutela, diseinuaren kostua txikiagoa izanik. Transistoreen dimentsioen etengabeko murrizketari esker, gailu horien konputazio-ahalmena nabarmen handitu da azken hamarkadetan, beraien erabilera automozioan, aeronautikan, trenetan eta aplikazio espazialetan zabalduz. Sektore horietan derrigorrezkoa da diseinuak fidagarritasunera eta segurtasunera bideratzea, onargarriak diren arrisku-tasak araudi zorrotzetan espezifikatuta baitaude. Hori dela eta, FPGAetan inplementatutako zirkuitoen hutsegite-tasen neurketarako prozedurak zehaztu behar dira. Erradiazioak induzitutako SEUak (Single Event Upset) dira gailu horietan gertatzen diren hutsegiteen arduradun nagusiak; hori da, memoria batetako bit baten (edo batzuen) bat-bateko aldaketa. Gertaera horiek FPGAen konfigurazio-memorian gertatuz gero, inplementatutako zirkuitu mailan hardware erroreak sortzen dituzte.

Hori guztia kontutan hartuta, FPGA komertzialak erabiltzen dituzten fidagarritasunera bideratutako diseinuek SEUaren eragina murrizteko estrategiak erabili behar dituzte. Ez hori bakarrik, estrategia horien eraginkortasuna balioztatu behar da, hutsegite-tasa behar bezain txikia dela justifikatzeko. Ebaluazio-prozedura guztien artean, SEUen emulazioa nabarmentzen da. Horren funtsa da gailua akatsak dituen konfigurazio-fitxategi batekin programatzea, konfigurazio-memorian SEU baten eragina emulatzeko. Literatura zientifikoan SEUen emulaziorako aurkeztutako prozedura ezberdinak aztertu ondoren, gabezia batzuk antzeman dira. Alde batetik, FPGAk berak injekzioak bideratzekotan, emulazio sistema blokeatzeko arriskua dago, injektatutako erroreak berak emulazio-sistema hondatu dezakelako. Bestalde, FPGAtik kanpo bideratu nahi bada emulazioa, hardwareari dagozkion aldaketa nabarmenak beharrezkoak dira.

Tesi honen helburu nagusia da SEU emulazio-metodologia baten deskribapena, gaur egun existitzen diren sistemetan modu erraz batean inplementatzeko gaitasuna duena. Proposatutako prozeduraren muga bakarra da sistemak FPGA SoC bat, Zynq edo bestelakoa, izan behar duela. Horrez gain, literaturan antzemandako gabeziak gainditzen ditu proposatutako metodoak, FPGA eta prozesadorea (PS) konbinatzen duten SoCen ezaugarriez baliatuz. Horretarako, erroreen injekziorako sistema PSan kokatzen da, errore-injekzio blokeatzaileak ekiditzeko. Horrela, injekzioak FPGAtik kanpo egiten diren arren, txiparen barrutik gauzatzen dira; beraz, PCBan hardware-aldaketak ez dira beharreskoak. Egiaztapen-eskema unibertsala proposatu da, konplexutasun ezberdineko zirkuituetara modu sinple batetan egokitzeko ahalmena duena. Horrez gain, beste bi ekarpen burutu dira tesi honetan. Alde batetik, egiaztatu da diseinu-fasean hartutako erabaki ezberdinen eragina hutsegite-tasan, eta baita ere diseinu berdinak parametro ezberdinekin % 50eko gorabeherak izatera heldu daitekeela. Bestalde, literaturan aztertutako emulazio-metodoek SBU (Single Bit Upset) motako SEUak aztertzen dituzte, eta lan honetan MCU (Multiple Cell Upset) motakoen azterketarako prozedura bat proposatu da ere.

Resumen

Las FPGAs (Field-Programmable Gate Array) y los SoC (System-on-chip) basados en FPGA son dispositivos electrónicos configurables en campo (in field), que ofrecen la posibilidad de desarrollar un circuito a medida con un tiempo de salida al mercado y unos costes de diseño reducidos en comparación con los ASICs. Debido a la reducción continua del tamaño de los transistores, las prestaciones de estos dispositivos se están incrementando de manera vertiginosa en las últimas décadas, lo que ha generado interés en sectores muy específicos como automoción, ferroviario, industrial, aviónico o aeroespacial. En estos sectores se exige que los diseños estén orientados a confiabilidad y que cumplan con diversas normativas de seguridad, lo que requiere de métodos para la estimación y justificación de la tasa de fallos del sistema. El problema radica en que las FPGAs son especialmente susceptibles al SEU (Single Event Upset) generado por radiación en la memoria de configuración, un tipo de error que provoca la modificación aleatoria de uno o más bits de dicha memoria, afectando al circuito implementado.

Por lo tanto, los diseños orientados a confiabilidad que utilicen FPGAs comerciales han de considerar la inclusión de una serie de medidas y mecanismos para mitigar sus efectos. No solo eso, sino que también es necesaria la aplicación de mecanismos de evaluación para corroborar que las estrategias aplicadas permiten alcanzar los objetivos de confiabilidad. De entre los diferentes procedimientos de evaluación aplicables se destaca la emulación de SEUs, que consiste en programar el dispositivo con un archivo intencionadamente corrompido para que se almacene contenido erróneo en la memoria de configuración, lo que genera un efecto análogo al SEU. Se han estudiado diferentes metodologías de emulación en la literatura y se han observado una serie de deficiencias. Por un lado, los métodos de emulación internos (los errores se inyectan desde la propia FPGA) tienen el problema de ser autobloqueantes, ya que el error inyectado puede afectar al propio sistema de emulación. Por otro lado, los sistemas de emulación externos pueden requerir cambios importantes a nivel de hardware.

El objetivo principal de este trabajo es el desarrollo de un mecanismo de emulación de SEUs que pueda implementarse de manera sencilla en sistemas ya construidos, cuyo único requisito es que dicho sistema tenga un SoC FPGA del tipo Zynq o similar. Además, se pretenden solventar las deficiencias observadas en la literatura aprovechando las diferentes capacidades que ofrecen los SoCs que combinan FPGA y sistema procesador (PS). Para ello se ha planteado la implementación del sistema de inyección de errores en el PS, ya que de esta manera se previenen las inyecciones de errores bloqueantes. De igual modo, aunque las inyecciones de realicen desde fuera de la FPGA, las inyecciones se llevan a cabo desde el interior del propio chip, evitando la necesidad de añadir modificaciones en el hardware. Se ha propuesto un esquema de verificación universal independiente de la aplicación, de modo que el esquema de test pueda ser adaptado a diferentes sistemas de forma sencilla, independientemente de su complejidad.

Una vez planteada la metodología de emulación, se han realizado otras dos aportaciones. En primer lugar se ha comprobado cómo afectan las diferentes decisiones que puedan tomarse en las diferentes etapas de la fase de diseño. Aquí se ha comprobado que un mismo diseño puede tener fluctuaciones de hasta el 50 % si se modifican algunos parámetros. Por otro lado, habiendo observado que los emuladores de SEU existentes en la literatura se centran en el estudio del SBU (Single Bit Upset), se ha propuesto un procedimiento para la estimación de la tasa de fallo en presencia de MCUs (Multiple Cell Upsets).

Organización del documento

Este documento consta de dos partes. La primera de ellas se corresponde con el estado del arte, y analiza los aspectos fundamentales para la comprensión de la emulación de SEUs en FPGAs. Se parte de los conceptos más genéricos de las FPGAs y de confiabilidad y de seguridad. Posteriormente se hace una revisión de las diferentes posibilidades para la obtención de sistemas confiables en FPGAs, y se finaliza analizando los conceptos concretos sobre emulación de SEU en FPGAs.

Por otro lado, la segunda parte se centra en las aportaciones derivadas de este trabajo. El capítulo 6 presenta el método de emulación propuesto. En el capítulo 7 se realiza el estudio del impacto de las diferentes fases de diseño en la tasa de fallo, y en el capítulo 8 en el estudio de los eventos múltiples. Por último, se resumen las conclusiones y aportaciones principales en un capítulo final.

Contenido

Lista de Figuras

Lista de Tablas

Tabla de Acrónimos

AR	*Availability Ratio*
ASIC	*Application Specific Integrated Circuit*
BC	*Bits Críticos*
BRAM	*BlockRAM*
CAD	*Computer Asisted Design*
CB	*Connection Block*
CDE	*Cross Domain Errors*
CLB	*Configurable Logic Block*
CLK	*Clock*
CMOS	*Complementary Metal-Oxide Semiconductor*
COTS	*Commercial Off-The-Shelf*
CRAM	*Configuration RAM*
CRC	*Cyclic Redundancy Check*
DDR	*Double Data Rate*
DMR	*Double Modular Redundancy*
DWC	*Dupplication With Comparison*
ECC	*Error Correction Code*
EDAC	*Error Detection and Correction*
EMIO	*Extended Multiplexed I/O*
FAR	*Frame Address Register*
FMEA	*Failure Mode and Effect Analysis*
FPGA	*Field Programmable Gate Array*
FR	*Failure Rate*
FTA	*Fault Tree Analysis*
HLS	*High Level Synthesis*
ICAP	*Internal Configuration Access Port*

IDF	*Isolated Design Flow*
IOB	*Input-Output-Block*
JTAG	*Joint Test Action Group*
LUT	*Look-Up Table*
MBU	*Multiple Bit Upset*
MCU	*Multiple Cell Upsets*
MRAM	*Magnetoresistive RAM*
MTBF	*Mean Time Between Failures*
MTTF	*Mean Time To Fail*
MTTR	*Mean Time To Repair*
OTP	*One Time Programmable*
PCAP	*Processor Configuration Acces Port*
PCB	*Printed Circuit Board*
PIP	*Programmable Interconnection Point*
PL	*Programmable Logic*
PS	*Processing System*
SB	*Switching Block*
SEE	*Single Event Effect*
SEFI	*Single Event Functional Interrupt*
SET	*Single Event Transient*
SEU	*Single Event Upset*
SRAM	*Synchronous Random-Access Memory*
TDDB	*Time-Dependent Dielectric Breakdown*
TID	*Total Ionizing Dose*
TMR	*Triple Modular Redundancy*
XSG	*Xilinx System Generator*

Parte I

Estado del arte

Capítulo 1

Introducción a las FPGAs

En este capítulo se introducen los conceptos básicos de las FPGAs. Como punto de partida se toma la descripción de la arquitectura general de estos dispositivos, continuando con una retrospectiva histórica para valorar la evolución que han tenido desde su nacimiento hasta la actualidad. Posteriormente, se analizan las diferentes etapas para el desarrollo de circuitos en FPGAs, así como un análisis pormenorizado de los dispositivos de Xilinx de la serie 7, ya que el trabajo se va a desarrollar sobre un dispositivo de esta familia. El capítulo finaliza con un análisis del proceso de configuración de las FPGAs de Xilinx. La relevancia del presente capítulo radica en que, a lo largo de este capítulo se van a introducir múltiples conceptos que van a ser referenciados habitualmente durante el resto de la tesis.

1.1 Estructura de las FPGAs

Una FPGA (Field Programmable Gate Array) es un componente electrónico estándar con capacidad de ser eléctricamente programado en campo para que adquiera la funcionalidad de cualquier circuito digital. Sus principales ventajas son gran flexibilidad, tiempo de salida al mercado relativamente corto y coste bajo para volúmenes de producción pequeños y medianos. En contraposición con los ASICs (Application Specific Integrated Circuits), las FPGAs tienen un coste unitario mayor, y un rendimiento inferior en cuanto a consumo de potencia, área de silicio y consumo de potencia. Sin embargo, el desarrollo de un ASIC requiere una inversión inicial elevada, tanto económica como en tiempo y esfuerzo de

desarrollo, haciendo necesarios grandes volúmenes de producción para recuperar dicha inversión.

Los elementos básicos de los que se compone una FPGA son las **celdas lógicas** y los **recursos de rutado**. Estos elementos son replicados en el dispositivo de forma regular para poder así implementar cualquier tipo de circuito electrónico digital en el dispositivo. El tercer elemento común en cualquier FPGA es la **memoria de configuración**, en la que se almacenan los datos de configuración de los diferentes recursos contenidos en el dispositivo. La presente sección se centra en el análisis de estos tres elementos.

1.1.1 Celdas lógicas

Las celdas lógicas son los recursos de la FPGA sobre los que se implementan las diferentes funciones booleanas de las que consta el diseño. La implementación más común de la lógica combinacional es mediante LUTs (Look-Up Tables). Hasta ahora, las LUTs de 4 entradas han sido muy recurridas para la implementación de funciones lógicas de 4 variables booleanas. Éstas son memorias de 16 bits que proporcionan un valor almacenado para las diferentes combinaciones de valores en las entradas. Además, estas celdas contienen uno o varios flip-flops, generalmente de tipo D, que se emplean como base de los circuitos secuenciales que puedan implementarse. Adicionalmente, cada celda contiene elementos de rutado internos (generalmente multiplexores) que sirven para encaminar las señales a los flip-flops, a las LUTs, a las celdas contiguas o a líneas de rutado externas de diferentes prioridades.

Estos elementos pueden observarse en la figura 1.1, donde se representa la estructura de una celda lógica básica. El objeto de la figura es meramente ilustrativo, ya que no sigue la estructura concreta de ninguna FPGA del mercado. Las señales que llegan a la celda son encaminadas hacia los diferentes elementos (lógica combinacional, flip-flops, salidas, etc) mediante multiplexores internos de rutado. Para la implementación de circuitos exclusivamente combinacionales, los multiplexores se configuran de forma que los flip-flops sean ignorados. Mientras tanto, los flip-flops se toman como base para la implementación de circuitos secuenciales como registros, contadores y máquinas de estados.

1.1.2 Recursos de rutado

Las FPGAs se postulan como plataformas capaces de implementar cualquier tipo de circuito digital, por lo que su arquitectura interna de rutado debe ser muy fle-

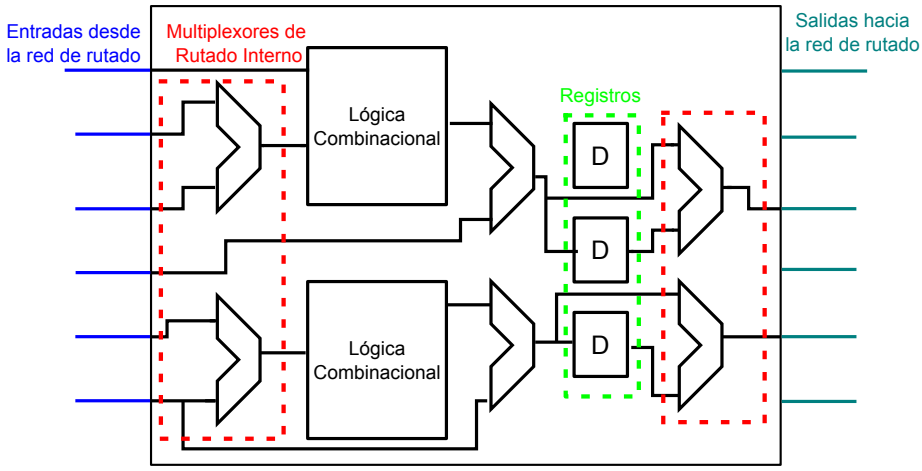


Figura 1.1: Esquema tipo de la celda lógica en FPGAs

xible para abarcar un rango de aplicaciones muy diverso. Sin embargo, los tipos de aplicaciones más comunes requieren dos tipos de línea de rutado: las líneas de rutado locales, que son líneas cortas para interconectar celdas lógicas contiguas o muy cercanas, y las líneas de rutado de alta prioridad, que se emplean para la transmisión de información entre bloques lejanos entre sí, o hacia los pines del dispositivo. Las FPGAs actuales contienen una gran cantidad de recursos de rutado, a fin de satisfacer las demandas de los sistemas que puedan ser implementados en ellas.

El esquema de rutado más común en las FPGAs comerciales es la arquitectura de islas, (figura 1.2). Este nombre se debe a que la disposición espacial de las celdas lógicas se asemeja de algún modo a islas sobre un mar de recursos de rutado. Las celdas lógicas se disponen en una rejilla bidimensional, y se conectan entre sí a través de los CB (Connection Block) y los SB (Switching Block), teniendo los IOBs (Input/Output Blocks) en la perifieria de la red de rutado.

La red de rutado de una FPGA se compone de cableado y de matrices de rutado (SB y CB). El componente fundamental de dichas matrices es el interruptor programable, que se basa en un transistor MOS programado desde la puerta que permite o impide el paso de corriente. Un CB no es más que una rejilla de cables horizontales y verticales con un único transistor programable en cada intersección, que en caso de estar programado desde la puerta, realiza una conexión efectiva entre el cable horizontal y el vertical (figura 1.3). Nótese también que la mayoría de los bits de configuración de estos elementos está a 0, aún en el caso de que el

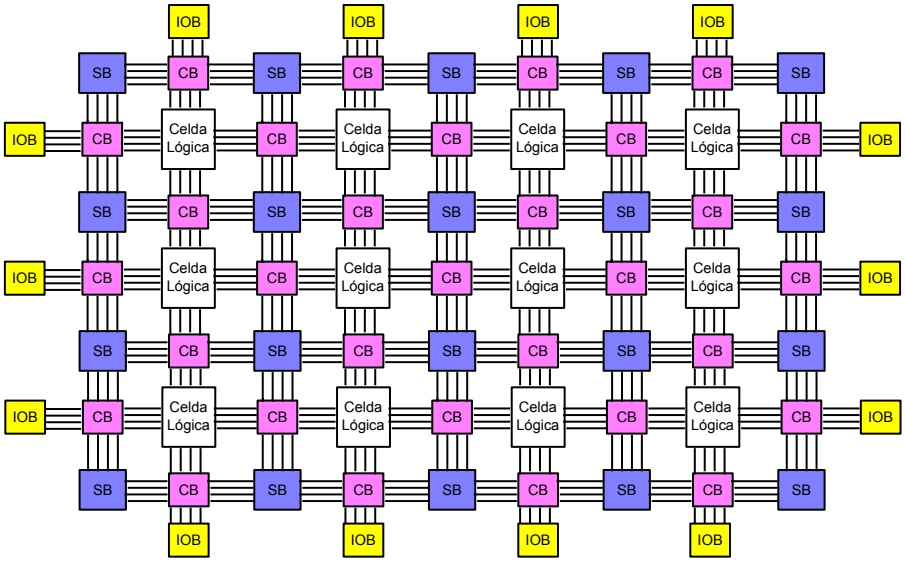


Figura 1.2: Arquitectura de rutado de islas

CB esté siendo utilizado por el diseño.

Los SB permiten la conexión entre dos cables cualesquiera de los que llegan a él, también se implementan como una rejilla, pero en este caso cada punto de interconexión se compone de 6 transistores, de forma que dos señales puedan cruzar por un punto concreto sin que se produzcan posibilidades de cortocircuito (figura 1.4). Los puntos de interconexión entre cables de rutado horizontales y verticales se denominan PIPs (Puntos de Interconexión Programables), y constan de un transistor para los CB y de 3 ó 6 para los SB.

El SB más común se implemente mediante puntos de interconexión de 6 transistores, pero también se han desarrollado esquemas de 3 transistores. En este caso, el PIP se considera direccional en lugar de bidireccional. En [17] se analizan las características de los PIPs direccionales y los bidireccionales, concluyendo que los PIPs direccionales consiguen mejores resultados en área y en frecuencia a cambio de una reducción en la flexibilidad de la arquitectura de rutado.

Debido a que las propiedades eléctricas de los transistores son inferiores a las de las conexiones metálicas, las señales que cruzan multitud de PIPs presentan mayor degradación de sus características frecuenciales. Por tanto, aquellas señales que comuniquen bloques lejanos dentro de la FPGA han de evitar cruzar por CBs

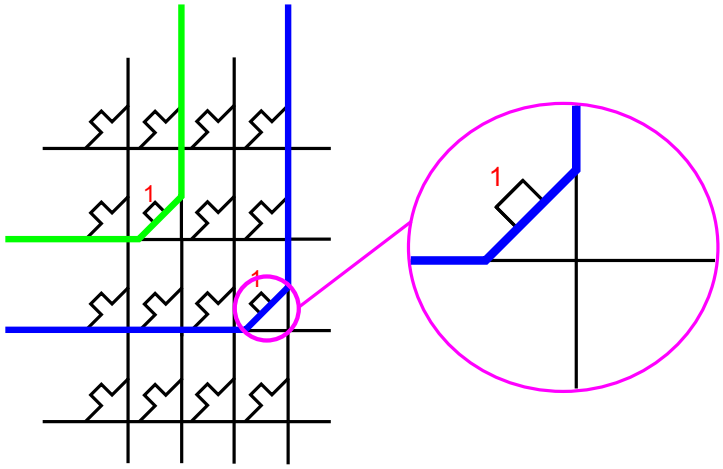


Figura 1.3: Connection Block (CB)

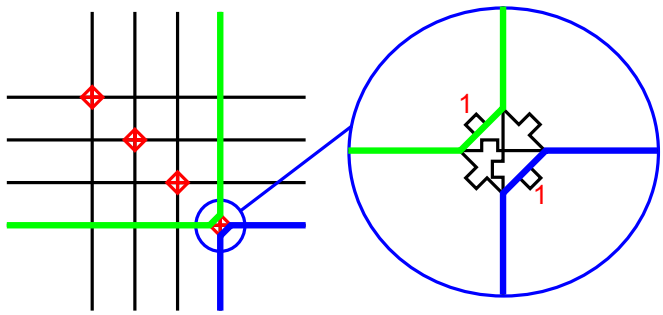


Figura 1.4: Switching block (SB)

y SBs en la medida de lo posible. Por este motivo, las FPGAs modernas contienen jerarquías en la arquitectura de rutado [18]. Hay líneas de alta prioridad que no cruzan por todos y cada uno de los SB, sino que recorren grandes distancias sin encontrarse con ningún PIP. Por otro lado, se prioriza el uso las líneas de menor jerarquía para conexiones de mayor localidad.

1.1.3 Memoria de configuración

La información que determina si un transistor concreto se encuentra en corte o en conducción se almacena en una memoria denominada memoria de configuración. La información almacenada en dicha memoria se lleva a las puertas de los transistores de rutado para que estos permitan o impidan el paso de corriente. Aunque la memoria de configuración también almacena la configuración interna de las celdas lógicas, el 90 % de los bits de ésta se corresponden con recursos de rutado [18], siendo la mayoría de éstos ceros (transistores en corte).

Teniendo esto en cuenta, una FPGA puede verse en dos capas: la memoria de configuración y el circuito implementado(figura 1.5). En la memoria de configuración se carga un fichero llamado bitstream, un fichero binario (compuesto por unos y ceros) que contiene la información del circuito a implementar en el dispositivo. La capa de circuito se puede ver como un conjunto de recursos lógicos (puertas lógicas, flip-flops, pines, buffers de reloj, etc) desconenctados entre sí. Cada bit almacenado en la memoria de configuración tiene su reflejo a nivel de circuito implementado, pudiendo especificar la configuración de un circuito o una conexión de rutado concreta. Cuando el bitstream es cargado en la memoria de configuración los recursos lógicos del nivel de circuito se conectan entre sí, dando sentido al circuito digital implementado en el dispositivo.

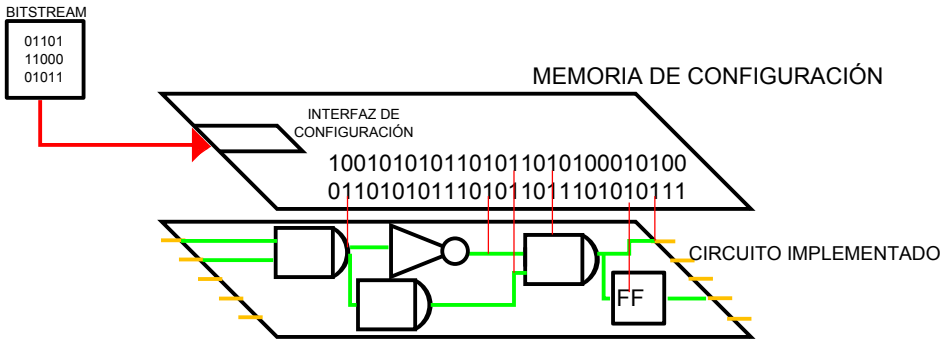


Figura 1.5: FPGA en capas

El contenido de la memoria de configuración es altamente sensible. En caso de almacenar información errónea, el reflejo sobre la capa de circuito implementado es un circuito con algún componente erróneo. Un error de software en la memoria de configuración desencadena un error de hardware a nivel de circuito. Este fenómeno se conoce como SEU (Single Event Upset), y requiere de la reconfiguración del dispositivo para su subsanación.

Sin embargo, los diseños implementados en FPGAs nunca hacen uso del 100 % de los recursos disponibles, por lo que algunos bits de la memoria de configuración hacen referencia a recursos no utilizados. De hecho, si un diseño concreto emplea el 100 % de las celdas lógicas (altamente improbable), va a seguir existiendo gran cantidad de recursos no utilizados. Tal y como se ha comentado anteriormente, la mayoría del contenido de la memoria de configuración de una FPGA se corresponde con bits de rutado, de los que muchos hacen referencia a PIPs que no están siendo utilizados. Cuando un SEU afecta a un bit de la memoria de configuración no utilizado no se produce ningún tipo de fallo en el funcionamiento del sistema. Por lo tanto, es conveniente una estimación del número de bits críticos para el funcionamiento del sistema, algo que es el objetivo principal de esta tesis.

La mayoría de las FPGAs contienen una memoria de configuración de tipo SRAM, conocida como CRAM (Configuration RAM). También hay FPGAs con memorias de configuración de otras tecnologías (principalmente flash y antifuse), cuyo uso se reduce a nichos de mercado. Estos dispositivos especiales son analizados en más profundidad en la sección 3.3.

1.2 Evolución histórica de las FPGAs

En esta sección se hace una retrospectiva de la evolución de las FPGAs desde su nacimiento a mediados de los 80 hasta la actualidad. Aquí se distinguen tres etapas fundamentales. En primer lugar, la época del nacimiento, que transcurre entre mediados de los 80 y principios de los 90, donde unos dispositivos de capacidades bastante reducidas se abrían paso a duras penas en un mercado muy competitivo. Una segunda época de expansión, que data entre mediados de los 90 y principios de los 2000, en la que las FPGAs pasan a ser válidas para un rango de aplicaciones mucho mayor debido a grandes avances en tecnologías de fabricación. Y una época de consolidación, desde mediados de los 2000 hasta la actualidad, en la que se van añadiendo diferentes funcionalidades para ir adaptándose a las exigencias del mercado.

Desde que en 1984 surgiera la primera FPGA, las prestaciones de éstas se han visto incrementadas de forma exponencial. En concordancia con la ley de Moore,

el número de transistores que las componen se ha visto duplicado cada dos años. De acuerdo con [19], la capacidad de las FPGAs se ha visto incrementada por 10000 en el intervalo 1984-2010. De igual forma, la frecuencia se ha multiplicado por 100, y el coste en energía por operación se ha reducido en un factor de 1000. En [19] se presenta la evolución que han tenido estos dispositivos desde su nacimiento hasta la actualidad, y se toma como documento de referencia para la elaboración de esta sección.

1.2.1 Etapa de nacimiento

A mediados de los 80, decenas de compañías se dedicaban a la venta de ASICs (Application-Specific Integrated Circuit). En esa competencia, los atributos más demandados por los compradores eran bajo coste, más capacidad y mayor velocidad. En el momento en que aparecieron las primeras FPGAs, las prestaciones ofrecidas en esos atributos eran claramente inferiores a los ASICs. Las FPGAs solamente podían dar cabida a una fracción pequeña de todos los sistemas electrónicos, ya que a nada que el sistema fuera medianamente grande y complejo, éste no iba a caber en el dispositivo.

La primera FPGA, fue la Xilinx XC2064 [20] en 1984, que contenía únicamente 64 celdas lógicas con 3 LUTs y un registro en cada una, equivalente a menos de 1000 puertas lógicas. A pesar de sus reducidas prestaciones, el dado de silicio era más grande aún que los microprocesadores de la época. El proceso de fabricación era de 2.5 micras, y su coste unitario superaba los centenares de dólares. Sin embargo, para el reducido rango de aplicaciones en el que eran viables, estos elementos novedosos resultaban muy atractivos. Tal y como se ha comentado anteriormente, un diseño basado en FPGA no requería de la gran inversión inicial del desarrollo de un ASIC. Además, el proceso de diseño era menos restrictivo, traduciéndose en un menor coste de desarrollo y un tiempo de salida al mercado más corto. Las FPGAs empezaban a ser atractivas incluso para diseños que no cabían en un único dispositivo, llegando a plantearse diseños con múltiples FPGAs [21, 22].

Las primeras FPGAs eran tán pequeñas que se consideraba aceptable la implementación manual de las funciones lógicas y las conexiones de rutado. Las tecnologías empleadas eran de carácter OTP (One-Time Programmable), como la de fusibles o la antifuse. La escasa complejidad de los diseños implementados reducía las probabilidades de fallos en el diseño, por lo que la reprogramabilidad de los dispositivos tampoco era una característica tan relevante. A finales de la década ya comenzaban a aflorar propuestas de FPGAs de tecnología SRAM para permitir que los dispositivos puedan ser reprogramados varias veces durante el diseño, solventando las problemáticas y limitaciones de los dispositivos OTP [23].

1.2.2 Etapa de expansión

La década de los 90 supuso una etapa de expansión del mercado de las FPGAs. En este intervalo las prestaciones de los dispositivos crecieron de manera exponencial, haciéndolos adecuados para rivalizar con los ASICs en sectores que anteriormente no era posible. Muchos clientes de ASICs se convirtieron en clientes de FPGAs, lo que permitió la consolidación de la tecnología.

Durante esta década, el número de transistores se duplicaba cada dos años en consonancia con la ley de Moore, lo que supuso un gran desafío para los desarrolladores, que debían traducir los innumerables avances tecnológicos en un aumento de prestaciones de las FPGAs. La capacidad de los dispositivos era adecuada para muchas aplicaciones, propiciando que aspectos como el rendimiento en frecuencia o consumo de potencia comenzasen a considerarse relevantes. Esto no quiere decir que el área y la cantidad de recursos no se considerasen importantes, sino que otras características de los dispositivos comenzaban a ser exigidas por los clientes para poder hacer frente a los ASICs, que también estaban mejorando de forma vertiginosa.

En estos años las FPGAs de tecnología SRAM se consolidan, y debido a la ley de Moore, se colocan al frente de la tecnología de circuitos electrónicos programables. En el momento en el que se produce cualquier evolución en las tecnologías de fabricación, los primeros elementos en ser adaptados son los elementos de interconexión y los transistores, que son precisamente los elementos que componen una FPGA de tecnología SRAM. Para cuando los avances de fabricación eran aplicados al antifuse o a las celdas flash, las SRAM ya iban por la siguiente generación. Además, las SRAM ofrecían la posibilidad de configurarse una y otra vez, facilitando enormemente el proceso de diseño. Aunque una vez optimizadas las FPGAs de tecnología antifuse pueden tener mayor densidad de integración que las SRAM, la ley de Moore provocó que las FPGAs antifuse fueran trasladadas a productos de nicho, quedándose la mayor parte del mercado para los fabricantes de FPGAs SRAM [19].

Otro de los hitos que propició la evolución de las FPGAs fue el proceso conocido como CMP (Chemical-Mechanical Polishing), que permitió a los fabricantes de circuitos integrados disponer de más capas metálicas. Esto fue aprovechado por los fabricantes de FPGAs para incrementar las capacidades de interconexión entre celdas, adaptándose así a la mayor cantidad de celdas que tenían los dispositivos debido al aumento del número de transistores [24].

El continuo incremento de las prestaciones de los dispositivos comenzó a exigir la automatización completa del proceso de diseño. Las aplicaciones que podían implementarse en una FPGA empezaban a ser tan grandes y complejos que la im-

plementación manual resultaba impracticable, por lo que empezaban a proponerse algoritmos de emplazamiento y rutado automático [25]. En este punto, algunos fabricantes de FPGAs fueron conscientes de la importancia de un software de desarrollo que posibilitara la síntesis, el emplazamiento y el rutado automático. Un software sencillo, intuitivo y fácil de aprender por parte de los desarrolladores podía suponer que un cliente de FPGA se decantara por una familia u otra. Aquellas compañías que apostaron por controlar el desarrollo de su propio software, y que hicieron de éste un elemento central de su modelo de negocio terminaron por adueñarse del mercado.

1.2.3 Etapa de consolidación

A partir de la década de los 2000 las FPGAs ya empezaban a ser muy grandes. Mucho más grandes que las aplicaciones que en ellas se implementaban. Desde este instante, los clientes dejan de tener la necesidad de adquirir el dispositivo premium de la época. Por lo tanto, el incremento de capacidad deja de estar ligado al aumento de cuota de mercado. En este punto, los fabricantes comienzan a derivar el mercado de las FPGAs hacia productos low-cost. A nivel de software, se empieza a considerar la relevancia de proveer a los clientes de librerías de soft-logic para facilitar el diseño.

Las FPGAs de mayor tamaño se utilizan para la implementación de diseños muy grandes, que son en muchos casos sistemas completos. Por ello, los desarrolladores de FPGAs no trabajan únicamente en implementar lógica, sino también en hacer que el sistema cumpla con unos estándares y protocolos de comunicación determinados. Las comunicaciones se volvieron un elemento clave en los sistemas implementados en FPGAs, ya sea con elementos externos como entre bloques internos del propio sistema.

En este punto, los fabricantes de FPGAs toman un cambio radical en su estrategia. Ya no se trata de únicamente de hacer crecer el número de celdas lógicas para que puedan ser implementados sistemas cada vez más grandes. Estas celdas se complementan con diversos elementos "hard" tales como multiplicadores, bloques de memoria, PLLs, transceivers o procesadores PowerPC [26]. De esta forma, se facilita el desarrollo del sistema, se puede reducir el consumo de potencia, pueden aumentar las prestaciones de frecuencia y se facilita la adhesión a diferentes estándares.

A los desarrolladores también se les proporcionan librerías soft compatibles con todo tipo de estándares, tanto buses de comunicación interna [27, 28] (AXI, PLB, wishbone, etc) como externa (Ethernet, CAN, USB, SPI, etc). Estas librerías no se utilizan únicamente para la implementación de lógica en LUTs y flip-flops,

sino que han de ser capaces de integrar los nuevos elementos hard añadidos. El desarrollo de "IP Cores" deja de ser competencia exclusiva de los fabricantes, ya que empiezan a surgir empresas dedicadas exclusivamente a este cometido.

En las anteriores etapas, el crecimiento del mercado de las FPGAs se debía al reemplazo de ASICs. Sin embargo, a partir de esta década, las infraestructuras de comunicaciones son uno de los principales objetivo de este tipo de dispositivos. Se incorporan I/O transceivers de alta velocidad, así como líneas de rutado prioritarias para flujo de datos. Las FPGAs se convierten en routers, que encaminan paquetes desde unos interfaces a otros sin sacrificar throughput. Todo esto, sin perder la capacidad de implementar complejas unidades de procesamiento a nivel de bit en la lógica programable.

Estas tendencias se consolidaron durante las décadas 2000 y 2010. Las FPGAs fueron adquiriendo mayor cota de mercado, mientras que los ASICs mantuvieron los mercados con grandes volúmenes de fabricación, como por ejemplo, los dispositivos móviles. Por otro lado, los sistemas multiprocesador experimentaron un avance importante, pudiendo amenazar incluso mercados tradicionales de las FPGAs. Estos chips quad-core u octa-core pueden incorporar varias unidades de procesamiento gráfico (GPUs), mientras ejecutan software a frecuencias mucho mayores que las FPGAs. La implementación de sistemas operativos completos facilita la integración de una serie de servicios muy demandados en la actualidad, como por ejemplo, conectividad web.

En lugar de tomar los sistemas multiprocesador como una amenaza, los fabricantes de FPGAs lo vieron como una oportunidad. Debido a esto, la década del 2010 propició la aparición de SoCs complejos, que combinan FPGA con un sistema multiprocesador en el mismo chip. Se trata de un sistema procesador completo, que puede llegar a ocupar más de la mitad del total del silicio. La familia Xilinx Zynq 7000 salió al mercado el año 2012, conteniendo un ARM dual-core Cortex A9 a 200 MHz [29]. Este mismo fabricante sacó al mercado la familia Zynq UltraScale MPSoC el año 2015, con dos sistemas procesadores diferentes.Un ARM quad-core Cortex-A53 para ejecutar software a alta velocidad, y un ARM Cortex-R5 dual core para aplicaciones en tiempo real. Algunos productos de esta familia también incorporan unidades de procesamiento gráfico (GPUs). Altera, el otro gran fabricante de FPGAs de tipo SRAM ,tampoco se queda atrás en este tipo de productos. Esta empresa, que ahora forma parte del grupo Intel, ofrece diversas familias de SoCs que combinan sistema procesador y FPGA tales como la Stratix10 SoC, Arria10 SoC, ArriaV SoC y CycloneV SoC.

1.3 Flujo de diseño en FPGAs

El objetivo de esta sección es describir las diferentes fases por las que transcurre cualquier diseño implementado en una FPGA. Los conceptos relativos a esta sección van a ser referenciados en varias ocasiones a lo largo de la tesis, por lo que es preciso haberlos introducido. El primer paso es la generación de fuentes, que consiste en elaborar los ficheros necesarios que van a determinar la arquitectura hardware. Las etapas de síntesis, implementación y generación de bitstream se van a encargar de preparar el diseño descrito anteriormente para ser cargado en la FPGA. Las etapas de simulación y verificación permiten validar el funcionamiento del circuito descrito y proceder a modificaciones en caso de necesidad. Todas estas etapas de diseño pueden observarse en la figura 1.6.

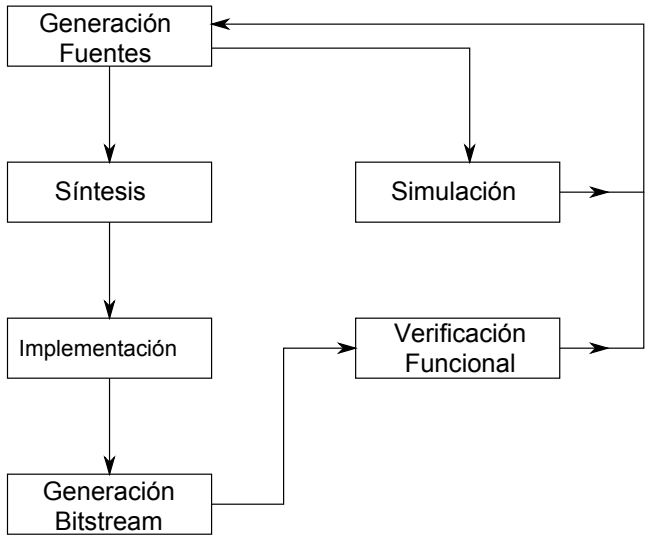


Figura 1.6: Flujo de diseño en FPGAs

1.3.1 Generación de fuentes

La primera fase del flujo de diseño es la generación de fuentes. Las fuentes son los ficheros que especifican las funciones lógicas que son implementadas en el dispositivo. Tradicionalmente, esto se ha venido haciendo mediante lenguajes de descripción de hardware, también conocidos como HDLs (Hardware Description Languages), que son lenguajes específicos para la descripción de circuitos

electrónicos. Aunque algunas de sus construcciones sintácticas puedan asemejarse a programas de software, la realidad no es esa. En estos ficheros se lleva a cabo una descripción funcional de los diferentes módulos de hardware, que realizan operaciones de forma concurrente.

Por otro lado, los diferentes paquetes de software de desarrollo para FPGAs actuales incorporan múltiples posibilidades para la generación de los archivos de diseño. El más habitual es el entorno de bloques, en el que se combinan diseños propios con otros de librerías del fabricante o adquiridos a externos. Algunos componentes muy concretos se generan mediante menús desplegables que forman parte del propio software de CAD [30].

En los últimos años los fabricantes de FPGAs están dando la posibilidad de generar hardware desde otro tipo de entornos. Un ejemplo de esto es el Xilinx System Generator [31], que busca facilitar el diseño en FPGAs en entornos del tipo Matlab/Simulink. También hay sistemas en los que se describen circuitos a un nivel de abstracción más elevado, como HLS (High Level Synthesis) [32] o system C, en los que se pueden generar módulos hardware a partir de lenguajes de software como el lenguaje C.

1.3.2 Síntesis

El siguiente paso es la síntesis, que consiste en la generación de un circuito equivalente al descrito en las fuentes, que únicamente contenga los elementos que se pueden encontrar en el interior de una FPGA de la familia especificada (LUTs, multiplexores, DSPs, IOBs, etc)[33]. Esta descripción equivalente, denominada netlist, es dependiente de la familia de FPGAs, ya que el tipo de celda lógica cambia de una familia a otra, y la síntesis que aprovecha al máximo los recursos puede variar. Aunque el proceso de síntesis se lleva a cabo de forma automática, los desarrolladores pueden introducir restricciones a la herramienta de síntesis. Unos ejemplos de restricciones son los siguientes:

- Forzar los elementos en los que se implementan algunos módulos concretos (CLBs, DSPs, BRAMs).
- Combinar varias funciones lógicas en una LUT para ahorrar area.
- Especificar a la herramienta que se abstenga de realizar operaciones de optimización sobre algunas partes.
- Indicar cuál es el máximo fanout permitido para las señales más críticas.

1.3.3 Implementación

Posteriormente a la síntesis viene la etapa de implementación, que consiste en ajustar el netlist a una FPGA concreta [34]. Consta de dos fases principales: emplazamiento y rutado (más conocidas por sus denominaciones en inglés: place and route (PAR)). El emplazamiento se basa en adjudicar a cada elemento del netlist un recurso concreto del dispositivo. Por otro lado, el rutado es el responsable de gestionar los recursos (connection blocks y switching blocks) para establecer las conexiones entre las celdas y otros elementos que conforman el circuito lógico. Mientras que el proceso de síntesis es mayoritariamente automático, los desarrolladores tienen la posibilidad de añadir múltiples indicaciones para influir sobre la herramienta de PAR. Los mecanismos para realizar estas indicaciones son las restricciones o constraints.

Las órdenes de restricción más habituales son aquellas relacionadas con la frecuencia y la localización. La restricción de frecuencia limita el tiempo máximo de los caminos combinacionales entre registros, algo que ha de ser tenido muy en cuenta por la herramienta de PAR para restringir la cantidad de PIPs por los que cruza una línea concreta. La restricción de posicionamiento más habitual es la relativa a los pines hacia los que han de ser encaminadas las señales, ya que ésto viene fijado por el PCB.

Otra de las restricciones de emplazamiento más habitual es el pblock. Esto consiste en señalar un área restringida, generalmente rectangular, en la que un módulo del sistema se implementa aislado del resto. Una restricción de pblock indica que un módulo concreto ha de implementar sus funciones lógicas en las celdas lógicas y otros elementos pertenecientes al rectángulo indicado, y que, a su vez, el resto de módulos no podrán utilizar dichas celdas. De esta forma, el módulo implementado en el pblock queda aislado del resto del circuito. La restricción de pblock tiene impacto a nivel de celda lógica, pero no a nivel de rutado, ya que las señales pueden cruzar el pblock utilizando los recursos de rutado que se encuentran en el interior de este. La principal utilidad del pblock es la RPD (Reconfiguración Parcial Dinámica). En la figura 1.7 se puede observar la implementación de un módulo concreto en un pblock.

Además de la posibilidad de introducir restricciones, los desarrolladores pueden adaptar la herramienta de PAR para que el diseño esté optimizado de acuerdo a los criterios especificados. Aunque no suela ser habitual introducir modificaciones en estos parámetros a no ser de que existan problemas muy concretos, las optimizaciones más comunes son las de área, congestión y performance.

- **Optimización de área:** Agrupar al máximo los diferentes módulos para poder así ahorrar espacio y poder añadir más módulos al diseño.

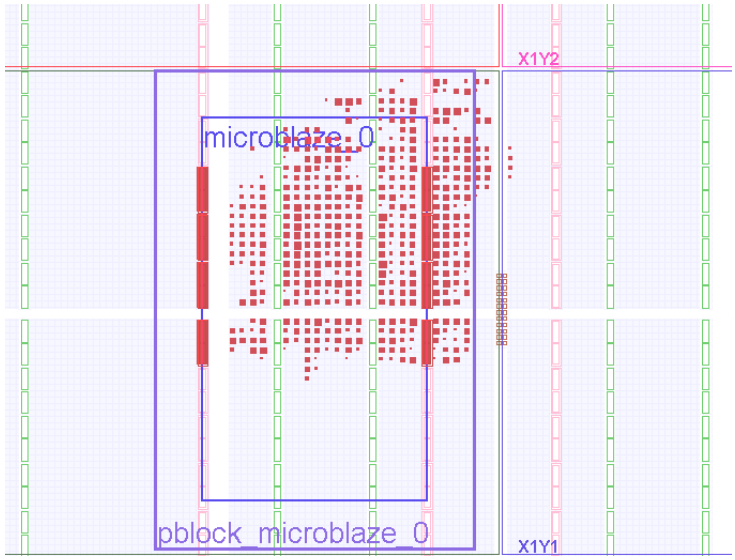


Figura 1.7: Módulo implementado en pblock

- **Optimización de congestión:** Reducir la cantidad de señales que han de ser rutadas a través de un área concreta, y se lleva a cabo dispersando los recursos a través de toda la FPGA.
- **Optimización de performance:** Maximizar la frecuencia del circuito. Esto se consigue con una congestión baja, para que haya muchos recursos de rutado disponibles y no sea necesario atravesar muchos PIPs. Sin embargo, en caso de que la lógica esté muy dispersa, el rendimiento se puede ver reducido por el retardo introducido por la longitud de las líneas.

1.3.4 Otras etapas

La etapa final es la de generación del bitstream, que consiste en generar un fichero binario para ser cargado en la memoria de configuración del dispositivo a partir de los archivos generados en la implementación. Este es un proceso prácticamente unívoco, en el que el desarrollador apenas puede influir.

Tal y como se aprecia en la figura 1.6 el proceso de desarrollo de aplicaciones para FPGAs consta de otras dos etapas, simulación y verificación funcional. El objeto de estas etapas es la detección de errores en el funcionamiento del sistema, para

poder introducir las pertinentes modificaciones en las fuentes. La simulación se lleva a cabo en el propio PC mediante software específico y ficheros de generación de estímulos. La verificación funcional consiste en chequear el funcionamiento con el bitstream cargado en la propia FPGA, para poder así analizar las interacciones con el resto de elementos del PCB.

1.4 FPGAs de Xilinx de la serie 7

En esta sección se va a analizar la estructura particular de las FPGAs de la serie 7 de Xilinx, debido a que la investigación se ha llevado a cabo en un dispositivo de esta familia, y las menciones a la estructura van a ser recurrentes. El punto de partida es el slice, que es el elemento que se replica de manera regular a lo largo del dispositivo. Lamentablemente no se dispone de información acerca de la estructura de rutado, ya que Xilinx no la comparte. Aún así se presupone similar al esquema de islas anteriormente mencionado. Posteriormente se analizan otros elementos contenidos en este tipo de dispositivos, tales como bloques de DSP, bloques de memoria y red de reloj.

1.4.1 CLBs y slices

El CLB (Configurable Logic Block) es el elemento lógico de mayor jerarquía, y es clonado de manera regular a lo largo del dispositivo en las FPGAs de Xilinx. Cada CLB tiene dos slices, que no tienen conexión directa entre sí, sino que pertenecen a cadenas de llevada de diferentes columnas. En los dispositivos de Xilinx, el slice es el elemento que anteriormente se ha definido como celda lógica. Cada slice contiene 4 LUTs de 32 bits y 8 flip-flops D. En los dispositivos de esta familia existen dos tipos de slice: las sliceM y las sliceL. La diferencia entre estos dos tipos de slice radica en que el primero puede configurar su LUT como memoria RAM distribuida o como registro de desplazamiento, mientras que el segundo carece de dicha posibilidad. En la figura 1.8 se presenta el esquema de un slice tipo de la serie 7. Como puede observarse, presenta una arquitectura similar a la presentada en la sección 1.1. La disposición de las slices en un CLB se ve representada en la figura 1.9. Aquí se aprecia que las dos slices de un CLB no tienen conexión directa entre ellas, forman parte de carry-chains paralelas.

Es importante reseñar que la utilización de LUTs como registros de desplazamiento es una implementación altamente eficiente, ya que puede hacerse en un único slice. Por el contrario, si el registro de desplazamiento se implementa en los flip-flops, se requiere de múltiples CLBs. El inconveniente radica en que a

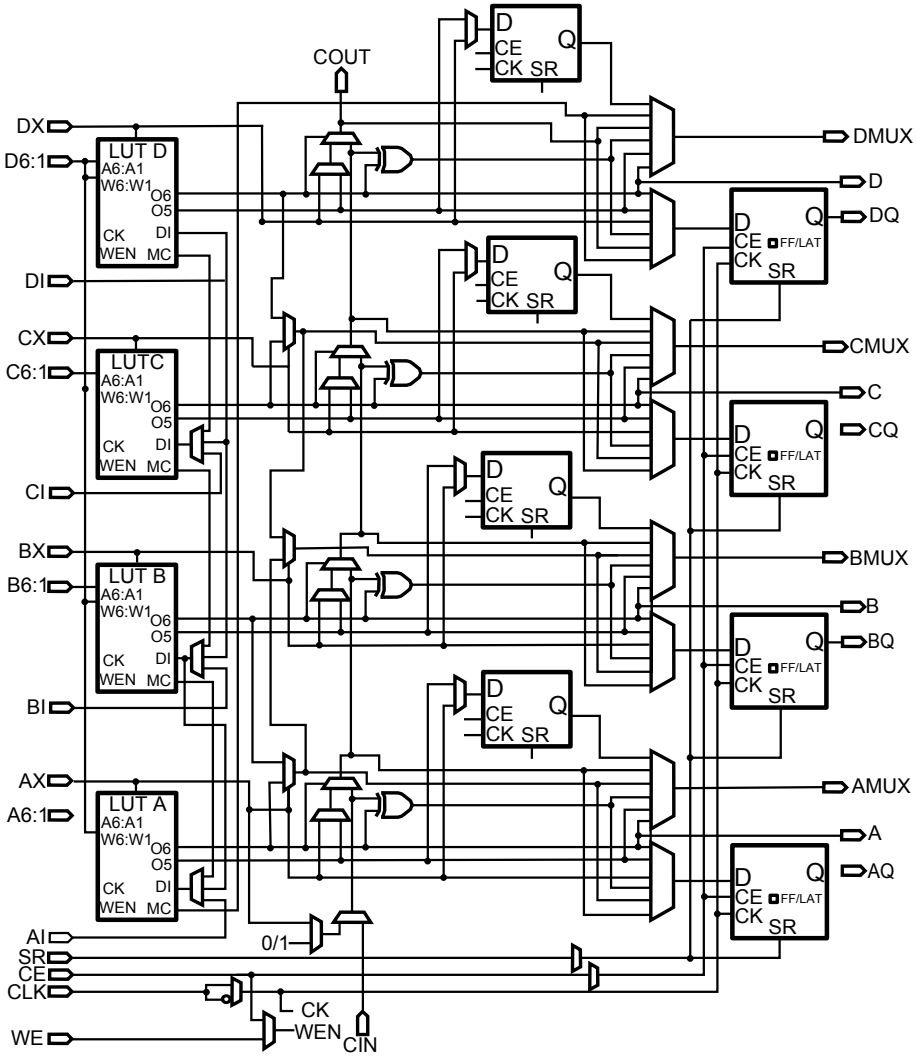


Figura 1.8: SLICEM de las FPGAs de la serie 7

esta configuración no se le pueden asignar entradas de set ni reset, y la única inicialización posible es mediante el bitstream. Esta carencia de ausencia de reset sucede igualmente cuando las LUTs se utilizan como memoria RAM distribuida.

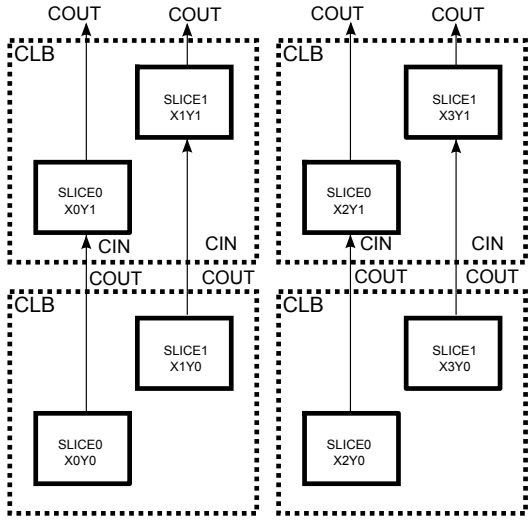


Figura 1.9: Distribución de slices en un CLB

1.4.2 DSP48

Las FPGAs son plataformas eficientes para la implementación de algoritmos de procesado de señal, ya que se pueden implementar de manera sencilla algoritmos totalmente paralelos y adaptados específicamente para la aplicación en cuestión. Estas aplicaciones necesitan realizar muchas multiplicaciones. Sin embargo, la implementación de multiplicadores en las celdas lógicas requiere de muchos recursos, y es ineficiente en términos de frecuencia y potencia. Para ello, las FPGAs de Xilinx de la serie 7 implementan bloques de DSP (Digital Signal Processing) en hard. En la figura 1.10 se adjunta un esquema del bloque DSP48E1, que se implementa en las FPGAs de la serie 7. Estos bloques están físicamente ubicados agrupados en columnas, tal y como puede observarse en la figura 1.11 [35].

En este esquema se aprecia que el elemento central es un multiplicador en complemento a 2 de 25*18 bits. Dicho elemento se complementa con una serie de registros y otros bloques operacionales, como un sumador, un detector de patrones y un comparador, siendo la configurabilidad de estos bloques escasa. Aún así,

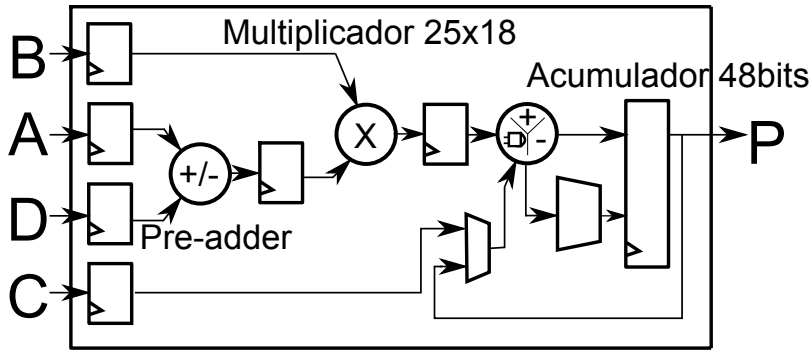


Figura 1.10: Estructura de una slice DSP48

en la memoria de configuración hay algunos bits para este cometido, como los que definen las operaciones de los bloques o el multiplexor para realimentar la salida. Tal y como se aprecia en la figura 1.11, existen columnas únicamente compuestas por bloques de DSP48 a fin de facilitar el rutado de señales entre este tipo de bloques y hacer más sencilla la implementación de filtros FIR.

1.4.3 Recursos de reloj

El reloj es un elemento esencial en cualquier sistema electrónico. Es una señal que ha de ser llevada a todos los flip-flops del sistema (además de las BRAM y otros recursos que emplean reloj), en caso de existir únicamente un solo dominio de reloj. En caso de ser un sistema con varios relojes, solamente ha de ser llevado uno a cada flip-flop. Los flancos de subida y de bajada han de darse de manera simultánea en cada uno de los flip-flops. Esto implica que las señales de reloj no pueden ser rutadas a través de la red de rutado convencional, sino que requieren de una red paralela para el rutado de señales de reloj. La red de rutado de reloj está organizada de manera jerárquica [36].

Todas las FPGAs de la serie 7 tienen una franja vertical para las señales de reloj, por la que se encaminan 32 líneas de reloj globales. Los dispositivos se dividen en regiones de reloj, que son franjas horizontales con una anchura de 50 CLBs. El número de regiones de reloj va desde 1 en el dispositivo más pequeño hasta 24 en el dispositivo más grande. Cada región de reloj es cruzada por una franja horizontal que contiene 12 líneas de reloj. Dicha franja divide la región en 2, dejando 25 CLBs por encima y otras 25 CLBs por abajo. Desde esta franja horizontal las señales de reloj se distribuyen a las diferentes CLBs. Este esquema

se puede ver en la figura 1.11.

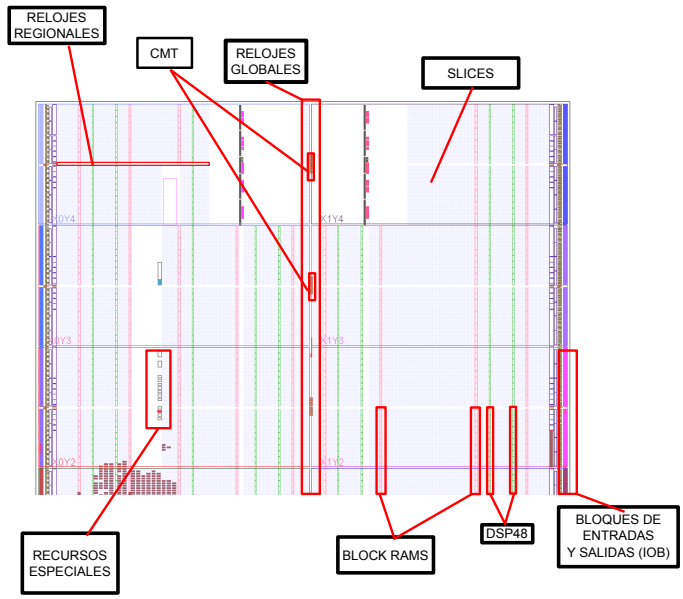


Figura 1.11: Estructura de FPGAs de la serie 7

Además de líneas de reloj, la red de reloj de una FPGA contiene algunos elementos programables, que son los buffers y los CMT (Clocking Management Tiles). Cada línea de reloj tiene como driver un buffer de reloj, que permiten regenerar la señal, adaptarla para cumplir con las características eléctricas de la línea por la que va a ser rutada y eliminar el skew y el jitter. Los buffers se pueden configurar como habilitadores/deshabilitadores de la señal de reloj, para así ajustar el consumo de potencia del sistema. Además, tienen la opción de funcionar como divisores de frecuencia. Estos elementos tienen diferentes denominaciones según su jerarquía. Los BUFG hacen referencia a los buffers de las señales globales de reloj, los BUFH a los buffers de la franja horizontal interna de una región de reloj, y los BUFR para buffers regionales.

Las FPGAs de la serie 7 tienen hasta 24 CMTs (Clock Management Tile), cada una de las cuales consiste de un MMCM (Mixed Mode Clock Manager) y un PLL (Phase-Locked Loop). Estos elementos son configurables desde el bitstream, y se utilizan para generar múltiples relojes con fases y frecuencias definidas, relacionadas con un reloj de entrada que se utiliza como referencia. Físicamente están localizados adyacentes a la franja vertical que distribuye los relojes globales.

1.4.4 Memorias

El almacenamiento de datos es un proceso de gran relevancia en los sistemas electrónicos. Para facilitar este punto, las FPGAs actuales incorporan bloques de memoria RAM. De esta forma, no es estrictamente necesario utilizar los CLBs como memoria distribuída, optimizando así el uso de recursos en la FPGA. Las FPGAs de la serie 7 concretamente implementan bloques de 36 kb de almacenamiento, que a su vez pueden funcionar como bloques independientes de 18 kb. Tal y como se puede observar en la figura 1.11, estos bloques están colocados en líneas verticales que atraviesan una región de reloj concreta, siendo el tamaño de las palabras de datos y la anchura de los buses configurable por el usuario [37].

Muchos diseños en FPGAs utilizan las BRAMs como FIFOs, siendo muy frecuente utilizar esta configuración para sincronizar diferentes dominios de relojes. Por este motivo, las BRAM de los dispositivos de la serie 7 tienen incorporada lógica para la implementación de FIFOs, tanto síncronas como asíncronas. De esta manera, no es necesario utilizar las CLBs para implementar los contadores, comparadores o generadores de flags. Esta lógica es aplicable para unos determinados tamaños de bus y de palabra de datos. Sin embargo, en caso de necesitar alguna configuración más específica se pueden utilizar los CLBs para la implementación de lógica dedicada.

Los bloques de RAM de los dispositivos de la serie 7 de Xilinx tienen incorporado el hardware necesario para corregir errores mediante código Hamming. Para ello, es necesario implementar estos bloques como RAM 512x72. De los 72 bits de cada palabra 64 se corresponden con información útil y los otros 8 para información redundante relativa al código Hamming. Este hardware se encarga de generar el código Hamming cada vez que se produce una operación de escritura, así como de verificarlo en las operaciones de lectura.

1.5 SoCs combinando FPGA y procesador

Una vez descrita la arquitectura de las FPGAs de la serie 7 se va a proceder al estudio de los SoCs que combinan FPGA y procesador. Se va a hacer especial hincapié en los dispostivos Zynq7000 ya que el desarrollo de esta tesis se lleva a cabo sobre un dispositivo de este tipo. Adicionalmente se van a introducir otras familias de dispositivos de esta naturaleza.

1.5.1 Zynq7000

La familia Zynq7000 integra en un único chip un sistema procesador completo basado en ARM y lógica programable (FPGA). La parte del procesador se conoce como PS (Processing System), y la parte de FPGA como PL (Programmable Logic). El elemento central del PS es un CPU ARM Cortex-A9, que también contiene memoria on-chip, interfaces para memoria externa y un amplio conjunto de periféricos de I/O. El PL, por su parte está compuesto por una FPGA de la serie 7, que puede ser Artix o Kintex. La familia Zynq7000 vio la luz en 2012, la tecnología de fabricación es de 28 nm y es una de las pioneras en lo referente a SoCs que combinan FPGA y procesador.

La familia Zynq7000 pretende combinar la flexibilidad y escalabilidad de una FPGA, con el rendimiento, potencia y facilidad de uso asociadas normalmente con procesadores. La familia Zynq7000 presenta una amplia gama de dispositivos, lo que permite a los desarrolladores enfocar los diseños hacia un bajo coste o hacia un mayor rendimiento. Si bien todos los dispositivos de esta familia contienen el mismo PS, los recursos de PL e I/O varían entre los dispositivos. En la guía de Xilinx [38] se presentan los dispositivos Zynq7000 AP SoC como adecuados para una amplia gama de aplicaciones. Éstas incluyen procesamiento de imagen, visión artificial, asistencia automotriz del conductor, radio LTE, control de motores o redes industriales entre otras.

Como la parte de PL coincide plenamente con la estructura de una FPGA de la serie 7, que ha sido detallada en profundidad en la sección 1.4. A continuación se detalla la arquitectura de la parte del PS, representada en la figura 1.12.

El elemento central del PS es un procesador ARM Cortex-A9 dual core (las gamas más bajas de Zynq puede ser single core). Cada core tiene su propio procesador, unidad de memoria y caché de primer nivel (L1) de 32 kB. Hay una caché de segundo nivel (L2) compartida por ambos cores de 512 kB. Adyacente a los procesadores se encuentra también la OCM (On-Chip Memory), que es una memoria con 256 kB de RAM y 128 kB de ROM. La ROM se utiliza exclusivamente en el proceso de arranque del procesador, y no es visible para el usuario (BootROM). La RAM del OCM es accesible desde el bus AXI, y en ella se pueden almacenar tanto software como datos.

La memoria del sistema es ampliable mediante memoria DDR externa, para lo que el PS dispone de un controlador de DDR. El bus de direcciones permite direccionar hasta 1 GB de memoria, aunque no sean frecuentes ampliaciones de memoria tan grandes. También se dispone de un controlador de SD, lo que va a permitir una capacidad de almacenamiento mucho mayor. La tarjeta SD puede utilizarse como soporte para la implementación de sistemas operativos

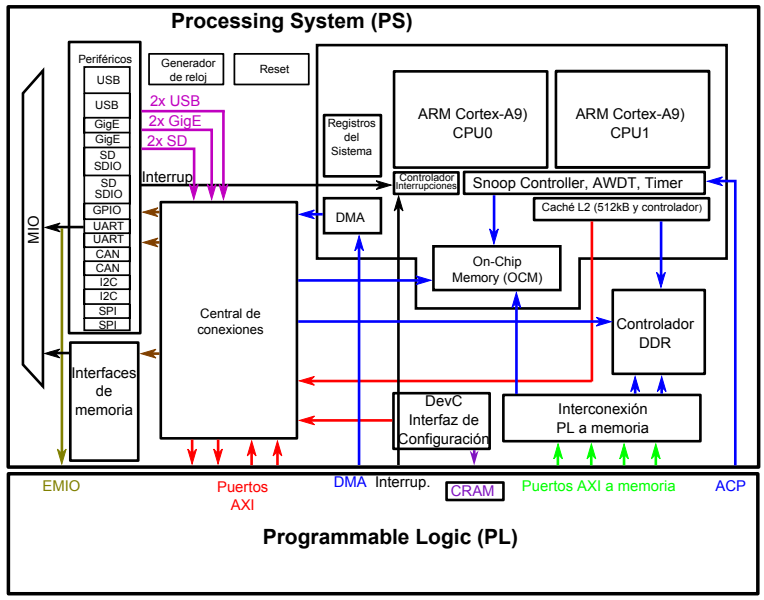


Figura 1.12: Arquitectura del PS del Zynq

completos, principalmente basados en Linux. Xilinx provee de imágenes de Linux adaptadas para Zynq, así como las directrices para hacer compatible el hardware implementado en la FPGA con el propio sistema operativo.

Además de esto, el PS también contiene controladores de comunicaciones USB, SPI (x2), CAN (x2), UART (x2), I2C (x2) y Ethernet (x2). Estos controladores llevan mapeada una dirección de memoria para poder ser accesibles desde el procesador. La salida de estos controladores puede llevarse hacia los pines del dispositivo o hacia la FPGA, como se representa en la figura 1.13. Para el primero de los casos se hace uso de la interfaz conocida como MIO (Multiplexed Input/Output), mientras que para el segundo caso se emplea el EMIO (Extended Multiplexed Imput/Output). El MIO y el EMIO pueden emplearse también para encaminar señales desde la FPGA hacia las interfaces de comunicación. Estas rutas se configuran mediante multiplexores, que son programados desde la memoria de configuración. Por todo esto, se puede concluir que algunos bits del bitstream tienen impacto directo sobre el PS.

Además del EMIO, la comunicación entre el PS y el PL puede darse a través del bus AXI. Entre ambas regiones del Zynq se pueden configurar hasta 4 interfaces AXI, que pueden ser maestros o esclavos. Esto permite la implementación de

periféricos "custom" en la FPGA direccionables desde la memoria del procesador. También existe una línea de interrupción entre ambas entidades, de forma que eventos ocurridos en el PL puedan interrumpir el software del PS. Además de esto, desde el PS se pueden llevar señales de reloj de frecuencia programable hacia el PL.

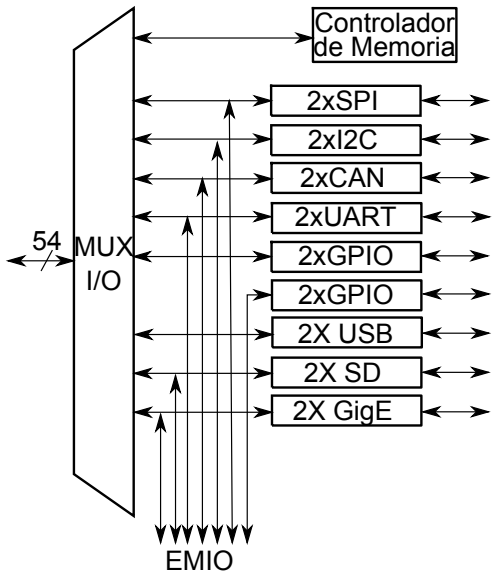


Figura 1.13: Estructura del interfaz MIO y EMIO

1.5.2 Otos SoCs que combinan FPGA y procesador

Además de la familia Zynq7000, en el mercado existen otros productos SoC que combinan FPGA con procesador. Los ejemplos más representativos son los siguientes:

- **Zynq UltraScale**

  La familia Zynq UltraScale MPSoC se corresponde con la siguiente generación de Zynq [39]. Con una tecnología de fabricación de 16 nm, se introducen una serie de características adicionales, principalmente en la zona del PS. El sistema procesador ARM dual core del Zynq7 se sustituye por un ARM Cortex-A53, que puede ser dual core o quad core dependiento de la gama. Este módulo puede operar a una frecuencia hasta 1.5 GHz y puede traba-

jar con datos de 32 y 64 bits. Además, se incorpora un RPU (Real-Time Processing Unit) basado en un módulo ARM Cortex-R5 dual core para la implementación de aplicaciones en tiempo real. Adicionalmente, el PS puede contener un GPU ARM Mali-400 (procesador gráfico de 4 núcleos) para algunas de sus gamas, además de una serie de avances a nivel de gestión de memoria y periféricos. Esta familia fue lanzada al mercado a finales de 2015.

A nivel de FPGA, el principal avance viene dado por el salto de 28 nm a 16 nm. De esta forma, la densidad de integración es mayor, pasando de 444 k celdas lógicas del dispositivo más grande de Zynq7000, a 1143 k celdas para el mayor dispositivo de la famila UltraScale. El lanzamiento de dos familias de SoCs en un periodo de tiempo tan corto deja claro que la línea de SoCs que combinan FPGA y procesador es de gran relevancia para la compañía.

- **Altera - Intel**

  Altera, el otro gran fabricante de FPGAs de tecnología SRAM, que fue adquirido por Intel en Junio de 2015, también incorpora en su portfolio una amplia variedad de SoCs que combinan FPGA con procesador [40]. La familia Stratix10 es el tope de gama de esta empresa. La tecnología de fabricación es de de 14 nm, e incluye un sistema procesador basado en ARM Cortex-A53 quad-core de 64 bits que puede funcionar hasta a 1.5 GHz. La familia Arria 10 SoC es la gama baja de la misma generación. Estos dispositivos están fabricados a 20 nm y contienen un ARM Cortex A9 dual-core de 32 bits a 1.5 GHz.

  La anterior generación de chips de este mismo fabricante (28 nm) también incluye SoCs combinando procesador y FPGA. Aquí el ARM es dual-core de 32 bits y la frecuencia máxima de trabajo es de 1.05 GHz para los dispositivos ArriaV y de 925 MHz para los CycloneV. Cabe destacar que existe un claro paralelismo entre Xilinx y Altera en su estrategia respecto a este tipo de chips. Ambos los introdujeron en el mercado de manera prácticamente simultánea y ambos han dado continuidad a esta línea en la siguiente generación. Además las prestaciones ofrecidas por dispositivos equivalentes de ambos fabricantes son similares.

- **Microsemi smartfusion y smartfusion2**

  Los dispositivos que combinan FPGA con procesador no se limitan a la tecnología SRAM. Las familias SmartFusion y SmartFusion2 de Microsemi [41], un fabricante de FPGAs de tecnología flash, también combinan sistemas microprocesador basados en ARM con FPGA. Estos productos se focalizan principalmente hacia sectores de nicho de mercado, como el aero-

espacial o el avíonico debido a las mejores características de tolerancia a la radiación (capítulo 3). La familia SmartFusion data de 2012, y al igual que en los otros casos, el fabricante ha decidido dar continuidad a este tipo de producto.

1.6 Configuración y bitstream de Xilinx

Las FPGAs de Xilinx se configuran mediante un fichero llamado bitstream [42]. Es un fichero relativamente ligero, que como mucho llega a los 4 MB. Este fichero es el que ha de cargarse en la memoria de configuración de la FPGA, y contiene los valores de configuración y los valores iniciales de los distintos elementos del dispositivo. Los valores de configuración son aquellos que definen el hardware implementado y se mantienen estables durante la ejecución. Los valores iniciales definen el estado inicial de los elementos secuenciales del circuito (memorias y flip-flops).

Gran parte de los bits del bitstream son para programar los recursos de rutado, es decir, indican el valor de puerta de los transistores de rutado que componen las diversas matrices de conmutación y matrices de conexiones a lo largo del dispositivo. Estos bits van a permanecer constantes durante la ejecución de la aplicación.

El bitstream también contiene los bits de configuración de los CLBs. Estos bits comprenden los valores iniciales de los flip-flops, los valores de los multiplexores de rutado internos y los valores de las LUTs. Estos últimos son valores constantes cuando las LUTs se emplean como funciones combinacionales estándar, y valores iniciales cuando las LUTs se utilizan como memoria RAM distribuída o como registros de desplazamiento. Cabe destacar que el bitstream es la única posibilidad para la inicialización de las LUTs, ya que, a diferencia de los flip-flops, no tienen entrada de reset o preset.

Otro de los grandes grupos de bits del bitstream es el referente a las BRAMs. En este caso la gran mayoría de los bits hacen referencia a valores iniciales del contenido de la memoria. Sin embargo, también hay bits que detallan la configuración de estos bloques. Éstos configuran la BRAM para emplearla como FIFO, para definir los anchos de bus de datos y de direcciones o para implementar la detección y corrección de errores por código hamming. Además de esto, en el bitstream también se almacenan los bits de configuración relativos al resto de componentes de la FPGA. Estos componentes son los bloques de entradas y salidas, los bloques de DSP, los recursos de reloj y el resto de elementos configurables. Además, en

TIPO DE CABECERA	CÓDIGO DE OPERACIÓN	DIRECCIÓN DEL REGISTRO	RESERVADO	CONTENIDO
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	XX	RRRRRRRRRXXXXX	RR	XXXXXXXXXXX

Figura 1.14: Estructura de los comandos del bitstream

casos de SoCs como el Zynq, el bitstream también puede contener información de configuración del sistema procesador.

1.6.1 Estructura del bitstream

La estructura de los ficheros de configuración para dispositivos de Xilinx se especifica en el documento [42]. El bitstream de Xilinx consta inicialmente de una secuencia de arranque estructurada en comandos de palabras de 32 bits. Comienza con unas cuantas palabras vacías "FFFFFFFF" (dummy word). Luego prosigue una secuencia de iniciación y selección de anchura de bus. Posteriormente se transmiten comandos que contienen la estructura indicada en 1.14.

Hay dos tipos de paquetes, los de tipo 1 y los de tipo 2. Para los paquetes de tipo 1 hay que especificar a qué registro se va a mandar la información y cuantas palabras se van a enviar a dicho registro. Entre los registros más importantes se destacan los siguientes:

- CRC: En este registro se almacena el valor del CRC para verificar la integridad del bitstream.
- FAR: (Frame Address Register), se utiliza para direccionar la FPGA, indica en qué posición se va a escribir o leer de la PL.
- FDRI: Este registro se utiliza para escribir en la FPGA, lo que aquí se escriba se carga en la PL en el punto indicado en el registro FAR.
- FDRO: Este registro se utiliza para leer de la FPGA, cuando se hace una lectura de este registro se está leyendo el contenido de la PL en el punto indicado en el registro FAR
- CMD, utilizdo para escribir comandos. Los más habituales son los siguientes.
  - RCRC: Se utiliza para resetear el registro CRC.

- START: Iniciar la FPGA, tiene efecto cuando llega el comando DESYNC.
- SHUTDOWN: Iniciar la detención de la FPGA, esta operación concluye al hacer RCRC.
- DESYNC: Finalizar la transmisión de datos por la interfaz de configuración.

Las palabras que van a continuación del comando se cargan en el registro especificado. El bitstream contiene bastantes comandos NOOP, "20000000". Este comando no hace nada, pero el bitstream lo incluye para facilitar la sincronización. Existen también paquetes de tipo 2. A diferencia de los de tipo 1, aquí no se especifica a qué registro se transfiere la información. Los paquetes de tipo 2 van a continuación de un paquete de tipo 1 y tienen más bits de direccionamiento para indicar el número de palabras que van a ser cargadas en un determinado registro.

La configuración total de la FPGA se hace mediante un bitstream total. Tras la secuencia de sincronización detallada anteriormente, se guarda un valor en el registro CRC. El registro FAR se inicializa a 0 y se transmite la orden de enviar al registro FDRI el número total de palabras del bitstream. Tras finalizar la transmisión se hace el chequeo del CRC. Si es correcto se envía la orden de START para iniciar la FPGA y, finalmente, se cierra la comunicación con la interfaz mediante el comando DESYNC.

Para hacer reconfiguración parcial dinámica se requieren bitstreams parciales. A diferencia de los totales, estos ficheros se cargan en una dirección FAR distinta a 0 perteneciente a la región reconfigurable. Además el número de palabras que se cargan en el registro FDRI es distinto del total. Si la región reconfigurable ocupa zonas no consecutivas según el direccionamiento FAR, tras la carga del primer fragmento hay que especificar una nueva dirección FAR y un nuevo tamaño hasta cargar todos los fragmentos del bitstream parcial. Los bitstreams parciales no contienen el comando START, ya que son cargados con la FPGA en marcha, lo que se conoce como "Reconfiguración Parcial Dinámica (DPR)".

El número de palabras que se carga en el registro FDRI ha de ser múltiplo del tamaño de frame. Una frame hace referencia a los bits de configuración y valores iniciales de una columna concreta. Cada tipo de recurso (Slice, BRAM, DSP, IOB, etc) tiene un número de frames distinto y cada familia de dispositivos de Xilinx tiene un tamaño de frame diferente. En la figura 1.15 puede observarse de forma gráfica la estructura del fichero.

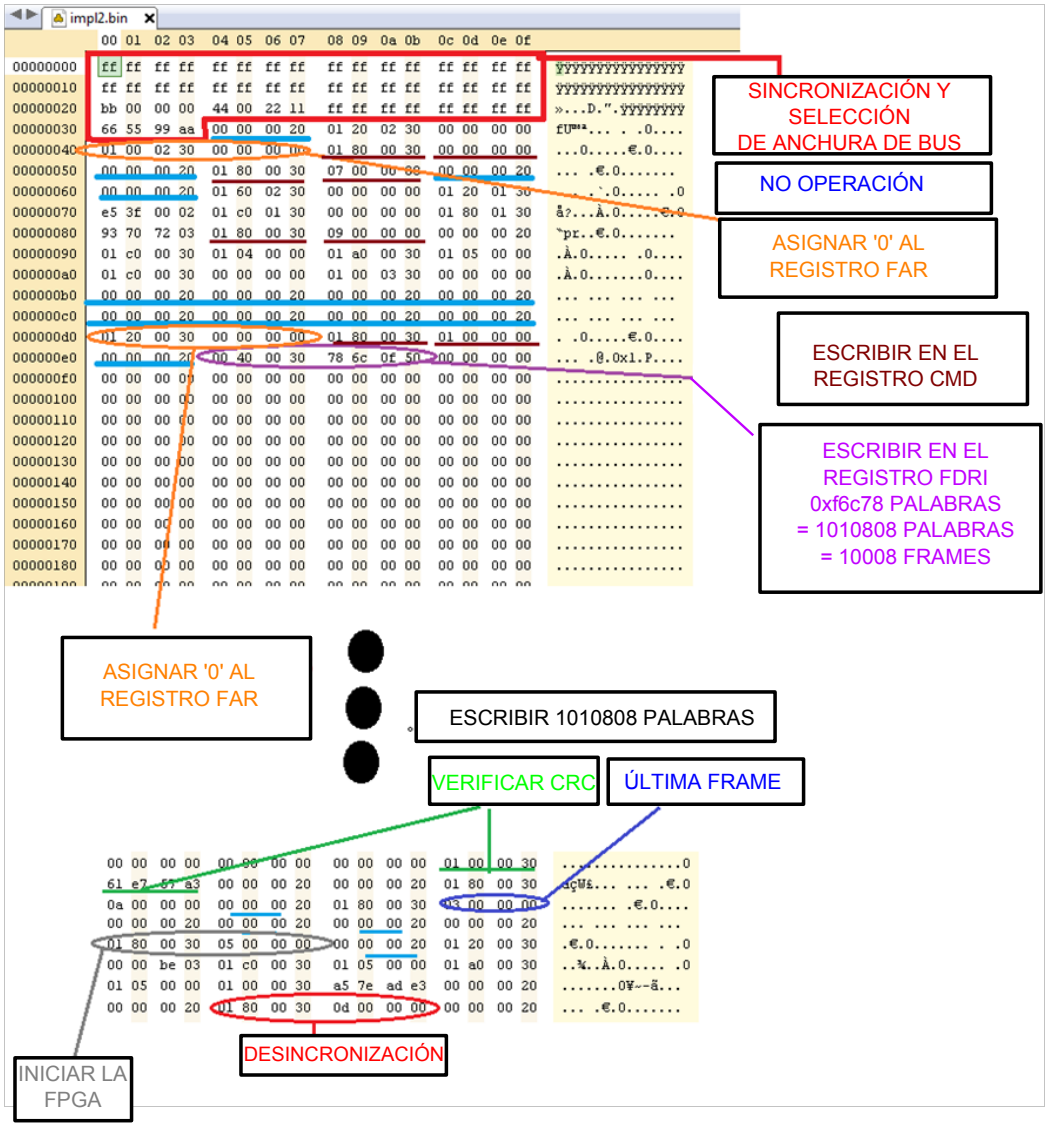


Figura 1.15: Estructura del fichero de bitstream

1.6.2 Interfaces de configuración

Las FPGAs modernas tienen diversos interfaces mediante los cuales se puede efectuar su programación. Para las FPGAs de Xilinx de la serie 7 estas posibilidades son JTAG, SelectMAP e ICAP. JTAG es un protocolo de comunicaciones serie bastante extendido para la configuración de dispositivos electrónicos. En el ámbito de las FPGAs, se emplea este protocolo para conectar con un PC externo, ya que los softwares de desarrollo de aplicaciones para FPGAs suelen implementar este protocolo. Es un interfaz relativamente lento, ya que puede necesitar decenas de segundos para cargar un bitstream completo. Cabe reseñar que no es viable realizar centenares de miles de reconfiguraciones desde este interfaz debido a cuestiones de tiempo.

La segunda de las interfaces es SelectMAP, que es una interfaz de configuración paralelo con un bus configurable de 8 bits, 16 bits o 32 bits. La reconfiguración completa requiere de no más de decenas-centenares de milisegundos según el tamaño del dispositivo. La tercera de las interfaces, el ICAP (Internal Configuration Access Port) es un puerto interno, y está controlado por lógica implementada dentro de la FPGA. Este interfaz no acepta reconfiguraciones completas, ya que la lógica que está controlando la reconfiguración requeriría ser reconfigurada. Por lo tanto, su uso quedará restringido a aplicaciones de reconfiguración parcial dinámica.

En dispositivos SoC que combinan procesador con FPGA como el Zynq existe una cuarta interfaz denominada PCAP (Processor Contriguration Access Port), que es una interfaz paralelo para configurar la FPGA desde el procesador. Las características de esta interfaz son similares a las de SelectMAP. La particularidad consiste en que no se efectúa la programación desde agentes externos mediante los pines del dispositivo, sino que la lleva a cabo un periférico del sistema procesador denominado DevCfg. En este caso, el bitstream ha de estar almacenado en memoria direccionable del procesador. En uno de los registros de dicho controlador se especifica la dirección de memoria en la que está almacenado el bitstream, y en cuanto se le da la orden de configurar la FPGA, el propio controlador se encarga de generar las señales de control de la interfaz para el correcto envío del fichero.

1.7 Conclusiones

En este capítulo se han introducido los conceptos básicos de las FPGAs en general, y de las de Xilinx en particular. También se ha destacado la relevancia de los

SoCs que combinan procesador con FPGA, cuyo peso en el mercado puede incrementarse notablemente en el futuro próximo. Durante el resto de la tesis se va a hacer referencia en múltiples ocasiones a los conceptos discutidos y analizados en este capítulo.

Uno de los objetivos principales del capítulo es el estudio de la arquitectura global de las FPGAs. Aquí se ha destacado la relevancia de la memoria de configuración, que almacena información muy sensible. Se ha introducido el concepto de SEU en la memoria de configuración, así como los diferentes recursos que pueden ser afectados por este tipo de eventos a nivel de circuito (slices, recursos de rutado, BRAMs, DSP48s, recursos de reloj, etc)

La estructura del bitstream va a ser clave para entender el concepto de emulación de SEUs, que se basa en configurar la FPGA con un fichero intencionadamente corrompido para que tenga el mismo efecto que un SEU sobre la memoria de configuración. Adicionalmente, para llevar a cabo el proceso de emulación de SEU ha de emplearse necesariamente una interfaz de configuración, y en este capítulo han sido introducidas las diferentes interfaces que pueden emplearse para esta finalidad.

La discusión acerca de las fases de diseño también va a ser referida en posteriores capítulos. El concepto de pblock es empleado por otras metodologías de emulación de SEUs. Además, una de las aportaciones de esta tesis es el estudio de la repercusión que las diferentes decisiones a nivel de síntesis e implementación pueden tener sobre la tasa de fallo general del sistema.

Capítulo 2

Conceptos generales de confiabilidad y seguridad

Las continuas innovaciones tecnológicas de las últimas décadas han posibilitado un incremento sustancial en la calidad de vida de las personas. Muchas de estas innovaciones se corresponden con sistemas electrónicos extremadamente complejos con multitud de componentes. Los fallos en este tipo de sistemas pueden provocar interrupciones del servicio, provocando pérdidas económicas, pero también pueden suponer riesgos para las personas o el medio ambiente. Los desarrolladores han de demostrar que las tasas de fallo son extremadamente bajas, para que así los usuarios puedan tener un alto grado de confianza en los sistemas. Para lograr una tasa de fallos aceptablemente baja, los diseños han de estar enfocados hacia confiabilidad y seguridad.

Y éste es, precisamente, el objetivo de este capítulo. Introducir los principales conceptos relacionados con confiabilidad y seguridad, así como discutir su aplicabilidad a sistemas basados en FPGAs. La definición de confiabilidad se toma como punto de partida, así como los parámetros y atributos que han de ser tenidos en cuenta al evaluarla.

Además, se estudia la curva de la bañera, una gráfica muy empleada en la ingeniería de confiabilidad. Partiendo de esta curva se analiza el concepto de ciclo de vida de un sistema. En este punto resalta la etapa de validación, en la que se emplea un esquema conocido como "validación en V". Esto es de gran interés para el desarrollo de la tesis, ya que en ésta se propone una metodología para el testeo de la tolerancia a fallos de circuitos implementados en FPGAs compatible

con este esquema de validación.

Posteriormente, se establece un modelo para representar las amenazas que comprometen la confiabilidad general de un sistema y su propagación hasta provocar interrupciones en el servicio. Una vez indentificadas las amenazas, se estudian las técnicas aplicables para garantizar la confiabilidad desde el punto de vista de ingeniería general.

Tanto las amenazas como los mecanismos se particularizan para sistemas basados en FPGA en los capítulos siguientes. El capítulo 3 aborda los efectos de la radiación, que es la amenaza principal para este tipo de dispositivos. Por otro lado, en el capítulo 4 se estudian las diferentes técnicas para hacer frente a estas amenazas en FPGAs.

La seguridad (safety) se considera un caso particular de la confiabilidad para casos en los que un fallo pueda provocar daños graves a sus usuarios del sistema o al entorno. Lo que obliga al cumplimiento de normativas muy estrictas. Esta sección es de gran interés, ya que muchos de los sistemas que exigen un diseño orientado a confiabilidad han de cumplir con normativas específicas de safety para la industria en cuestión.

2.1 Confiabilidad

2.1.1 Definiciones

Desde el punto de vista de la ingeniería, un **sistema** se define como un conjunto de elementos agrupados como una única entidad que implementa una función concreta. Los sistemas interactúan con otros sistemas, personas y el mundo físico. Todos estos elementos componen el entorno del sistema. Los puntos de interacción entre un sistema y su entorno se denominan fronteras del sistema [43].

El **estado** de un sistema es la condición en la que éste se encuentra en un momento concreto (su condición física, la forma en la que está conectado, los datos e información que almacena, etc). El **comportamiento** del sistema define la forma en la que los estímulos externos condicionan el estado. En sistemas electrónicos, el comportamiento computacional de un sistema se representa como una sucesión de estados discretos. El **estado computacional** del sistema lo conforman los diferentes bits almacenados en los elementos de memoria del sistema. El **estado externo** es la parte del estado visible a los usuarios, mientras que el **estado interno** es la parte no visible.

Desde un punto de vista estructural, un sistema está compuesto de **componentes**. Estos componentes pueden ser otros sistemas (llamados subsistemas) o componentes atómicos, que son los componentes que no se pueden dividir en componentes más pequeños, o que su división no tiene interés. Se puede concluir que el estado total de un sistema es el conjunto de estados externos de sus componentes atómicos.

El objetivo principal de cualquier sistema es su **funcionalidad**. Cumplir las especificaciones funcionales constituye el mayor esfuerzo de ingeniería durante la etapa de diseño. La funcionalidad se corresponde con el servicio que se proporciona a los usuarios, que pueden ser seres humanos u otros sistemas. Los sistemas también pueden solicitar un servicio a otros sistemas, llamados proveedores. Un **servicio** es la sucesión de estados externos percibida por los usuarios. El punto en el que se transmite la información entre proveedores y usuarios se denomina **interfaz** de servicio. En este trabajo, los términos servicio y funcionalidad se emplean en singular, aunque no es raro encontrar sistemas que implementan más de una funcionalidad y proporcionan más de un servicio. En la figura 2.1 se hace una representación esquemática de los conceptos definidos en esta sección.

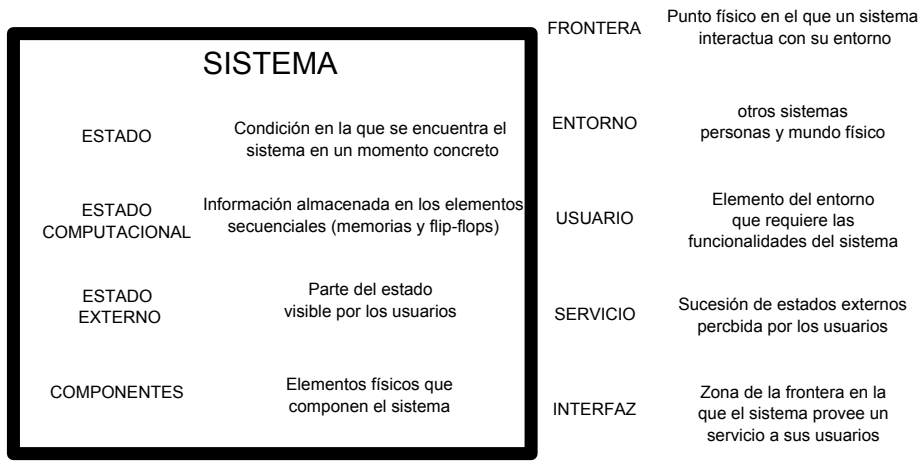


Figura 2.1: Definiciones asociadas al concepto de sistema

Un **servicio confiable** es aquel en el que existen **evidencias** demostrables de que **no va a fallar** más de lo que se considera razonable. Si un servicio es confiable, otros usuarios y sistemas pueden confiar en él. La confianza se define como dependencia aceptada. La dependencia de un sistema A respecto de otro sistema B representa cómo se ve afectada la fiabilidad de A en caso de fallo en B. Si el grado de dependencia entre ambos sistemas es 0, un fallo en B nunca provo-

cará fallos en A; mientras que una dependencia del 100 % indica que cualquier fallo en B provoca un fallo en A. De este modo, en caso de ser B un sistema confiable, puede existir confianza en el servicio que este proporciona. A puede aceptar ser dependiente de B, ya que B es capaz de evitar fallos que sean más severos o más frecuentes de lo que se considera aceptable. Esta definición de **confiabilidad**, al igual que el resto de definiciones mencionadas, está tomada de [43][44], y está recogida dentro de la norma IEC60300 [45].

2.1.2 Atributos y parámetros de la confiabilidad

La cuantificación de la confiabilidad de un sistema se lleva a cabo considerando los siguientes atributos:

- **Fiabilidad** (reliability): Este atributo nos indica cuanto de fiable es la continuidad de un determinado servicio. En otras palabras, la probabilidad de servicio correcto en un instante de tiempo $t = t_1$ siendo el servicio correcto en $t = t_0$. El parámetro que lo identifica es el tiempo medio entre fallos, o MTBF (Mean Time Between Failure).
- **Mantenibilidad** (maintainability): Es la capacidad de un sistema de abordar modificaciones y reparaciones. Se cuantifica mediante el tiempo medio de reparación MTTR (Mean Time To Repair). Representa la probabilidad de servicio correcto en un instante de tiempo $t = t_1$, habiendo un funcionamiento incorrecto en $t = t_0$.
- **Disponibilidad** (availability): Este parámetro representa la disposición de un sistema a proveer un servicio a sus usuarios. Dicho de otra manera: la probabilidad de funcionamiento correcto en un instante de tiempo $t = t_1$, sin tener datos de cómo es el funcionamiento en $t = t_0$. Se cuantifica mediante el AR (Availability Ratio), que es una relación entre el MTBF y el MTTR (ecuación 2.1).

$$AR = \frac{MTBF}{MTBF + MTTR} \tag{2.1}$$

- **Integridad** (integrity): Es la ausencia de alteraciones indeseadas en el estado del sistema o en la información que éste contiene. El parámetro a tener en cuenta es el tiempo medio hasta el fallo MTTF (Mean Time To Fail). La integridad es crítica cuando se desea preservar la existencia del propio sistema por su alto coste, o cuando es usado para almacenar información valiosa.

- **Seguridad** (safety): Este concepto indica la ausencia de consecuencias catastróficas para los usuarios y el entorno del sistema. Su medida se da en términos de FIT (Failures In Time), que indica el número de fallos catastróficos en 10^9 horas.
- **Seguridad** (security): Este concepto, a diferencia del concepto anterior, tiene en cuenta los ataques malintencionados, mientras safety trata los errores provocados por causas naturales o accidentales. Desde la normativa se apunta a security como un atributo de confiabilidad [43]. Sin embargo, su tratamiento es diferente en algunos casos, ya que además de la integridad y la disponibilidad es necesario preservar la confidencialidad de la información.

2.1.3 Curva de la bañera y ciclo de vida

La curva de la bañera es un modelo muy extendido para representar el riesgo de un sistema en función del tiempo [9]. Toma este nombre porque la gráfica adopta la forma de una bañera cortada a lo largo. Representa la idea de que la operación de un sistema se divide en tres periodos claramente diferenciados. En las fases primeras se encuentra un pico de fallos en las fases más tempranas del diseño, al que también se le denomina como "mortalidad infantil". Le sigue un periodo de tasa de riesgo relativamente bajo, denominado "vida útil". Finalmente, la tasa de fallo crece de nuevo, representando los fallos debidos al envejecimiento del sistema.

La curva de la bañera se menciona en la mayoría de los textos relativos a la confiabilidad de sistemas. En su representación clásica aparece como una curva perfectamente simétrica en la que se diferencian nítidamente las tres regiones (figura 2.2). Éste es, precisamente, el objetivo inicial de la curva de la bañera, representar la vida operativa de un sistema mediante una ilustración sencilla y muy explícita. La aplicabilidad de esta gráfica es por tanto reducida, ya que no procede de ninguna expresión analítica o matemática. Otros trabajos hacen una representación más crítica de esta curva [9, 10] (figura 2.3). En estos casos la similitud con una bañera no es tan evidente, pero las expresiones matemáticas que se aportan tienen mayor aplicabilidad.

Los fallos en componentes durante la etapa de mortalidad infantil se deben principalmente a defectos durante el proceso de fabricación [46]. Estos defectos se agrupan en dos grupos, los defectos globales y los locales. Los primeros afectan a toda la oblea de semiconductor, y tienen su origen en partículas contaminantes que se encuentran en el ambiente. Los defectos locales tienen orígenes más variados, como errores humanos, partículas del equipamiento o manchas químicas.

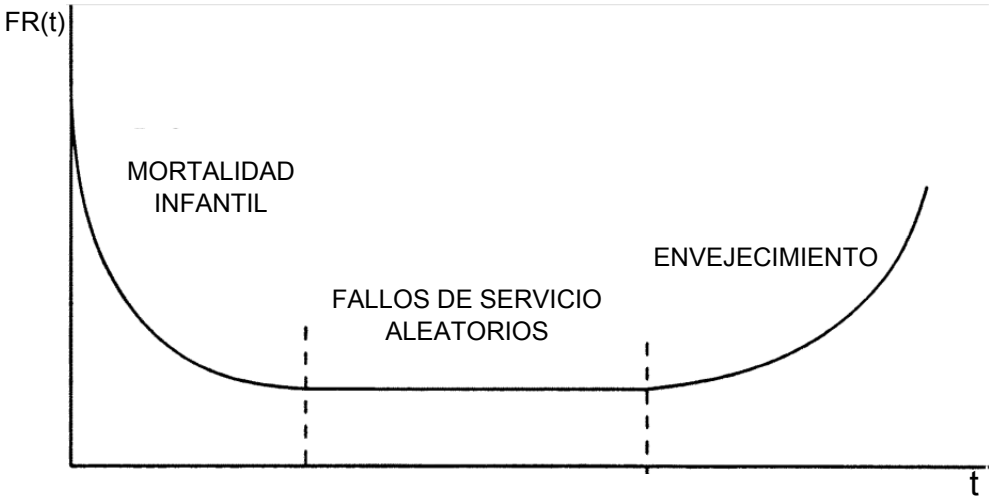


Figura 2.2: Representación clásica de la curva de la bañera [9, 10]

Para evitar este tipo de fallos, las aplicaciones de confiabilidad y seguridad requieren del uso de tecnologías y procesos de fabricación maduros, probados y debidamente documentados. De esta forma, se tiene la certeza de que la tasa de mortalidad infantil es razonablemente baja y el componente es apto para desempeñar su función dentro del sistema. El sistema puede fallar también en fases tempranas debido a errores de diseño, que se minimizan mediante la aplicación de los métodos de verificación y validación adecuados.

La fase de envejecimiento se alcanza en el momento en que la probabilidad de fallo debido al desgaste de los componentes supera los niveles exigidos por la normativa. No es estrictamente necesario minimizar todo lo posible estos efectos, como sí sucede para el caso de mortalidad infantil. Es suficiente con identificar en qué momento van a acentuarse los efectos de la degradación para realizar acciones de mantenimiento preventivo o correctivo en los puntos del sistema que así lo requieran. El envejecimiento no se alcanza de forma abrupta como se representa en la curva de la bañera clásica, sino que se alcanza de una forma más gradual, según se presenta en la figura 2.3.

En los chips basados en transistores MOS (que son la inmensa mayoría de circuitos integrados digitales y FPGAs de la actualidad) hay 4 efectos principales de envejecimiento: la electromigración, los ciclos térmicos, los efectos"hot carrier" y el TDDB (Time-Dependent Dielectric Breakdown) [47]. La electromigración es la degradación de los metales a causa de las densidades de corriente que cir-

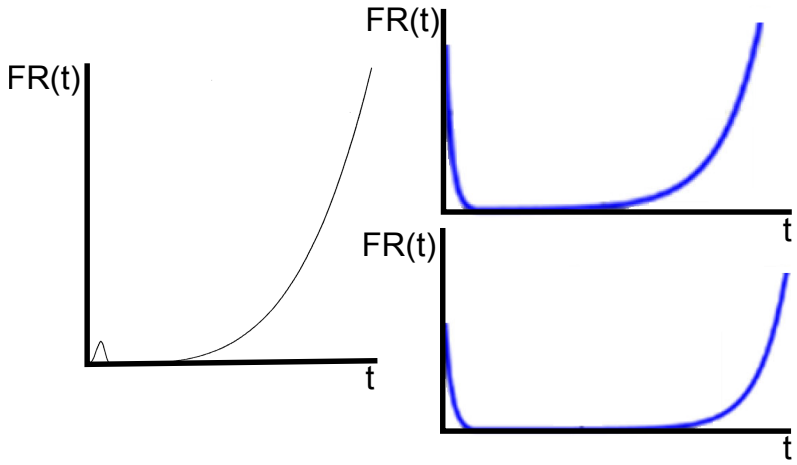


Figura 2.3: Representaciones realistas de la curva de la bañera [9]

culan a través de ellos. El TDDB y los efectos "hot carrier" degradan el óxido que separa la puerta del sustrato de los transistores MOS. Finalmente, los ciclos térmicos producen degradaciones en las características eléctricas y mecánicas del dispositivo. Condiciones climáticas y ambientales adversas tales como temperaturas extremas, humedad, polvo o exposición a la radiación pueden acrecentar la aparición de fenómenos relacionados con la degradación.

Por último se encuentra la fase denominada como vida útil. Durante esta fase la función de riesgo es prácticamente plana, constante y razonablemente baja. Se considera que los errores son aleatorios y causados por eventos del entorno. La probabilidad de error tiene distribución de Poisson, con una tasa de llegadas (λ) suficientemente baja. El tiempo transcurrido entre errores sigue por tanto una distribución exponencial, y está caracterizado por el parámetro MTBF (Mean Time Between Failures). Los sucesos de error se consideran completamente incorrelados entre sí.

Este trabajo se centra en el estudio de la tasa de fallo de un SoC FPGA durante la etapa de vida útil, más concretamente en el estudio de la tasa de fallo de la parte correspondiente a FPGA. Según se especifica en el reporte de fiabilidad de Xilinx [15], la tasa de fallos debido al SEU inducido por radiación es muy superior a la tasa de fallo debido al resto de factores analizados en el reporte. Los efectos de la radiación en FPGAs se estudian en el capítulo 3.

Muy ligado con la curva de la bañera, se encuentra el concepto de ciclo de vida de un sistema. El ciclo de vida es un esquema que representa todas las acciones

tomadas en cada fase de la existencia del sistema. Este concepto está comprendido en las regulaciones y analiza todas las etapas, desde la concepción del sistema hasta su desmantelamiento, pasando por las fases de diseño, test, fabricación, montaje, operación y mantenimiento. En la figura 2.4 se representa el ciclo de vida propuesto por la norma de seguridad 61508. La relación con la curva de la bañera es evidente, pues las acciones de mantenimiento y decomisionado están relacionadas con la fase de envejecimiento. De igual forma, la validación del sistema guarda relación con la fase de mortalidad infantil.

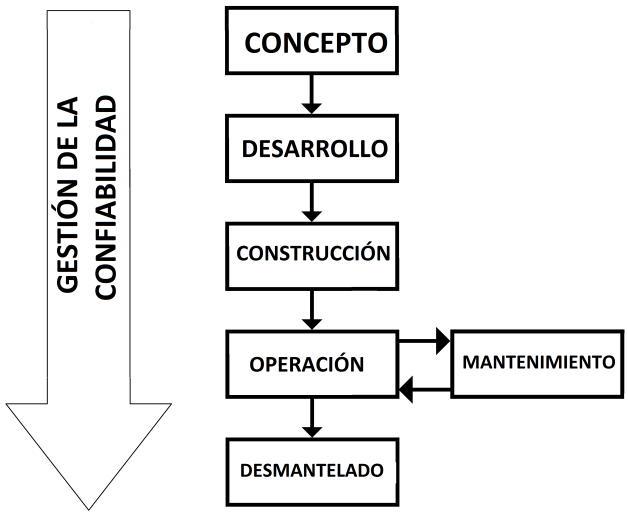


Figura 2.4: Ciclo de vida de un sistema

En sistemas orientados a confiabilidad, es necesario que exista una gestión de ésta en cada etapa del ciclo de vida. La gestión de la confiabilidad es una tarea en sí, y está representada por una flecha a la izquierda en la figura 2.4. Es necesario considerar la confiabilidad desde las etapas iniciales, ya que las medidas tomadas en las primeras etapas del diseño van a tener un gran impacto en la tasa general de confiabilidad. Las distintas regulaciones definen las medidas referentes a confiabilidad que han de ser adoptadas en cada etapa, así como la competencia requerida por el personal implicado y la necesidad de auditoría de confiabilidad por un agente externo.

2.1.4 Validación en V

Uno de los hitos más críticos relativos a confiabilidad es la validación del diseño. Para esto habitualmente se emplea un esquema de validación en V, representado en la figura 2.5 [48, 49]. Esto supone que el diseño comienza como una concepción de especificaciones, y se va concretando durante las etapas posteriores. En el ejemplo de la figura se indican 5 pasos intermedios; arquitectura del sistema, diseño del sistema, diseño de módulos y codificación. El concepto de validación en V indica que los diferentes tests se realizan en orden ascendente, desde el nivel más concreto (componentes, código) hasta el nivel más general (test funcional del sistema). Cada hito en diseño ha de estar acompañado de un test, que será el que valide dicho hito. Este modelo también indica que no se pueden hacer modificaciones después de haber testeado, y en caso de hacerse han de pasar nuevamente por el test.

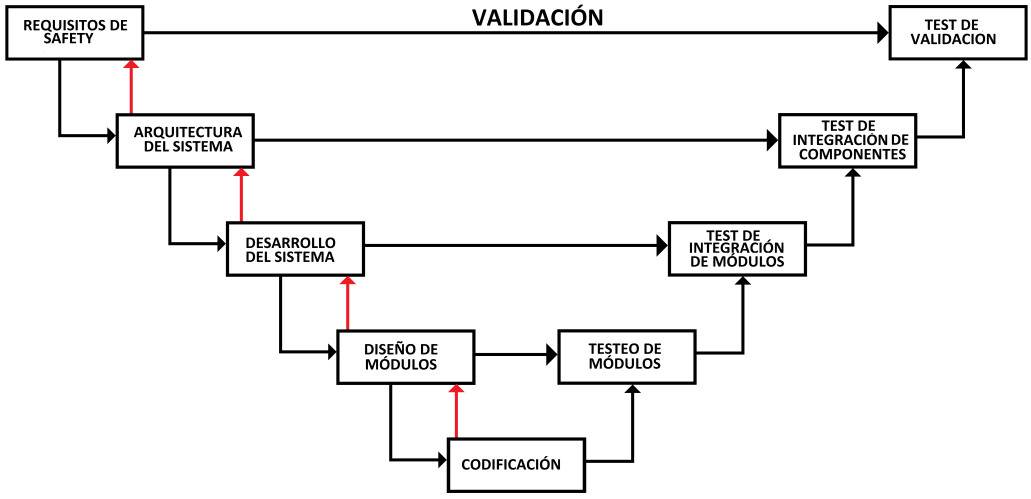


Figura 2.5: Modelo de validación en V

Finalmente, cabe destacar que, en ocasiones, el concepto de validación se confunde con el de verificación. Verificar significa testear el correcto funcionamiento del sistema en un momento concreto, es decir, chequear la presencia/ausencia de errores. El concepto de validación es más amplio. Esto implica que un sistema validado es válido para alcanzar las prestaciones descritas en sus especificaciones. En [50] se menciona que la validación funcional es la parte más crítica de la validación en aplicaciones de automoción, donde señales analógicas complejas han de ser procesadas en tiempo real. En otro tipo de aplicaciones, sin embargo,

la validación de fiabilidad es más crítica. Por ejemplo en aplicaciones espaciales, debido a la imposibilidad de realizar reemplazos.

2.1.5 Amenazas de la confiabilidad

De acuerdo con [43], las amenazas a la confiabilidad de un sistema son los **eventos**, **errores** y **fallos en el servicio** (faults, errors and failures). Un **error** es una alteración impropia en el estado del sistema. Puede corresponderse con un elemento de memoria que almacene información errónea, o con daños o degradación física del propio sistema. Los errores pueden ser transitorios o permanentes. Un error transitorio es aquel cuyo efecto dura por un tiempo limitado, que, por ejemplo, es reparado mediante un reinicio del sistema. Un error permanente es aquel que no puede ser reparado en ningún caso, requiriendo el reemplazo del componente dañado o aceptando una degradación en el servicio.

Los **fallos en el servicio**, denominados en inglés como "system outage" o "failure", suceden cuando los errores se propagan hasta las salidas del sistema, por lo que se interrumpe o se degrada el servicio que está siendo provisto a los usuarios. Un fallo en el servicio puede tener diferentes consecuencias: degradación de calidad del servicio, interrupción total del servicio, pérdida de información, daños a los usuarios (humanos u otros sistemas), etc. Los fallos también se clasifican según su severidad: interrupciones menores, pérdidas económicas, eventos catastróficos y otros. El parámetro que aporta una medida acerca de la confiabilidad del sistema es la **tasa de fallo** o **FR** (Failure Rate), que indica la cantidad de interrupciones en el servicio por unidad de tiempo.

El término conocido en ingés como **fault** es el fenómeno físico o evento que lleva al sistema de un estado correcto a un estado erróneo. Aquí surge un conflicto de carácter semántico, ya que la traducción de "fault" al castellano es también "fallo". Para evitar este tipo de conflictos, a este concepto se le va a denominar como **evento potencialmente peligroso**, o **evento** a secas. Sin embargo existe una excepción en esta nomenclatura. Los conceptos de "fault prevention", "fault tolerance", "fault removal" o "fault forecasting" han sido traducidos al castellano en múltiples ocasiones como prevención de fallos, tolerancia a fallos, eliminación de fallos o previsión de fallos. Por lo tanto, para estos cuatro conceptos el término fallo se asociará como fault. Sin embargo, en el resto del documento, fallo significará failure. Aún así, en muchos casos, al término "failure" se le refiere como fallos en el sistema o fallos en el servicio para evitar ambigüedades.

Los eventos peligrosos pueden deberse a fenómenos naturales o pueden ser provocados por humanos. Entre los **provocados** por humanos nos encontramos los accidentales, los maliciosos y los de diseño. Tanto los eventos accidentales como

los de diseño pueden deberse a insuficiencias de formación de empleados y desarrolladores, y su mitigación pasa por una correcta formación y certificación. La prevención contra ataques maliciosos requiere de la aplicación de distintos tipos de medidas de seguridad, pero su estudio no es el objetivo de este trabajo.

Los **eventos naturales** pueden ser aleatorios o programados. Los programados son aquellos que pueden ser previstos, y se deben al envejecimiento de componentes o al desgaste ante un fenómeno natural concreto, y suelen marcar el final del ciclo de vida del sistema o requerir la implantación de una política de reemplazos adecuada. Los aleatorios son causados por eventos de ocurrencia impredecible, y pueden ocurrir en cualquier punto del ciclo de vida. En caso de producirse degradación debida a una selección de componentes errónea se considera evento de diseño (design fault).

Figura 2.6: Propagación de errores

Los errores se propagan de acuerdo con la figura 2.6. Aquí se aprecia que los "faults" trasladan al sistema a un estado erróneo desde un estado inicial correcto. El error queda latente en el interior del sistema, hasta que finalmente se manifiesta afectando a las salidas. Es en este momento cuando un servicio incorrecto es provisto a los usuarios. En esta figura tanto fault como failure aparecen en inglés para evitar ambiguedades.

2.1.6 Mecanismos para el diseño de sistemas confiables

Esta subsección se centra en el estudio de los diferentes mecanismos existentes para orientar un diseño a confiabilidad. A lo largo de esta subsección el término "fallo" hará referencia al concepto de "fault", mientras que el concepto "failure" se traducirá como "interrupción de servicio" o se reproducirá directamente en inglés. Ésto se hace así porque la traducción más natural y más habitual para los conceptos que se abordan aquí (fault prevention/tolerance/removal/forecasting) es prevención/tolerancia/eliminación/previsión de fallos.

Los mecanismos para el diseño de sistemas confiables se agrupan en 4 grupos de acuerdo con [43]. Cada uno de estas 4 opciones actúa en un punto diferente de la cadena de propagación de errores presentada en la figura 2.6. Los mecanismos de **prevención** pretenden evitar la activación de faltas que lleven al sistema a un

estado erróneo mediante construcciones robustas. Los mecanismos de **tolerancia** permiten que el sistema pueda seguir proporcionando el servicio en presencia de errores mediante esquemas de redundancia. La **eliminación** de fallos se basa en chequear el estado del sistema y reportar posibles "failures" en sus salidas para ejercer acciones de mantenimiento. Finalmente, la **previsión** de fallos pretende hacer una estimación de los posibles fallos que puedan aparecer, para tomar acciones en caso de que sea necesario.

Como puede verse en la figura 2.7, los mecanismos de prevención actúan en la primera etapa de propagación de errores. Se trata de evitar que los fallos se activen, previniendo que el sistema pueda pasar de un estado correcto a un estado erróneo. La prevención de fallos naturales se aborda mediante la utilización de componentes robustos, preparados para soportar eventos naturales concretos. Otra medida aplicable con este fin es el uso de construcciones robustas, utilizando revestimientos y apantallamientos capaces de soportar condiciones ambientales extremas.

La prevención de fallos de diseño tiene su base en la aplicación de metodologías de diseño bien definidas. Un ejemplo es la utilización de técnicas robustas de codificación de software, como es el estándar MISRA-C, unas directrices de programación C para facilitar la seguridad, portabilidad y fiabilidad del software [51]. Otra medida de prevención es la selección de componentes y tecnologías suficientemente maduras, que hayan tenido un recorrido en sistemas con requisitos de confiabilidad similares. Esta filosofía hace que algunos sectores de la industria sean, en muchas ocasiones, reticentes a la incorporación de las tecnologías más novedosas, prefiriendo el uso de tecnologías más probadas.

Figura 2.7: Prevención de fallos

El objetivo de los esquemas de tolerancia a fallos (fault tolerance), tal y como se representa en la figura 2.8, es el de proporcionar un servicio correcto en presencia de errores en el estado del sistema. Los mecanismos de tolerancia a fallos se dividen en dos subgrupos: detección y recuperación de errores y enmascarado de errores. El primero de los grupos consiste en chequear el estado del sistema, y, en caso de detectar incorrecciones, aplicar algún mecanismo para corregirlas. El chequeado del sistema puede hacerse concurrentemente mientras proporciona servicio a sus usuarios, o puede llevarse a cabo durante interrupciones programadas de mantenimiento. Finalmente, el enmascarado de fallos se consigue aplicando

esquemas de redundancia, utilizando varias réplicas de un módulo concreto. De este modo, en caso de producirse un error en una de las replicas, los otros módulos redundantes continúan proporcionando el servicio correcto.

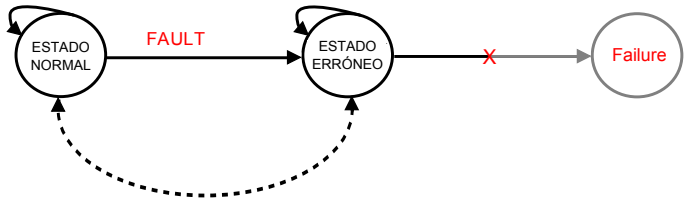


Figura 2.8: Tolerancia a fallos

La eliminación de fallos consiste en verificar el servicio ofrecido por el sistema, y, en caso de detectar deficiencias, subsanarlas mediante acciones previamente previstas. Este planteamiento requiere un análisis previo de las consecuencias de las interrupciones de servicio, ya que éstas no deben ser excesivamente severas o deben poder subsanarse antes de que puedan llegar a causar males mayores. Cuando se detecta una deficiencia en el servicio, han de tomarse medidas de mantenimiento. Para esto hay dos opciones; mantenimiento correctivo, reemplazar los componentes dañados una vez se haya detectado el fallo, o mantenimiento preventivo, consistente en reemplazar los componentes antes de que fallen. La eliminación de fallos también se adopta durante la fase de diseño. Se aplica un proceso de verificación al servicio proporcionado por el sistema, requiriendo un rediseño del mismo cuando el resultado de dicha verificación sea negativo. La implicación de este mecanismo de confiabilidad en la cadena de propagación de errores se ilustra en la figura 2.9.

Finalmente, el mecanismo de previsión de fallos se basa en hacer una evaluación sobre el comportamiento del sistema respecto a la activación de fallos. La evaluación puede hacerse tanto de forma cuantitativa como cualitativa. El propósito de la evaluación cualitativa es identificar, clasificar y jerarquizar los diferentes modos de fallo, así como las distintas combinaciones de fallos en los componentes y condiciones ambientales. Por otro lado, la cuantitativa se centra en el estudio de la tasa de fallos mediante valores numéricos.

En sistemas donde la confiabilidad es un elemento crítico, cualquier mecanismo de prevención, tolerancia o eliminación tiene que estar combinado con un proceso de previsión de fallos, para cerciorarse de que la probabilidad de error está dentro de los rangos permitidos. En muchos casos es imposible evitar todos los posibles errores, pero si se es capaz de justificar que su probabilidad de aparición es

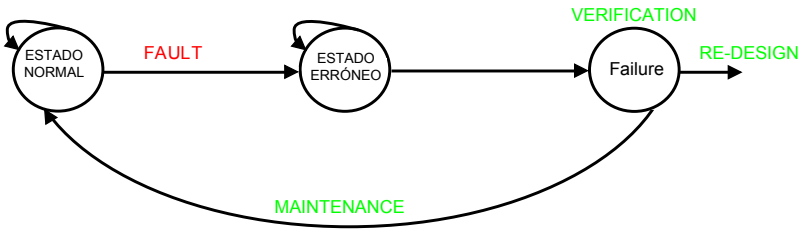


Figura 2.9: Eliminación de fallos

suficientemente baja, el sistema puede ser considerado como confiable.

2.2 Seguridad (Safety)

La seguridad (safety) es un atributo de la confiabilidad que evalúa la habilidad de un sistema para evitar daños a sus usuarios y al entorno. Mientras que la confiabilidad analiza el servicio provisto por el sistema principal, la seguridad estudia la fiabilidad de sistemas extra, añadidos para actuar en caso de interrupciones de servicio. Los sistemas de safety no tienen ningún impacto en la funcionalidad del sistema, sino que incorporan funcionalidades de safety para minimizar los riesgos potenciales.

El proceso de diseño de sistemas de safety se presenta en la figura 2.10. El objetivo es el cumplimiento de unas normativas de seguridad muy estrictas. Para esto, el primer paso es hacer un análisis de safety, que consiste en caracterizar las amenazas del sistema, las consecuencias que éstas pueden provocar y su frecuencia de aparición. Posteriormente se definen los objetivos de safety en concordancia con las normativas. Además de cumplir dichos objetivos, las normativas exigen el cumplimiento de una serie de procedimientos y requisitos, que se van a concretar a lo largo de las próximas subsecciones.

La figura 2.10 refleja el índice de esta sección. En la subsección 2.2.1 se estudia la normativa de confiabilidad, en la 2.2.2 el análisis de safety, la 2.2.3 para los objetivos y la 2.2.4 para los requisitos. En esta sección se utilizan los términos seguridad y safety indistintamente. Se considera que el término seguridad traducido como security, que hace referencia a ataques maliciosos, queda fuera del

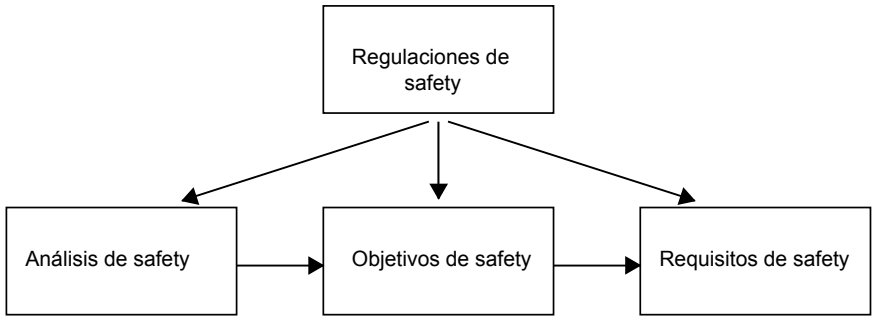


Figura 2.10: Diseño orientado a safety

alcance de esta tesis.

2.2.1 Normativas de seguridad

El origen de las normativas de seguridad y confiabilidad está en los años 70, cuando las compañías industriales interiorizaron que aprender de los errores no era aceptable [52]. En este punto, algunos métodos de identificación de amenazas y cuantificación de sus efectos fueron adoptados. Había preocupación por la falta de estandarización en este tipo de actividades. Los primeros estándares fueron publicados en los años 70 y 80. Finalmente, en la década de los 90 se publicó el estándar de safety IEC61508, un estándar general que hoy en día es la base de la seguridad de sistemas electrónicos. Este estándar especifica los requisitos de seguridad que cualquier sistema orientado a safety debe cumplir.

El estándar IEC61508 tiene por título "Functional Safety of Electrical/Electronic/ Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES)". Traducido al castellano sería algo como "Seguridad funcional de sistemas eléctricos, electrónicos o electrónicos programables orientados a safety". Un equipamiento orientado a safety es aquel cuyos fallos pueden provocar muertes humanas o daños muy graves en el medio ambiente. Si el sistema contiene algún componente eléctrico o electrónico (hoy día prácticamente todos) ha de cumplir con la norma. Siendo IEC61508 el estándar de referencia para sistemas electrónicos, diferentes industrias (automoción, aviónica, aeroespacial, ferroviaria, etc) lo han adoptado como base de sus estándares de safety propios. Algunos de estos se resumen en la tabla 2.1.

La norma IEC61508 especifica los objetivos de confiabilidad y las acciones que han de ser añadidas al ciclo de vida para satisfacer dichos objetivos. Los objetivos

Tabla 2.1: Safety regulations

Normativa	Título
IEC61508	Safety standard for electric, electronic and programmable devices
IEC61511	Electronic devices controling the safety of industrial processes (chemical, refineries, pharmaceutical, paper...)
IEC61513	Requirements and recommendations for the instrumentation and control for systems important to safety of nuclear power plants
IEC62061	Safety of machinery, machinery specific implementation of IEC/EN 61508
ISO26262	Adaptation of the functional safety standard IEC 61508 for automotive electric/electronic Systems
DO-254	Design assurance guidance for airborne electronic hardware
IEC62279	Safety management systems in railway projects
IEC62304	Adaptation of the functional safety standard IEC 61508 for medical systems

pueden ser cualitativos o cuantitativos. Los cuantitativos asignan al sistema una tasa de fallo máxima permitida y están relacionados con los errores aleatorios ocurridos durante la vida útil. Por otro lado, los cualitativos focalizan en evitar errores de diseño, que son imposibles de cuantificar. El estándar define una serie de acciones a llevar a cabo para justificar debidamente la ausencia de este tipo de fallos.

2.2.2 Análisis de safety

El primer paso en el diseño de sistemas cuyos fallos puedan provocar daños a la población o al medio ambiente es llevar a cabo un análisis de safety. Esto comprende la identificación de las amenazas que pueden hacer fallar el sistema, así como la tasa de aparición y severidad de dichos fenómenos. El riesgo puede ser cuantificado desde un punto de vista cualitativo o cuantitativo. Las formulaciones cualitativas usan variables lingüísticas para expresar la frecuencia de un evento (frecuente, probable ocasional, improbable, remoto) y su severidad (catastrófica, crítica, marginal, despreciable). La formulación cuantitativa del riesgo significa el cálculo de la tasa de fallo del sistema, que se obtiene mediante la combinación de las tasas de fallo individuales de los componentes atómicos.

El análisis de safety puede ser descendente (top-down) o ascendente (bottom-up).

Un enfoque descendente consiste en analizar un modo de fallo concreto y determinar qué eventos en los diferentes componentes pueden provocar la activación de dicho modo. El esquema más común de análisis descendente es el FTA (Fault Tree Analysis). Por otro lado, el enfoque ascendente consiste en estudiar los fallos de los componentes elementales y analizar sus efectos en los diferentes niveles de abstracción. Para esto se utiliza la metodología FMEA (Failure Modes and Effect Analysis). Estos métodos formales de análisis se describen en [53–55].

En las etapas tempranas de diseño se utilizan enfoques de análisis de safety descendentes, ya que la arquitectura específica de cada componente puede no estar todavía diseñada. De esta forma se obtienen datos preliminares sobre la fiabilidad del sistema. Una vez el diseño está más avanzado es posible la aplicación de métodos ascendentes que corroboren que los requerimientos de safety están siendo cumplidos.

2.2.3 Establecimiento de los objetivos de safety

Una vez han sido identificados los eventos de riesgo, se definen los objetivos de safety para el sistema en desarrollo. La normativa define diferentes niveles para el equipamiento que realiza funciones de safety, de acuerdo con la severidad y frecuencia de las interrupciones de servicio. Cada nivel define los requisitos en términos de software, hardware y gestión. La norma IEC61508 define 4 niveles de SIL (Safety Integrity Level), siendo el 1 el menos exigente y el 4 el más restrictivo. La primera fase del establecimiento de los objetivos de safety consiste en definir qué nivel de SIL le corresponde a un sistema concreto teniendo en cuenta sus características.

SIL1 tiene que cumplirse en todos los sistemas que puedan provocar lesiones a personas, mientras que SIL4 es obligatorio en sistemas capaces de provocar múltiples muertes o catástrofes medioambientales. IEC61508 no exige únicamente una tasa de fallo determinada, sino también la adopción de una serie de medidas para evitar fallos sistemáticos. Las tablas 2.2 y 2.3 muestran el nivel de SIL requerido en función de la severidad y la frecuencia de aparición.

El estándar IEC61508 hace distinción entre sistemas de alta demanda y baja demanda. Los sistemas de baja demanda son aquellos cuyo periodo de uso es superior al periodo de revisión. Por ejemplo, un accidente que requiera la activación del airbag de un coche sucede una vez cada 50 años, mientras que el periodo medio de revisión es alrededor de 2 años. Por otro lado, el sistema de frenado del coche es un ejemplo de sistema de alta demanda, cuyo periodo de uso es de un minuto, muy inferior al de revisión. Los sistemas de alta demanda exigen una

Tabla 2.2: Nivel de SIL requerido en función de la severidad y la probabilidad de los riesgos

Severidad	Probabilidad baja	Probabilidad moderada	Probabilidad alta
Catastrófico	3	3	4
Crítico	2	3	3
Marginal	1	2	3
Despreciable	NR	1	2

Tabla 2.3: Safety integrity levels (SILs)

Safety-Integrity Level	Tasa de alta demanda (failures/hr)	Tasa de baja demanda (probabilidad de failure bajo demanda)
4	$\geqslant 10^{-9}$ to $< 10^{-8}$	$\geqslant 10^{-5}$ to $< 10^{-4}$
3	$\geqslant 10^{-8}$ to $< 10^{-7}$	$\geqslant 10^{-4}$ to $< 10^{-3}$
2	$\geqslant 10^{-7}$ to $< 10^{-6}$	$\geqslant 10^{-3}$ to $< 10^{-2}$
1	$\geqslant 10^{-6}$ to $< 10^{-5}$	$\geqslant 10^{-2}$ to $< 10^{-1}$

tasa de fallo muy inferior a los de baja demanda, según se puede observar en la tabla 2.3.

Los sistemas de baja demanda, también denominados sistemas orientados a safety, están destinados a mejorar la seguridad global del sistema. Al contrario de los sistemas de alta demanda, no tienen ninguna incidencia sobre el servicio que el sistema provee. Estos sistemas "safety related" se encargan de llevar al sistema a un estado denominado como "seguro" cuando una amenaza se activa. Si el sistema orientado a safety resulta dañado, es probable que sea reparado en la próxima revisión, muy por encima de la probabilidad de activación de la amenaza. Los sistemas de alta demanda tienen un uso continuado. Por tanto, cuando éstos fallan, se produce un fallo a nivel de sistema global.

2.2.4 Requisitos para el diseño de sistemas safety-related

Además de los objetivos de tasa de fallo y aplicación del principio ALARP, la aplicación de la normativa de safety requiere el cumplimiento de una serie de requisitos a nivel de hardware, software y gestión. Los requisitos anotados en esta subsección se corresponden con la norma IEC61508. Como ya se ha comentado previamente, la mayoría de estándares de seguridad específicos de las distintas industrias parten de este estándar.

2.2.4.1 Requisitos a nivel de gestión

Las regulaciones de safety especifican las acciones a llevar a cabo a nivel de gestión de proyecto. El primer requisito exigido es probar la evidencia de que el equipo de desarrollo tiene un nivel de competencia adecuado para llevar a buen término las diferentes tareas. Se exige demostrar que la plantilla tiene los conocimientos técnicos y de ingeniería adecuados, además de conocimientos de safety y conocimientos legales. La relevancia de las diferentes certificaciones y la experiencia previa han de estar convenientemente documentados.

Las normativas de seguridad requieren que el proceso de diseño esté auditado por un ente independiente al equipo de desarrollo. Este proceso de auditoría es más severo o menos dependiendo del nivel de SIL requerido. La norma IEC61508 requiere la evaluación de una persona independiente en los niveles SIL1 y SIL2, un departamento independiente para SIL3 y una organización independiente para SIL4. La gestión a nivel de safety exige la documentación adecuada de todas y cada una de las acciones del ciclo de vida del sistema. Un evaluador no puede certificar el cumplimiento de la normativa si esta documentación no existe.

2.2.4.2 Requisitos a nivel de hardware

El estándar de safety IEC61508 especifica el grado de separación a nivel físico y eléctrico de los sistemas de safety del resto del sistema. De esta forma se pretenden evitar fallos de causa común, que se producen cuando un evento de riesgo afecta por igual al sistema principal y al sistema orientado a safety. SIL1 y SIL2 requieren una documentación precisa de las fronteras de separación, definiendo adecuadamente las interfaces de datos y eléctricas. Además, la separación física ha tomarse en consideración como posibilidad para incrementar la seguridad del sistema. SIL3 requiere separación física entre el equipamiento principal y el equipamiento de safety, SIL4 exige que la separación entre estos entes sea completa. Entre las entidades redundantes del sistema también es requerido un nivel de separación similar.

El estándar también especifica el nivel de redundancia requerido para cada nivel de SIL, dependiendo del SFF (Safe Failure Fraction) del sistema. El SFF representa el porcentaje de fallos que provocan fatalidades. El SFF puede ser estimado por dos vías, por inyección de errores o por métodos formales (FMEA o FTA). La próxima tabla 2.4 expresa el nivel de redundancia requerida en función del SFF y el nivel de SIL. En esta tabla, el parámetro m define la cantidad de fallos que provocan la caída del servicio. (m+1) significa que hay un solo sistema redundante, mientras que (m+2) se corresponde con dos sistemas redundantes

extra.

Tabla 2.4: Redundancia requerida

SFF	SIL for Simplex	SIL for (m+1)	SIL for (m+2)
<60 %	1	2	3
60-90 %	2	3	4
90-99 %	3	4	4
>99 %	3	4	4

El estándar IEC61508 también define más aspectos a nivel de hardware. Cómo tratar con elementos adquiridos externos o con subsistemas "legacy" y cómo llevar a cabo las pertinentes modificaciones y operaciones de mantenimiento. Para esto, se aportan una serie de plantillas para su adecuada documentación.

2.2.4.3 Requisitos a nivel de software

Los errores sistemáticos son la mayor amenaza al desarrollar software confiable. Esto se produce por situaciones de valores de variables y entradas no contempladas durante la validación. El mecanismo fundamental para evitar este tipo de errores es un testeo muy profundo. El parámetro que indica la calidad de un test es el "coverage", que indica el porcentaje de combinaciones de entradas y estados testeados respecto del total. La validación del software se lleva a cabo por un proceso de validación en V, que ha sido anteriormente analizado.

El estándar de safety IEC61508 hace las siguientes recomendaciones para el desarrollo de software en entornos de seguridad. El software debe estar estructurado en módulos pequeños y manejables. Cuando sea posible, se recomienda el uso de módulos de software ampliamente probados y verificados. El uso de objetos dinámicos no está permitido en software orientado a confiabilidad, ya que su testabilidad y auditabilidad es reducida desde herramientas "offline". Las bifurcaciones incondicionales y la recursividad también están prohibidas, y la cantidad de punteros e interrupciones está limitada. El código ha de seguir un estándar de codificación específico para evitar el uso no seguro y mal estructurado del lenguaje de programación. El estándar más conocido para el lenguaje C es el MISRA-C [51].

2.3 Conclusiones

En este capítulo se han introducido los conceptos de confiabilidad y seguridad, en torno a los cuales gira en gran medida el contenido de esta tesis. En primer lugar se define el concepto de confiabilidad, y se detallan los atributos que miden el grado de confiabilidad de un sistema. Posteriormente se analiza la curva de la bañera, que es una de las representaciones más comunes de los riesgos en las diferentes fases de la existencia del sistema.

Se detallan las amenazas que comprometen la confiabilidad de un sistema, y se presenta un esquema sobre su propagación hasta provocar la interrupción en el servicio. Sobre este esquema se desarrollan los diferentes mecanismos existentes para hacer frente a dichas amenazas. Tanto las amenazas como los mecanismos de seguridad se particularizan para FPGAs en los próximos capítulos. El capítulo 3 estudia los efectos de la radiación en este tipo de dispositivos, mientras que los medios para combatirlos y cuantificarlos se debaten en los capítulos 4 y 5 respectivamente.

La mayoría de las veces en las que los diseños se orientan a confiabilidad se hace para cumplir con las normativas de seguridad que requiere la industria. La gran mayoría de estas normativas tienen como base la norma IEC61508, cuyos fundamentos se describen en este capítulo. Esta norma indica la tasa de fallo máxima que puede tener un sistema. Sin embargo, para cumplir con la normativa no basta con alcanzar dicha tasa, sino que también hay que cumplir con una serie de requisitos a nivel de hardware y software. Además, se ha de demostrar la competencia del personal desarrollador. Todo esto ha de ser adecuadamente documentado a fin de ser evaluado por un ente competente.

De este capítulo cabe resaltar el proceso de validación en V, ya que la aportación principal de esta tesis está directamente relacionada con este punto. El concepto de validación en V se discute a lo largo del capítulo 5, para evaluar la compatibilidad de otras metodologías con este esquema. De igual modo, durante el capítulo 6 se describe una metodología de inyección de errores compatible con este modelo.

Capítulo 3

Efectos de la radiación en FPGAs

La **radiación** se define como la propagación de energía en forma de ondas electromagnéticas o partículas subatómicas a través del vacío o de un medio material. la propagación de ondas se conoce como radiación electromagnética, mientras que la de partículas se denomina radiación corpuscular. El impacto de la radiación electromagnética en circuitos electrónicos es un fenómeno ampliamente estudiado, y tiene una reglamentación muy estricta. Este campo de estudio se conoce como EMI (ElectroMagnetic Interference), y estudia el impacto de las interferencias electromagnéticas sobre los circuitos electrónicos. Sin embargo, su análisis queda fuera del alcance de este trabajo.

La **radiación corpuscular** comprende la propagación de partículas subatómicas a través del espacio. Dichas partículas se generan como producto de reacciones nucleares, que pueden ser terrestres o cósmicas. El impacto de este fenómeno es mayor en misiones espaciales, debido a que la atmósfera protege en gran medida la superficie terrestre de este tipo de eventos. De todas formas, los efectos de estas partículas no pueden ser ignorados en los sistemas electrónicos terrestres, ya que la atmósfera no protege plenamente. A partir de aquí, cada vez que se indique el término "radiación" se hará referencia a la radiación de partículas.

Este capítulo se centra en el estudio de la radiación de partículas y el análisis de su impacto en las FPGAs. El objetivo de la primera sección es introducir los conceptos básicos de la radiación y relacionarlos con los sistemas electrónicos. La segunda sección analiza los diferentes entornos de radiación, que se clasifican

en espaciales y terrestres. Finalmente, la tercera y última sección estudia los diferentes tipos de FPGAs utilizadas en entornos de radiación hostiles, haciendo hincapié en los datos de radiación de las FPGAs de Xilinx, que van a ser utilizadas a lo largo del presente trabajo.

3.1 Fundamentos de la radiación

Esta primera sección se divide en dos partes. En la primera se describen los parámetros que definen la radiación de partículas en dispositivos electrónicos desde un punto de vista físico: la energía, el LET (Linear Energy Transfer), el flujo y la sección de cruce. La segunda parte se centra en los efectos que se inducen en sistemas electrónicos en general y en FPGAs en particular. En esta segunda parte el estudio se hace desde un punto de vista de ingeniería, describiendo las consecuencias de la radiación en los dispositivos a nivel funcional.

3.1.1 Parámetros fundamentales de la radiación

La radiación corpuscular comprende la propagación de partículas subatómicas a través del espacio. Cuando dichas partículas inciden sobre un dispositivo semiconductor, se considera que ha ocurrido un evento de radiación. El parámetro fundamental que va a determinar las consecuencias de dicho evento es la **energía de la partícula** incidente. Esto sucede debido a la estructura interna de la materia. A nivel subatómico, la materia está compuesta por estados discretos, y para pasar de un estado a otro, es necesario superar umbrales energéticos concretos. Por ejemplo, para generar un par e-h en silicio se requieren 3.6eV. Las energías típicas de las partículas de radiación que afectan a los semiconductores van desde 1MeV hasta 500MeV, aunque una mínima minoría de rayos cósmicos espaciales puede llegar hasta los 10000MeV.

Además, es necesario conocer de qué forma se va a transferir la energía desde la partícula incidente al semiconductor. Esta transferencia puede hacerse por dos mecanismos principales, por tránsito y por espalación.

- **Tránsito:** Consiste en que las partículas ionizadas que atraviesan el semiconductor van transfiriendo energía a su paso, generando un reguero de pares e-h. Estos portadores generados pueden hacer conducir uniones p-n en inversa, generando corrientes no deseadas. El parámetro que define la cantidad de energía transferida durante el tránsito es el **LET** (Linear Energy Transfer). Este parámetro se expresa en MeV/cm, y cuantifica la cantidad de energía transferida por unidad de longitud. El valor de LET depende

de tres factores: la energía de la partícula incidente, el tipo de partícula (protones, electrones, iones, etc) y el tipo de material atravesado. En [56] se aborda el cálculo del LET para diferentes iones cuando atraviesan silicio, donde se concluye que cuanto más pesada y más energética es la partícula mayor es la transferencia de energía al semiconductor. El LET por sí solo es un indicador de la cantidad de energía transferida, ya que lo más frecuente es que las partículas atraviesen el semiconductor sin transferir toda su energía, siendo la distancia recorrida igual al espesor del silicio.

- **Espalación:** La partícula incidente interacciona con los núcleos de los átomos del propio silicio. La partícula radiactiva suele tener energía suficiente para provocar la descomposición del átomo, fenómeno conocido como reacción nuclear de espalación. Fruto de esto se generan partículas secundarias ionizadas que cruzan el semiconductor generando pares e-h. Este es el fenómeno dominante en la radiación de neutrones, ya que estas partículas no están cargadas y no transmiten energía de forma directa durante el tránsito [57]. En [58] se indica que la transferencia de energía de un protón en tránsito no es muy relevante, y que éstos transfieren su energía mayoritariamente por espalación, de forma similar a como lo hacen los neutrones.

Una vez conocida la energía del evento, y la cantidad de ésta que se transfiere al dispositivo, es necesario conocer la frecuencia de ocurrencia de dichos eventos. Esto viene marcado por el **flujo de radiación**, que indica la cantidad de partículas que inciden contra un cuerpo durante un periodo de tiempo determinado. Éste se expresa como el número de partículas incidentes por unidad de tiempo y superficie. Habitualmente se presenta en función de la energía (MeV), o del LET, ya que la energía de las partículas incidentes puede ser diferente.

Esto significa que un entorno de radiación queda caracterizado completamente al conocer el flujo de cada tipo de partículas en función de su energía. También se puede deducir la cantidad de energía transferida al dispositivo si se conoce el material de éste. Sin embargo, la aplicación de estos efectos es poco práctica, ya que el conocimiento de la cantidad de energía transferida y pares e-h generados no permite identificar los efectos reales en el funcionamiento del sistema.

Para evaluar el daño real producido en la electrónica, se define el parámetro conocido como **sección de cruce** (σ) (cross section). Se trata de un área probabilística (expresada en cm^2) que indica la probabilidad de ocurrencia de un evento potencialmente peligroso. Al multiplicar la sección de cruce por el flujo (ϕ) se obtiene la **tasa de eventos** (τ), expresada en número de eventos peligrosos por unidad de tiempo. La sección de cruce es función de la energía de la partícula, ya que a mayores energías es más probable que el evento peligroso se produzca. Por tanto, la probabilidad del evento viene dada por la integral en el dominio de

la energía del producto entre la sección de cruce y el flujo de radiación (ecuación 3.1). Este parámetro va a ser clave para el cálculo de la **tasa de eventos** (τ), que indica la cantidad de eventos potencialmente peligrosos que ocurren en un periodo de tiempo concreto. De la ecuación 3.1 el valor de τ se obtiene en unidades del sistema internacional (eventos/segundo). Sin embargo, es más habitual la presentación de este parámetro en términos de FIT que indica la cantidad de eventos en 10^9 horas (ecuación 3.2).

$$\tau = \int \sigma(E) * \phi(E) * dE \tag{3.1}$$

$$\tau(FIT) = \tau * 3600 * 10^9 \tag{3.2}$$

3.1.2 Efectos provocados por la radiación

Los efectos producidos por la radiación se clasifican en dos grupos: Por un lado se encuentran los efectos acumulativos, y por otro los efectos de evento único. La acumulación de radiación soportada por un dispositivo produce una degradación de su funcionamiento. La cantidad total de radiación máxima que puede soportar un dispositivo a lo largo de su vida útil es el TID (Total Ionizing Dose).

Los efectos de evento único, conocidos en inglés como SEE (Single Event Effects), tienen lugar cuando una única partícula de radiación impacta contra el semiconductor. Mayoritariamente, los SEE producen fallos de caracter reversible, como modificaciones en los bits de memoria. Sin embargo, tienen implicaciones particulares en caso de una FPGA, ya que los efectos sobre la memoria de configuración suponen un fallo permanente en el nivel de circuito implementado. Un SEE también puede provocar daño irreversible en el hardware en el caso de partículas de muy alta energía mediante un mecanismo conocido como SEL (Single Event Latch-up).

3.1.2.1 Dosis Total Ionizante (TID)

La exposición continua a la radiación, principalmente en misiones espaciales, provoca degradación temporal en los dispositivos electrónicos. Los problemas de degradación se acentúan a medida que aumenta la dosis total acumulada en el dispositivo. Debido a esta degradación, se define el parámetro conocido como TID (Total Ionizing Dose), que indica el máximo de radiación que puede soportar un dispositivo electrónico en el conjunto de su vida operativa.

En [59] se especifica que la parte de los dispositivos electrónicos más vulnerable al TID es el óxido que separa la puerta de los transistores MOS del sustrato de silicio. Cuando una partícula de radiación incide sobre estas zonas, se generan pares e-h. En óxidos como el SiO_2, la movilidad de los electrones es mucho mayor que la de los huecos, por lo que los primeros llegan a la frontera del material en un periodo de tiempo corto. Por otro lado, los segundos requieren un tiempo mayor para alcanzar el silicio. Además, algunos huecos pueden quedar atrapados en el óxido sin alcanzar la frontera y recombinar, lo que provoca que el material se cargue positivamente con el tiempo.

Esta carga parásita que aparece en el óxido de la puerta del transistor MOS hace variar la tensión umbral para que el transistor conmute. La tensión umbral se reduce para los transistores de canal n y aumenta para los del canal p. En el momento en que el umbral de conmutación no coincida con las tensiones de alimentación del circuito, el dispositivo dejará de funcionar, apareciendo un daño permanente e irreversible.

3.1.2.2 Single Event Effects (SEE)

El SEE (Single Event Effect) es un efecto que engloba los eventos que ocurren cuando una única partícula de radiación incide en un semiconductor. En contraposición con el TID, los SEE son efectos aleatorios. No tienen dependencia temporal, es decir, su probabilidad de aparición es constante mientras el sistema está en operación. En su mayoría, los SEE provocan errores de carácter transitorio, de los que el sistema es capaz de recuperarse. Cuando una partícula ionizada cruza un semiconductor va generando un reguero de pares e-h a su paso. Si estos pares e-h se generan cerca de una unión p-n inversamente polarizada, los portadores pueden cruzar dicha unión, generándose un pequeño pico de corriente no deseada.

El SEE más relevante en lo relativo a FPGAs es el SEU (Single Event Upset). Un SEU es la alteración repentina de una celda de memoria, debido a que la corriente producida por la partícula de radiación introduce o extrae carga de dicha celda. Si el umbral de carga del biestable se rebasa, el valor almacenado se vuelve erróneo. Lo más habitual es que cada SEE tenga solamente impacto sobre un bit, lo que se denomina SBU (Single Bit Upset). Sin embargo, en ocasiones, la energía de la partícula incidente es tan alta que puede influir sobre 2 o más bits, lo que se conoce como MCU (Multiple Cell Upset). En [16] se especifica que el término MBU (Multiple Bit Upset) se utiliza para definir a MCUs que afectan a 2 o más bits de la misma palabra lógica (Byte, Word o Frame según el caso).

En el caso particular de las FPGAs, el SEU puede afectar a ambas capas: tanto

a la capa de circuito implementado como a la memoria de configuración. La mayoría de celdas de memoria de la capa de circuito implementado de la FPGA se corresponden con los flip-flops de las CLB y los bloques de memoria. Un error en estos recursos probablemente significará una interrupción en el servicio, especialmente si afecta a información sensible. Sin embargo, es posible proteger el diseño frente a estos eventos mediante codificaciones robustas (Hamming, ECC) o haciendo volver al sistema a un estado conocido (Reset, Rollback, Rollforward) (véase sig. capítulo).

Un SEU también puede afectar la memoria de configuración de la FPGA, lo que se traduce en una modificación del circuito implementado. Este efecto se conoce como SEFI (Single Event Functional Interrupt), ya que el sistema no es capaz de cumplir con las funcionalidades para las cuales ha sido diseñado hasta que no haya una reconfiguración del dispositivo. Sin embargo, un SEU en la memoria de configuración solamente provoca un mal funcionamiento en caso de afectar a un recurso utilizado. No todos los bits de configuración tienen un impacto crítico en el diseño ya que todos los recursos configurables no están siendo utilizados por el sistema.

Cuando el SEE afecta la lógica combinacional se genera un SET (Single Event Transient). Esto es un pico de corriente repentino que puede provocar un SEU secundario en caso de que coincida con un flanco de reloj. La probabilidad de un SEU secundario inducido por un SET se incrementa con la frecuencia, siendo especialmente relevante en el rango de múltiples gigahercios. En las frecuencias de trabajo habituales de una FPGA no suele tener especial importancia [60].

Otro tipo de SEE es el SEL (Single Event Latch-up), que sucede cuando el evento hace conducir el transistor BJT parásito entre sustrato y pozo de una estructura CMOS. Para que esto suceda es necesario el impacto de una partícula de muy alta energía. Este efecto es potencialmente destructivo y requiere al menos un apagado del sistema para recuperarse. Su impacto en entornos terrestres es prácticamente despreciable, pero puede suceder en algunas misiones espaciales como consecuencia de los rayos cósmicos.

3.2 Entornos de radiación

Los entornos de radiación principales son el espacial y el terrestre, y son discutidos en profundidad en la presente sección. El entorno espacial es el más complicado, ya que la atmósfera terrestre frena en gran medida las partículas radiactivas. Dentro del entorno terrestre se van a diferenciar dos regiones principales. Por un lado la superficie terrestre, donde tienen lugar la mayor parte de aplicaciones

electrónicas, y por otro el entorno de aviónica, ya que en las capas superiores de la atmósfera los índices de radiación son mayores.

3.2.1 Radiación Espacial

Los entornos espaciales han sido los primeros que han requerido un estudio sobre el impacto de la radiación sobre componentes electrónicos. Los flujos de radiación son mucho más potentes en el espacio, ya que la atmósfera terrestre mitiga en gran medida este tipo de fenómenos. En esta sección se analizan las diversas partículas existentes en este entorno, así como su influencia sobre la electrónica en diferentes órbitas y misiones espaciales.

3.2.1.1 Partículas de radiación en el espacio

En el entorno espacial, hay tres fuentes fundamentales de radiación: [61] [62]

- Eventos solares, eyecciones de masa coronal
- Rayos cósmicos, GCR (Galactic Cósmic Rays)
- Partículas atrapadas en la magnetosfera

Las partículas procedentes de eventos solares tienen su orígen en la atmósfera solar. Ésta está compuesta por un gas totalmente ionizado, lo que se conoce como estado de plasma. Las partículas están en completo movimiento, lo que se denomina como vientos solares. La energía cinética de estas partículas es tal que en ocasiones pueden escapar del campo gravitacional solar. El sol está compuesto mayoritariamente por hidrógeno, que en estado de plasma se compone de un protón y un electrón totalmente ionizados. Por tanto, los eventos de viento solar se componen mayoritariamente por protones y electrones de alta energía. En ocasiones, estos vientos solares pueden expulsar del campo gravitacional del sol ingentes cantidades de materia, que pueden superar el tamaño de la tierra. Este fenómeno se conoce como "eyección de masa coronal".

Cuando estas partículas eyectadas por el sol se dirigen hacia la tierra, se encuentran con la magnetosfera. La magnetosfera de un planeta es el punto de su atmósfera en el que las partículas emitidas por la estrella se desvían al entrar en contacto con el campo magnético de dicho planeta. De esta forma, la mayoría de estas partículas son desviadas por la fuerza de Lorentz, impidiendo que alcancen la superficie terrestre. Debido a esto, algunas de estas partículas quedan atrapadas orbitando en la magnetosfera.

La mayoría de las partículas atrapadas están localizadas en los llamados cinturones de Van Allen. Hay dos cinturones: el interno, que se encuentra entre 1000 km y 5000km de la superficie de la tierra, y el externo, que se encuentra entre los 15000 km y los 20000 km (la órbita geoestacionaria a 36000 km no queda afectada). En estas zonas de la atmósfera es frecuente encontrar partículas ionizadas a modo de plasma, que al incidir sobre los semiconductores pueden provocar los efectos analizados en el apartado anterior. En estas localizaciones, normalmente, no se encuentran órbitas satelitales. Sin embargo, los protones y electrones de estos cinturones pueden desprenderse afectando a vehículos espaciales en otras órbitas.

Debido a esto, la mayoría de la radiación que afecta al equipamiento electrónico en entornos espaciales se compone de protones y electrones, ya provengan del sol directamente o de la magnetosfera. En [61] se comenta que la energía de estos electrones suele ser bastante baja (hasta 7 MeV), lo que hace que no crucen las carrocerías de los vehículos espaciales. Por tanto, la partícula de radiación más relevante en entornos espaciales es el protón. Las energías de los protones espaciales pueden alcanzar los 500MeV.

Los rayos cósmicos, conocidos en inglés por las siglas GCR (Galactic Cosmic Rays) , son partículas subatómicas de muy alta energía. El origen de estas partículas está en el espacio profundo, fuera del Sistema Solar. Su energía es tan alta que son capaces de superar los campos magnéticos del sol, y penetrar en el Sistema Solar. Esta característica hace que también puedan superar la magnetosfera y adentrarse en la atmósfera terrestre. El flujo de GCR es dependiente de la actividad solar, ya que ésta modula la magnetosfera solar, provocando variaciones en la cantidad de GCRs que consiguen adentrarse en el Sistema Solar.

Según se indica en [61], los GCR no tienen impacto en el TID, ya que el flujo de estas partículas no es lo suficientemente elevado. Sin embargo, estas partículas provocan SEUs, que en ocasiones pueden ser MCUs y SELs debido a su alta energía.

3.2.1.2 Impacto de la radiación en misiones espaciales

Existen datos acerca de los flujos de radiación en las diferentes zonas y órbitas del espacio, que son tomadas en cuenta para hacer estimaciones de la fiabilidad de los sistemas embarcados en los satélites. Una de las plataformas para acceder a estos datos es el software de la ESA SPENVIS (Space ENVironment Information System) [63]. En dicho software el usuario introduce datos de la órbita y se proporcionan informaciones de los flujos de radiación de protones y rayos cósmicos, así como datos de la dosis de radiación total por año.

Como ejemplo de misiones espaciales que embarcan FPGAs y hacen una estimación de la tasa de SEU, se encuentran las misiones [64] y [65]. La primera de ellas se trata de una plataforma llamada NLCE (Netherlands China Low frequency Explorer), embarcado en el satélite chino Chang-e 4. Este satélite orbita en el punto de Langrage L2 del sistema tierra-luna. La tasa de fallos aportada para una memoria de 20Mb es de 4.4 SEU/semana, con picos de 3.6 SEU/5min.

Por otro lado, en [65] se analiza el entorno del satélite RazakSAT. Un satélite enviado a la órbita NEqO (Near Equatorial Orbit), a 685 km de la tierra y con una inclinación de 9 grados. Este satélite fue lanzado en 2009, y un año más tarde se perdió la comunicación con él. En el artículo se estudia el SEU en celdas SRAM de 6 transistores de 180nm, valorando la posibilidad de incluir un apantallamiento de $0.5g/cm^2$ de aluminio. Se concluye que la tasa de fallo se debe principalmente a partículas procedentes de eventos solares (SEP: Solar Event Particles). Su influencia es 4 órdenes de magnitud superior a la de los GCR. El impacto de los protones atrapados es todavía inferior (sobre 6 órdenes de magnitud inferior). El estudio indica tasas de fallo en torno a los 10^{-6} SEU/bit*día para alimentación de 1V con apantallamiento.

3.2.2 Radiación Terrestre

Además de en sistemas espaciales, la probabilidad de fallo debido a la radiación también ha de ser tenida en cuenta en sistemas terrestres que requieran una alta tasa de fiabilidad. En este caso, el impacto de la radiación es notoriamente inferior. En contraposición con los entornos espaciales, no se requiere el uso de dispositivos especialmente diseñados para soportar la radiación, lo que facilita enormemente la posibilidad de utilizar los dispositivos más recientes procedentes de la electrónica comercial.

Sin embargo, ciertos sistemas requieren justificar tasas de fallo extremadamente bajas. Esto sucede en el caso de los estándares SIL3 y SIL4, que exigen una tasa de fallo inferior a 100 FIT y 10 FIT respectivamente. Este tipo de sistemas requieren cuantificar el impacto de la radiación terrestre sobre sus sistemas y circuitos electrónicos en aras de justificar que cumplen con la normativa, o aplicar técnicas de tolerancia a fallos en caso de ser necesario. Tanto la reducción continua de las tecnologías de fabricación, como la disminución de las tensiones de alimentación, hacen disminuir la cantidad de carga almacenada en cada biestable. Este hecho aumenta considerablemente la probabilidad de aparición del SEU. Por ello, cada vez que una nueva tecnología de fabricación es lanzada al mercado, su resiliencia a la radiación ha de ser testeada.

Con el objeto de unificar este tipo de test, surge el estándar JEDEC JESD89

[11]. JEDEC (Joint Electron Device Engineering Council) es una de las principales entidades de estandarización de dispositivos de estado sólido. El estándar JEDEC JESD89 tiene como título "Measurement and Reporting of Alpha Particle And Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices". El estándar cubre en detalle las tres causas de error predominantes en el entorno terrestre; los neutrones de alta energía, las partículas alfa y los neutrones térmicos. Se proporcionan medidas de los flujos de radiación y se presentan fórmulas y tablas para obtener las tasas de fallo en diferentes situaciones.

Algunos sistemas terrestres muy específicos han de trabajar directamente con radiación generada por el hombre, y la electrónica de estos sistemas ha de ser capaz de proporcionar unas condiciones de confiabliidad adecuadas. Por tanto, es necesaria una caracterización pormenorizada de este tipo de entornos, como sucede en algunos entornos médicos [66] o en aceleradores de partículas [58]. A continuación se van a discutir las principales fuentes de radiación en entornos terrestres, que son los neutrones de alta energía, los neutrones térmicos y las partículas alfa.

3.2.2.1 Neutrones de alta energía

La principal fuente de radiación en entornos terrestres es el neutrón atmosférico, también denominado neutrón de alta energía. Esta partícula se genera en las capas más exteriores de la atmósfera. Los rayos cósmicos (GCR) colisionan con los átomos del aire (nitrógeno y oxígeno), provocando reacciones nucleares en cascada de todo tipo, siendo el neutrón atmosférico uno de los productos más relevantes de estas reacciones.

Según el estándar, estos neutrones tienen una energía entre 1 MeV y 10 GeV. [11]. El flujo es notoriamente mayor para energías bajas (1 -100 MeV), sin embargo, los neutrones de energías más altas tienen mayor probabilidad de provocar eventos dañinos, ya que generan más pares e-h, (la sección de cruce es mayor). Por tanto, todas las energías han de ser tenidas en cuenta para obtener el dato real de tasa de fallo. El gráfico de la figura 3.1 muestra el flujo de neutrones en función de la energía para la ciudad de Nueva York, los valores se obtienen mediante la ecuación 3.3.

$$\begin{aligned}\frac{d\phi_{NYC}(E)}{dE} &= 1{,}006 * 10^{-6} * e^{-0{,}35(ln(E))^2+2{,}1451*ln(E)} + \\ &+ 1{,}011 * 10^{-3} * e^{-0{,}4106(ln(E))^2-0{,}667*ln(E)}\end{aligned} \tag{3.3}$$

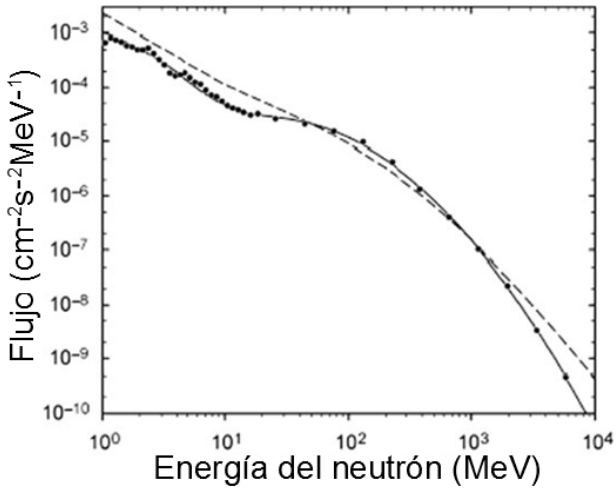


Figura 3.1: Flujo de neutrones ciudad de NY según el estándar JEDEC JESD98 [11]

La cantidad de partículas y su distribución energética varían con la altitud, [67], ya que los neutrones se generan en las capas más exteriores de la atmósfera. De todos estos neutrones solo unos pocos alcanzarán la superficie terrestre, y el flujo de neutrones se incrementa a medida que la altitud aumenta. Este hecho tiene relevancia en la industria de aviónica, ya que la concentración de neutrones puede llegar a ser entre 100 y 1000 veces la observada en la superficie terrestre. Como la tasa de fallos es proporcional al flujo de radiación, este hecho indica que las tasas de fallos son centenares de veces superiores en entornos aviónicos respecto de entornos de superficie terrestre.

Además de la altitud, el flujo de neutrones depende de otros dos factores, la latitud y la actividad solar. La causa de la dependencia con la latitud es la posición respecto de la magnetosfera. En latitudes concretas, los rayos cósmicos tienen mayor probabilidad de quedar atrapados en los cinturones de Van Allen, lo que reduce el flujo en la superficie terrestre. La actividad solar también modula el flujo de neutrones. A mayor actividad solar, mayor es el campo magnético del Sistema Solar, lo que dificulta la llegada de los rayos cósmicos.

El impacto de estos tres factores (Altitud, actividad solar y latitud) se cuantifica en el estándar mediante la fórmula 3.4. El flujo en la ciudad de Nueva York $d\phi_{NYC}$ se multiplica por los factores (F_A y F_B), que están tabulados en

el estándar y dependen de la altitud, la posición geomagnética y la actividad solar. Mediante esta fórmula se obtiene $d\phi_{LOC}$, el flujo de neutrones para una localización cualquiera.

$$\frac{d\phi_{LOC}(E)}{dE} = \frac{d\phi_{NYC}(E)}{dE} * F_A(d) * F_B(R_C, l, d) \tag{3.4}$$

En [12] Se presenta un gráfico realizado con estos mismos datos en el que se aprecian los flujos de neutrones (figura 3.2). Se aprecia que la relevancia de la altitud, que incrementa a 60000 pies en tres órdenes de magnitud el flujo en la superficie terrestre. La varianza respecto de la latitud no introduce tanta variabilidad.

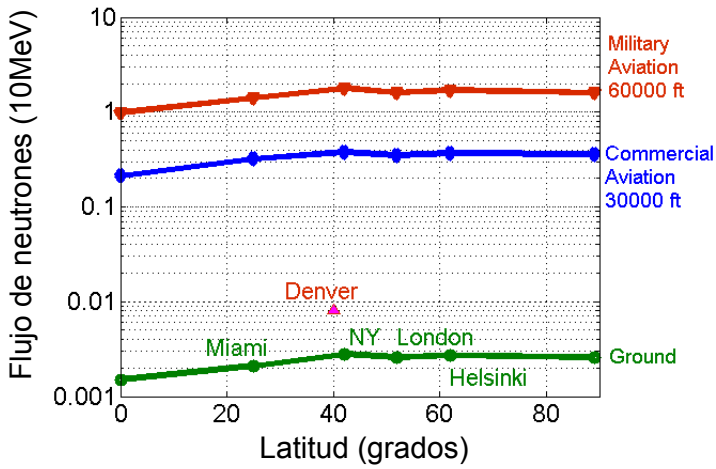


Figura 3.2: Flujo de neutrones de alta energía a diferentes latitudes y altitudes,[12]

3.2.2.2 Partículas alfa y Neutrón térmico

Aunque el neutrón de alta energía sea el factor dominante en la tasa de fallo total, especialmente en aplicaciones de aviónica, existen dos fuentes de error adicionales que no pueden ser obviadas; las partículas alfa y los neutrones térmicos. Las partículas alfa son núcleos de helio, que están compuestas por dos protones y dos neutrones. Estos iones generan pares e-h mientras transitan por el material, ya que son partículas cargadas. La distancia media que recorre una partícula alfa de

10MeV en el silicio es inferior a las 100 micras, lo que imposibilita que partículas externas al encapsulado alcancen el semiconductor.

A finales de los 70, estas partículas eran la primera amenaza a la confiabilidad de las memorias DRAM, debido a que las impurezas de uranio y torio que se encontraban abundantemente en el encapsulado del dispositivo emitían este tipo de partículas al descomponerse [57]. Los procesos de fabricación se han ido puliendo, hasta alcanzar una emisividad de 0,001 partículas alfa por cm^2 hora [57], partiendo desde las 100 partículas por cm^2 hora. Con estos valores, las partículas alfa suponen aproximadamente el 20 % de los SEE en entornos terrestres [15] (ciudad de NY). La emisividad alfa no aumenta en entornos más hostiles (aviónica, espacio), por lo que el impacto de las partículas alfa es mínimo en comparación con los flujos de protones para estos entornos.

Además del neutrón de alta energía y las partículas alfa, existen otras partículas de origen terrestre susceptibles de provocar SEEs, los neutrones térmicos. Éstos son neutrones cuya energía es muy baja (<1MeV), lo que hace que no puedan escapar del material una vez hayan entrado. Estas partículas interaccionan con un isótopo del boro presente en los dispositivos para generar los perfiles de dopado del semiconductor. Cuando un neutrón térmico interacciona con un átomo de boro-10, se genera una desintegración radiactiva, cuyos productos son un ion de litio y una partícula alfa. El tránsito de estos productos a través del semiconductor es el que provoca la generación de pares e-h, así como la aparición de eventos no deseados. Este fenómeno tiene un efecto muy relevante cuando aparece un material llamado BPSG (Borophosphosilicate Glass), que las nuevas tendencias de fabricación han conseguido eliminar en gran medida.

En contraposición con los neutrones de alta energía, resulta muy complicado elaborar perfiles de flujo de neutrones térmicos, ya que dependen en gran medida de los materiales y las condiciones del entorno en el que opera el sistema. Para la cuantificación del impacto en dispositivos electrónicos, se realizan tests de radiación en laboratorio, y los resultados se cotejan con dispositivos funcionando en entornos reales.

3.3 FPGAs empleadas en entornos hostiles

En esta sección se analizan los diferentes tipos de FPGAs utilizadas para conseguir los objetivos de confiabilidad marcados. La primera parte de la sección se centra en el estudio de diferentes tecnologías para la memoria de configuración (Flash, antifuse, SRAM). Asimismo, se valora la opción de utilizar FPGAs comerciales sustituyendo a las endurecidas. La segunda parte se centra en la

caracterización de la sección de cruce de FPGAs. Se estudian las diferentes metodologías posibles para abordar los diferentes componentes de la tasa de fallo relativos a cada entorno de radiación. Además, se presentan los resultados de SEU para las FPGAs de Xilinx, con las que se va a trabajar en esta tesis.

3.3.1 Diferentes tecnologías de FPGAs

Una medida para prevenir la aparición de faltas en FPGAs es la utilización de dispositivos más robustos. Esto tiene especial relevancia en la industria aeroespacial, donde las tasas de SEU debido a la radiación pueden llegar a ser extremadamente altas. Aquí encontramos FPGAs de tipo Flash o antifuse, que evitan la aparición de SEUs en la memoria de configuración. La utilización de FPGAs de tecnología SRAM queda restringida a casos concretos donde estas altas tasas de fallos debidas a la radiación puedan ser debidamente cuantificadas y compensadas.

3.3.1.1 FPGAs de tecnología antifuse

La programación de este tipo de FPGAs se hace por medio de antifusibles. Éstos se componen por dos cables metálicos separados por un dieléctrico. En el momento en que se aplica una tensión elevada entre ambos cables, el dieléctrico rompe y se produce un cortocircuito. Durante la fase de programación unos antifusibles estratégicamente escogidos son programados de modo que se implementen las funciones lógicas establecidas por el diseño. Esta tecnología es de naturaleza OTP (One Time Programmable), ya que sólo puede programarse una vez. Esta característica complica bastante la fase de diseño, que requiere de una FPGA de tipo SRAM o Flash equivalente para realizar las pruebas pertinentes.

Esta tecnología no utiliza memorias para almacenar la configuración. Por tanto, es imposible que puedan producirse SEUs en la memoria de configuración. No obstante, los registros del sistema están hechos en tecnología CMOS, y éstos son susceptibles de ser afectados por SEUs. Para menguar la incidencia de este efecto, los fabricantes incluyen construcciones robustas. Por ejemplo, las FPGAs RTAX-D de Actel triplican cada flip-flop mediante una configuración TMR [68].

3.3.1.2 FPGAs de tecnología flash

Otra posibilidad es el uso de FPGAs cuya memoria de configuración sea de tipo flash [69]. La celda elemental de esta tecnología es el FG-MOSFET (MOSFET de puerta flotante). Esta tecnología es no volátil, buena para la miniaturización

(1Transistor/celda) y versátil. Se utiliza en un gran rango de aplicaciones, desde FPGAs espaciales a pendrives comerciales. La programación de estas celdas de memoria se consigue mediante una tensión muy elevada en la puerta del transistor, haciendo conducir el oxido para introducir o extraer electrones de la puerta flotante. Esta necesidad de un voltaje muy elevado en la puerta hace prácticamente imposible rebasar el umbral digital de un biestable como consecuencia de un SEU inducido por radiación.

Hay tests de radiación independientes para FPGAs de tipo flash, y los resultados empujan el uso de estos dispositivos como alternativa a soluciones antifuse. En lo referente al SEU tienen tan buena resistencia como las antifuse. Según [69] en las FPGA flash se puede añadir un "design overhead" que en vez de empeorar mejore la tolerancia SEUs (RHBD Rad-Hard By Design).

Sin embargo, las FPGAs de tipo flash tienen una serie de problemas de confiabilidad que hay que corregir convenientemente:

- Hot carrier injection: Daño en el óxido al introducir y extraer electrones de la puerta flotante (Programar y desprogramar). No tiene mucha relevancia en misiones espaciales porque se programan solamente una vez.
- Exposición al TID: Se daña el óxido por la exposición continuada a la radiación, puede llegar a conducir y escaparse los electrones de la puerta flotante. Este problema no existe en las FPGAs de tipo antifuse, ya que no hay oxidos de puertas flotantes que puedan dañarse.

Por tanto, debido al impacto del TID en las FPGAs de tipo flash, los dispositivos antifuse son más recomendables que los flash en misiones muy expuestas a la radiación, ya sea por su duración o por estar en una órbita con un alto flujo de radiación [70].

3.3.1.3 FPGAs rad-hard de tecnología SRAM

Además de las antifuse y las flash, existen FPGAs de tecnología SRAM endurecidas (hardened), especialmente preparadas para entornos hostiles en términos de radiación. Como ejemplos de este tipo de dispositivos encontramos las Virtex4QV y Virtex5QV de Xilinx [71]. Estas FPGAs de teconología rad-hard cuentan con celdas SRAM de 12 transistores en la memoria de configuración, que proporcionan unos buenos niveles de mitigación contra el SEU [72]. La lógica de control de configuración del dispositivo se encuentra triplicada para dotar a esta parte tán crítica de inmunidad frente a SEEs. Estos dispositivos rad-hard también están robustecidos frente a efectos potencialmente destructivos como el SEL y el TID mediante un encapsulado más robusto y una capa epitaxial protectora.

Estas características hacen que este tipo de dispositivos sea adecuado para conseguir altos niveles de confiabilidad en entornos de radiación hostiles. Está reportado en [72] que las FPGAs de estas familias pueden tolerar TIDs de hasta 1000krads. Con la tecnología SRAM de 12 transistores se consiguie una mejora de tres órdenes de magnitud respecto a SEUs en la memoria de configuración, en comparación con la familia Kintex7. Además, cabe destacar la madurez de estos dispositivos: la Virtex5QV fue lanzada en 2010, mientras que la Virtex4QV data de 2008. Esta tecnología goza de un elevado grado de madurez, ya que ha sido testeada en multiples ocasiones y ha sido embarcada en infinidad de misiones espaciales.

Sin embargo, el empleo de este tipo de tecnología supone un alto coste en términos de comportamiento y prestaciones de diseño. En la última década, la evolución de las FPGAs ha sido muy importante. La tecnología de fabricación ha pasado de los 90nm (Virtex4QV 2008) a los 16nm (familia ultrascale 2016). La familia kintex7 cuenta con hasta 3.7 veces más recursos que la Virtex5QV debido a la mayor densidad de integración [72]. El consumo de potencia es 3 veces mayor en la familia Virtex5 respecto de la Kintex7 por el elevado consumo de las celdas SRAM de 12 transistores y las tensiones de alimentación más reducidas de los dispositivos modernos. Éste es un parámetro crítico en sistemas embarcables, que disponen de un presupuesto de potencia al que se han de ajustar. Las familias nuevas también permiten el uso de herramientas de diseño más modernas (Vivado), así como diferentes recursos integrados únicamente en FPGAs modernas (recursos de reloj, RocketIO, microprocesadores integrados, ZYNQ...).

Las FPGAs rad-hard de Xilinx Virtex4QV y Virtex5QV no incorporan protección contra SEUs en los flip-flops ni en las BRAM, por lo que se hace necesaria la aplicación de mecanismos de mitigación.

3.3.1.4 FPGAs COTS (Commertial Off The Shelf)

La electrónica comercial ha avanzado a gran velocidad en las últimas décadas, ofreciendo a los desarrolladores múltiples capacidades de diseño. Es por esto que surge el concepto de COTS (Commercial Off The Shelf), que se basa en la utilización de dispositivos de rango comercial en industrias como la aeroespacial, la aviónica y la militar. Este tipo de industrias son recelosas a la introducción de nuevas tecnologías que no hayan sido específicamente diseñadas para este tipo de entornos. Por ello, uno de los grandes desafíos actuales es la validación de confiabilidad para que los sistemas industriales, aviónicos, aeroespaciales y militares puedan aprovecharse de las multiples ventajas de la cada vez más pujante electrónica comercial. El concepto COTS surge en contraposición a los conceptos

MOTS (Military off the shelf) y GOTS (Government off the shelf), que se basaban en tomar la tecnología gubernamental y la militar como base, cuando éstos eran los ambitos pujantes de la innovación científica.

En [72] se hace la comparativa entre las familias Virtex5QV y Kintex7 para una misión espacial concreta (se menciona que la ausencia de datos de radiación para los dispositivos COTS dificulta bastante este estudio). Para la familia Kintex no existe un valor máximo de TID soportada (1000Krad para Virtex5QV). Se añade que el TID no debería suponer un mayor problema, ya que puede ser mitigado mediante un encapsulado metálico a costa de añadir más peso. La mitigación del SEL se antoja más problemática, ya que tiene su origen en los GCR, que no pueden ser frenados por el encapsulado. En el estudio se concluye que la probabilidad de éxito de la misión espacial de 15 años es de un 65 % debido a este factor. De todas formas, misiones espaciales actuales se han decantado por el uso de FPGAs de tipo COTS, como es el caso de la misión NCLE (Netherlands-China Low-frequency Explorer) [64].

La probabilidad de SEU en la memoria de configuración es entre 100 y 1000 veces mayor en las FPGA COTS que en las rad-hard como la Virtex5QV. Esto exige necesariamente la aplicación de medidas de tolerancia a fallos para mitigarlo. Las medidas más relevantes son el TMR (sección 4.2) y el scrubbing (Sección 4.3). Las BRAM y los flip-flops también son vulnerables, requiriendo de medidas de protección específicas.

De todos modos, la continua reducción de las tecnologías de fabricación, el incremento en frecuencia y la bajada de las tensiones de alimentación, están haciendo crecer la vulnerabilidad a SEUs incluso de dispositivos comerciales diseñados para entornos terrestres. Este hecho hace que los fabricantes traten de introducir continuamente innovaciones a nivel de silicio para revertir este efecto. Un ejemplo es la tecnología FinFET incluída por TSMC (Taiwan Semiconductor Manufacturing Company) en las celdas SRAM de 16nm [73]. Este hecho hace que las celdas SRAM actuales superen a sus predecesoras de tecnología planar en términos de tolerancia al SEU.

Como apunte novedoso para el futuro, en [74] se proponen las FPGAs de tipo MRAM (Magnetic RAM), que proporcionan una elevada resiliencia a los SEEs.

3.3.2 Caracterización de la sección de cruce y tasa de SEU de diferentes FPGAs

A lo largo de esta tesis se desarrolla una metodología para la caracterización del impacto de los SEUs en diseños implementados en FPGAs. El objetivo final va

a ser el cálculo de la tasa de fallo del sistema, que se va a representar con las siglas FR (Failure Rate). Este valor se calcula mediante la ecuación 3.5, como el producto entre la tasa de SEUs del dispositivo (τ_{SEU}) y un factor dependiente del diseño, denominado como DVF (Design Vulnerability Factor).

$$FR = \tau_{SEU} * DVF \tag{3.5}$$

La tasa de SEUs (τ_{SEU}) indica la cantidad de SEUs que afectan al dispositivo por unidad de tiempo. Al igual que la tasa de eventos (τ), presentada en la sección 3.1.1, y caracterizada por la ecuación 3.1, τ_{SEU} se puede presentar en unidades de eventos/segundo o en unidades de FIT. La diferencia entre ambas tasas consiste en que τ_{SEU} solamente considera SEUs por unidad de tiempo, mientras que τ tiene en cuenta cualquier evento de SEE (SEU, SET, SEFI, SEL). El cálculo de τ_{SEU} se lleva a cabo mediante la sección de cruce de SEUs (σ_{SEU}) y la ecuación 3.6. σ_{SEU} es el equivalente a σ en la ecuación 3.1, pero considerando únicamente SEUs en lugar de cualquier tipo de SEE.

Tanto σ_{SEU} como τ_{SEU} se dan en ocasiones normalizadas por bit de memoria (ecuaciones 3.7 y 3.8). Por lo general todas las FPGAs de una familia concreta son idénticas a nivel microelectrónico. Debido a esto, σ_{bit} y τ_{bit} permiten la caracterización de una familia de FPGAs completa. Para obtener σ_{SEU} y τ_{SEU} de un dispositivo concreto, es suficiente multiplicar σ_{bit} y τ_{bit} por el tamaño de la memoria del dispositivo en cuestión.

$$\tau_{SEU} = \int \sigma_{SEU}(E) * \phi(E) * dE \tag{3.6}$$

$$\sigma_{bit} = \frac{\sigma_{SEU}}{bits\ de\ memoria} \tag{3.7}$$

$$\tau_{bit} = \frac{\tau_{SEU}}{bits\ de\ memoria} \tag{3.8}$$

El DVF, por su parte, indica la fracción de SEUs que provocan fallos en el servicio. Teóricamente puede tomar valores entre 0 y 1. Sin embargo, difícilmente se va a superar el valor de 0.1, tal y como se va a explicar más adelante. Esta sección se centra en la estimación de la tasa de SEUs de una FPGA, mientras que la estimación del DVF va a ser objeto de discusión durante los siguientes capítulos.

En primer lugar se destaca la aceleración de radiación. Esto consiste en aplicar un flujo de partículas radiactivas en laboratorio. De este modo, se inyectan flujos de radiación millones de veces superiores a los del entorno real, acortando

dramáticamente los tiempos de test. Estos tests requieren de un equipamiento y de unas instalaciones muy específicas. La alternativa es el real-time test, que consiste en observar el funcionamiento de una gran cantidad de chips durante un largo periodo de tiempo. Este método es sumamente costoso en términos de tiempo y cantidad de recursos.

Ambas metodologías son empleadas también para la estimación de la tasa de fallo de memorias SRAM comerciales. Sin embargo, según se detalla en [75], no es conveniente extrapolar los datos de memorias comerciales a FPGAs. En este artículo [75], se explican las diferencias entre FPGAs y memorias SRAM comerciales en lo referente a su tolerancia al SEU. A continuación se procede a la discusión de estas metodologías.

3.3.2.1 Aceleración

La teoría sobre los tests de aceleración de radiación es muy simple. Se hace incidir un haz de partículas radiactivas sobre el dispositivo en un laboratorio y se monitoriza el estado de la memoria. Transcurrido un tiempo se detiene el haz de radiación, y se cuenta la cantidad de bits que han sido afectados por un SEU. La sección de cruce (σ_{SEU}) se obtiene aplicando la ecuación 3.9. La sección de cruce es la división entre la cantidad de SEUs y el producto entre el flujo del haz de radiación y el tiempo de test. En ocasiones, como en el reliability report de Xilinx, la sección de cruce se da por bit, dividiendo el valor de la ecuación anterior entre el número total de bits de memoria. De este modo, es posible estimar la susceptibilidad al SEU de dispositivos de diferentes tamaños (ecuación 3.9).

$$\sigma_{SEU} = \frac{Fallos\ detectados}{flujo * tiempo} \tag{3.9}$$

El estándar JEDEC-JESD98 para el estudio de la radiación terrestre propone 3 tipos de haces de radiación para la evaluación del impacto del neutrón atmosférico: haces monoenergéticos de protones, haces monoenergéticos de neutrones y haces de neutrones de varias energías. Las partículas alfa y los neutrones térmicos también son testeadas en el laboratorio. Xilinx testea las partículas alfa en sus propios laboratorios utilizando torio 232, y el neutrón térmico en los laboratorios MNRC (McClellan Nuclear Research Centre). Los haces de neutrones de energías múltiples son los más representativos del entorno de radiación terrestre. El problema radica en que existen pocas instalaciones en el mundo capaces de realizar estos tests. Una de ellas es el LANSCE (Los Alamos Neutron Science Center), donde Xilinx realiza sus tests. Sin embargo, en [76] se concluye que el flujo de protones de 64MeV es representativo del estudio del neutrón atmosférico.

Para el estudio de entornos de radiación espaciales, en [77] [78] se hace este estudio mediante un haz de iones, que permite obtener la respuesta al flujo de protones mediante los métodos presentados en [79] [80]. Una de las instalaciones que permite este tipo de tests es el laboratorio Lawrence Berkeley, en el que se usa un cocktail de iones de diferentes LETs que pueden emular diferentes entornos de radiación espaciales.

3.3.2.2 Test real-time

Esta técnica consiste en monitorizar el estado de un gran número de dispositivos durante mucho tiempo. Con las tasas de fallo de los sistemas actuales en entornos terrestres (entre 100-10000FIT), se requieren 1000 dispositivos trabajando durante 1000 horas de promedio para observar un único fallo. Estos requerimientos han de ser multiplicados por 10-100 para obtener datos estadísticamente representativos. Esta técnica solo tiene sentido para entornos terrestres, pues resulta inviable llevar a cabo semejantes operaciones para entornos espaciales. Aquí no se consiguen datos acerca de la sección de cruce, sino que se obtiene directamente la tasa de fallo del sistema.

El estándar JEDEC-JESD98 [11] indica los procedimientos para aislar las tres componentes de la tasa de eventos de la radiación terrestre. La tasa de eventos por radiación alfa se obtiene mediante tests subterráneos, donde no puedan llegar ni los neutrones atmosféricos ni los térmicos. El neutrón térmico se puede aislar mediante apantallamientos especiales de boro, cadmio o gadolinio. Este tipo de tests es llevado a cabo por Xilinx mediante el llamado experimento Rosetta [81]. Los resultados, distinguiendo entre fallos por radiación alfa, neutrón atmosférico y neutrón térmico, se incluyen en el reliability report de Xilinx [15].

3.3.2.3 SEUs en FPGAs de Xilinx para entornos terrestres

En esta tesis se trabaja fundamentalmente con las FPGAs de Xilinx. Esta decisión se toma debido a la transparencia que demuestra Xilinx respecto del SEU en su memoria de configuración, publicando los resultados de sus campañas de inyección de fallos. Todas estas campañas están resumidas en la tabla 3.1, extraída del reliability report de Xilinx [15]. Se aportan datos de la sección de cruce obtenida por inyección de neutrones de energías múltiples en LANSCE. También se aportan datos en FIT/Mb referentes al neutrón térmico, las partículas alfa y SEU total incluyendo neutrón atmosférico medidos en tiempo real.

Uno de los parámetos clave para la caracterización del comportamiento de las FPGAs frente a la radiación es la tasa de eventos general del dispositivo (τ_{bit}). Este

parámetro, cuya unidad es FIT/Mb, indica la cantidad de SEUs en la memoria de configuración en 10^9 años. En primer lugar se obtiene la tasa de SEU/Mb a partir del valor de sección de cruce LANSCE. Para ello, hay que multiplicar por el flujo de neutrones integrado para la ciudad de NY (13 neutrones/cm^2*h) y por 10^6bits (1024*1024). El valor de (τ_{bit}) se consigue multiplicando por 10^9 h/1FIT y dividiendo por el tamaño de la CRAM. Con este sencillo cálculo se aprecia que los valores medidos en LANSCE concuerdan con los valores medidos por test en tiempo real.

Tabla 3.1: SEUs en FPGAs de Xilinx en CRAM [15]

Tecnología	Familia	LANSCE, neutrón alta energía, cross section (σ_{bit})	Neutrón térmico (FIT/Mb)	Partículas alfa (FIT/Mb)	Tiempo real (τ_{bit}) (FIT/Mb)
180 nm	Virtex-E	$1{,}12 * 10^{-14} \pm 18\,\%$			$181 \pm 20\,\%$
150 nm	Virtex-II	$2{,}56 * 10^{-14} \pm 18\,\%$			$405 \pm 20\,\%$
130 nm	Virtex-II pro	$2{,}74 * 10^{-14} \pm 18\,\%$			$437 \pm 20\,\%$
90 nm	Virtex-4	$1{,}55 * 10^{-14} \pm 18\,\%$			$263 \pm 20\,\%$
90 nm	Spartan-3	$2{,}40 * 10^{-14} \pm 18\,\%$			$190 \pm 20\,\%$
90 nm	Spartan-3E	$1{,}31 * 10^{-14} \pm 18\,\%$			$104 \pm 20\,\%$
65 nm	Virtex-5	$6{,}70 * 10^{-15} \pm 18\,\%$			$165 \pm 20\,\%$
45 nm	Spartan-6	$1{,}00 * 10^{-14} \pm 18\,\%$	$21 \pm 13\,\%$	$88 \pm 100\,\%$	$177 \pm 20\,\%$
40 nm	Virtex-6	$1{,}26 * 10^{-14} \pm 18\,\%$	$0{,}7 \pm 13\,\%$	$7 \pm 97\,\%$	$105 \pm 20\,\%$
28 nm	Artix-7	$6{,}99 * 10^{-15} \pm 18\,\%$	$29 \pm 4\,\%$	$43 \pm 80\,\%$	$73 \pm 20\,\%$
28 nm	Kintex-7	$6{,}99 * 10^{-15} \pm 18\,\%$	$1{,}1 \pm 18\,\%$	$43 \pm 80\,\%$	$63 \pm 20\,\%$

En esta tabla se puede observar que las tasas de SEU debidas a la radiación aumentaban al mismo tiempo que iban desarrollándose nuevas familias de FPGAs. A medida que la tecnología de fabricación se iba reduciendo la tasa de SEUs pasó de 181 FIT/Mb para 180nm a 437 FIT/Mb para 130 nm. Es en este momento cuando los fabricantes toman conciencia en la problemática que supone bajar en tecnología de fabricación, lo que lleva consigo celdas de memoria más pequeñas, que almacenan memos carga efectiva y son más vulnerables al entorno. A partir de ahí, cada tecnología nueva de fabricación añade avances en esta materia, hasta llegar a las FPGAs actuales, con unas tasas de FIT/Mb más reducidas. Sin embargo, las FPGAs actuales son más grandes, y como la cantidad de Mbs en la CRAM es mayor, la tasa de SEU del dispositivo es similar.

En esta tesis se ha trabajado con la Zynq7020, cuya FPGA es de tipo Artix7 y tiene una CRAM de 20Mb. La tasa de fallo según real time es de 73 FIT/Mb $\pm 20\,\%$, según LANSCE es de 90,87FIT/Mb. Tomando el promedio entre ambos (82 FIT/Mb), la tasa de SEU de la memoria de configuración completa del

Zynq7020 es de 1640 FIT. Es decir, un SEU en el CRAM cada 69,6 años.

3.4 Conclusiones

El objetivo de este capítulo es conocer cómo, dónde y cuánto afecta la radiación de partículas a las FPGAs. Se comienza describiendo el fenómeno físico, así como los parámetros que lo definen (energía, flujo y sección de cruce). Posteriormente, se analizan los efectos del impacto de partículas sobre los dispositivos electrónicos en general, y las FPGAs en particular. Entre todos estos efectos se destaca el SEU como el principal desafío en la confiabilidad de los sistemas basados en FPGA.

Los efectos de la radiación son dependientes del entorno en el que trabaja el sistema, diferenciando entre entornos espaciales y terrestres. En el espacio es necesario tener en cuenta el TID, que va a limitar el tiempo de vida útil del dispositivo. También ha de ser tenido en cuenta el efecto potencialmente destructivo SEL, provocado por los rayos cósmicos (GCR). Los datos de TID como de GCR pueden ser obtenidos del software SPENVIS para las diferentes órbitas espaciales.

La principal fuente de SEU en el espacio es el protón procedente de eventos solares. Las FPGAs de tipo SRAM COTS tienen un promedio cercano a 1SEU/día (4.4SEU/semana Kintex7 en órbita L2), con picos en torno a 1SEU/min en las condiciones más desfavorables de tormenta solar (3.6SEU/5min Kintex7 en órbita L2). Los datos de flujo de protones se pueden obtener por medio del software SPENVIS y la sección de cruce se mide mediante tests de protones o de iones.

El SEU en entornos terrestres tiene en el neutrón atmosférico su principal causante. Los valores promedio son en torno a 1SEU/200 años. En aplicaciones de aviónica la tasa de fallo es en torno a 500 veces superior (2SEU/año). Los valores de flujo de neutrones están tabulados en el estándar JEDEC-JESD98. La sección de cruce se mide por inyección de neutrones o de protones. Adicionalmente, también es posible el test en tiempo real.

Los ingenieros disponen de otras tecnologías de FPGAs como la FLASH, la antifuse o las SRAM endurecidas para evitar en gran medida la ocurrencia de SEUs. Sin embargo, también se puede optar por el uso de FPGAs comerciales, que ofrecen múltiples ventajas en términos de rendimiento y capacidades de diseño.

El uso de FPGAs COTS requiere tener en cuenta la posibilidad de ocurrencia de SEUs. Cabe destacar que el objetivo de esta tesis es la medición del impacto de un SEU en el funcionamiento global de diseños implementados en FPGAs. De este modo, si la tasa de fallos medida fuera superior a los requerimientos, sería

necesario aplicar técnicas de tolerancia a fallos, que se explican en el próximo capítulo.

Capítulo 4

Mecanismos de tolerancia a fallos en FPGAs

En este capítulo se presentan los mecanismos de los que disponen los desarrolladores de sistemas basados en FPGAs para orientar sus diseños a confiabilidad. Se particularizan los conceptos presentados en el capítulo 2 a las FPGAs (capítulo 1) para mitigar los efectos de la radiación (capítulo 3). La totalidad de las técnicas desglosadas en el presente capítulo se corresponden con el mecanismo de tolerancia a fallos, según la clasificación presentada en la subsección 2.1.6.

La relevancia de este capítulo reside en varios aspectos. Por un lado, el método propuesto en esta tesis sirve para estimar la tasa de fallo de diseños, y así poder justificar que es suficientemente baja, lo que indica que la mayoría de circuitos a testear van a estar orientados a confiabilidad. Es decir, van a implementar los diferentes métodos que se detallan en el presente capítulo. En este punto, los esquemas TMR (Triple Modular Redundancy) y similares van a ser las aplicaciones más relevantes.

Otra de las técnicas presentadas en este capítulo, el BIST (Built-In Self Test), va a tener una gran relevancia a lo largo del documento, ya que una de las partes del diseño propuesto en esta tesis está implementado mediante un esquema de este tipo. También se va a detallar el mecanismo conocido como scrubbing, que es uno de los mecanismos de tolerancia a fallos más fácilmente implementable en FPGAs, ya que los dispositivos actuales contienen recursos específicos dedicados a la implementación eficiente de este mecanismo.

El capítulo está organizado de la siguiente manera: En primer lugar, se hace

una reseña sobre la aplicabilidad de mecanismos de confiabilidad generales a las FPGAs. Posteriormente, se detallan los mecanismos de tolerancia a fallos basados en esquemas redundantes. Se continúa con el scrubbing y finalmente se analizan otras técnicas de tolerancia a fallos, dando especial relevancia al BIST.

4.1 Aplicabilidad de mecanismos de confiabilidad a las FPGAs

El objetivo de esta sección es la discusión de la aplicabilidad de los diferentes mecanismos de confiabilidad a las FPGAs. Según se detalla en la subsección 2.1.6 del capítulo 2, los mecanismos de confiabilidad se agrupan en 4 bloques: prevención, tolerancia, eliminación y previsión de fallos. A continuación se realiza un análisis de la aplicación de estos 4 bloques de técnicas a los sistemas basados en FPGA. Este capítulo aborda principalmente los mecanismos de tolerancia. Sin embargo, se considera preciso relacionar los otros bloques de mecanismos con conceptos que ya han sido explicados previamente.

Los mecanismos de prevención se basan en una construcción robusta del sistema, evitando así que éste pueda ser dañado por elementos externos. Esto requiere la aplicación de principios de ingeniería de sistemas, como la selección de los componentes y materiales adecuados para resistir los eventos externos que puedan dañar el sistema como temperatura, humedad, polvo, vibraciones o radiación. En cuanto a evitar los errores provocados por la radiación externa en FPGAs mediante construcción robusta se destacan dos posibilidades:

- Incluir un **apantallamiento metálico** para mitigar el efecto del TID (subsección 3.1.2.1).
- Utilizar **dispositivos robustos a la radiación**, caso de las FPGAs de tecnología antifuse, flash o rad-hard SRAM. (sección 3.3).

El concepto de eliminación de fallos consiste en aplicar técnicas de mantenimiento correctivo. Es decir, sustituir los componentes defectuosos o averiados. Se requiere que el tiempo medio de reparación sea suficientemente bajo para garantizar que la disponibilidad del servicio no caiga por debajo de lo aceptable. Conceptos relacionados con gestión de stocks, gestión de operarios y planes de mantenimiento quedan fuera del alcance de este trabajo.

Los mecanismos de tolerancia a fallos basan su funcionamiento en el mantenimiento del servicio a pesar de haber errores en el sistema. Existen dos enfoques para conseguir esto: enmascaramiento de errores y detección-recuperación. El

enmascaramiento consiste en proporcionar una salida válida a pesar de haber errores en el sistema. La detección y recuperación se basa en que el sistema pueda automáticamente detectar errores y tomar inmediatamente acciones para solventarlos. Cabe destacar que ambas opciones pueden utilizarse de manera combinada en un diseño, donde los mecanismos de detección y recuperación evitan la acumulación de errores, y los sistemas redundantes se encargan de proporcionar salidas válidas en todo momento.

Las FPGAs son dispositivos muy adecuados para implementar mecanismos de tolerancia a fallos. La lógica programable confiere a los desarrolladores flexibilidad para implementar cualquier tipo de submódulos que realicen funcionalidades orientadas a confiabilidad. Además, las familias de FPGAs tienen dispositivos de diferentes tamaños, pudiendo elegirse uno más potente en caso de ser necesario más espacio para la implementación de este tipo de mecanismos.

Finalmente, el objetivo de la previsión de fallos es el de conocer con qué frecuencia y qué severidad va a fallar el sistema. Este bloque de mecanismos se abordará en el capítulo 5. Aquí se estudiarán principalmente los sistemas de emulación de SEUs, ya que están directamente relacionados con la solución que se propone en esta tesis. En la subsección 3.3.2 del capítulo 3 se detalla la metodología de caracterización del impacto de la radiación en dispositivos FPGAs, uno de los mecanismos de previsión más relevantes relacionados con este trabajo.

4.2 TMR (Triple Modular Redundancy)

Uno de los mecanismos más utilizados para lograr sistemas basados en FPGA tolerantes a fallos es la redundancia hardware. La configuración TMR (Triple Modular Redundancy) es un esquema basado en la implementación de tres copias del diseño funcionando de forma concurrente y sincronizada. Esta técnica cobra especial relevancia en FPGAs, ya que el triplicado de módulos puede llevarse a cabo dentro del mismo chip, sin necesidad de hardware adicional. La figura 4.1 presenta un esquema de implementación TMR para FPGAs.

En los sistemas basados en redundancia hardware, la salida se decide mediante el proceso de votado por mayoría. Las salidas de cada una de las réplicas llegan a un elemento llamado votador, donde son comparadas y la más común de ellas es trasladada a la salida del circuito. De esta forma, en caso de haber fallos en una de las réplicas, los valores de las otras dos coinciden. Este valor coincidente es trasladado a la salida del circuito, proporcionando así la salida correcta a los usuarios del sistema, garantizando la continuidad del servicio. Por esto, la configuración TMR se clasifica como un mecanismo de enmascaramiento de errores.

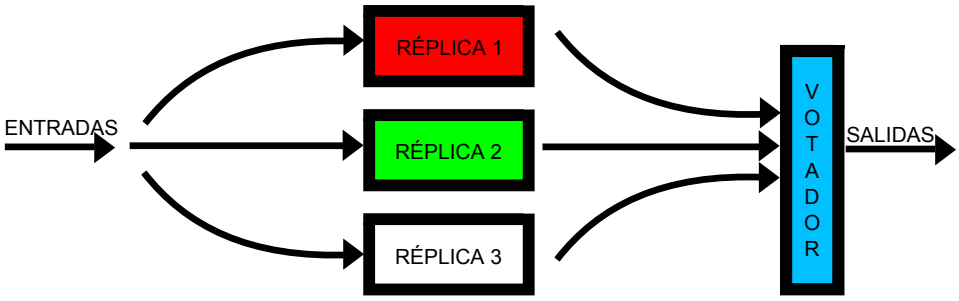


Figura 4.1: Esquema general de una implementación TMR

El error sigue latente en el sistema, pero la redundancia lo hace invisible de cara al exterior.

El esquema TMR es capaz de enmascarar cualquier tipo de error que se pueda producir en una de las réplicas. Este mecanismo es eficaz enmascarando SEUs en la memoria de configuración. Pero su eficacia no se limita a esto, ya que también se enmascaran muchos otros tipos de errores. Se enmascaran también SEUs en elementos secuenciales (flip-flops, registros, BRAMs) a nivel de circuito, otros tipos de SEEs como SETs y SELs y fallos a nivel hardware que afecten solamente a una única réplica.

La configuración TMR también puede ser implantada mediante dispositivos múltiples o mediante PCBs múltiples cuando las normativas de confiabilidad y safety lo requieran (ver sección 2.2.4.2). Sin embargo, esto no es una característica específica de las FPGAs, ya que puede aplicarse a cualquier tipo de circuito electrónico. La aplicación de esta medida supone un incremento sustancial en los costes (económicos, potencia, espacio, etc) y se puede implementar en los casos en los que sea requerido por normativa, o cuando sea la única opción para mantener una tasa de fallos baja.

4.2.1 Causas de fallo en implementaciones TMR en FPGAs

En una FPGA con implementación TMR existen cuatro posibilidades para que los errores se propaguen a las salidas del sistema y se genere un fallo en el servicio:

- Acumulación de errores
- Errores entre dominios (cross-domain errors)
- Errores en elementos comunes

- Errores de sincronización

El efecto conocido como **acumulación de errores** sucede cuando dos errores independientes afectan a dos módulos diferentes (cada error afectaría a una réplica distinta). Es extremadamente improbable que el sistema se vea afectado por dos errores de forma simultánea en un instante de tiempo concreto, y que además estos dos errores afecten a módulos diferentes. La acumulación de errores tiene lugar en casos en los que la implementación de TMR no considera ningún mecanismo para revertir los errores. De esta forma, cuando un error afecta a una de las réplicas, las otras dos proporcionan la salida correcta mientras enmascaran dicho error. Ese funcionamiento correcto se mantiene mientras ningún error afecte a las dos réplicas operativas. En el momento que un segundo error se acumula en una de éstas, aparece un fallo en el servicio. Para evitar que esto ocurra, es necesario aplicar medidas para corregir los errores tan pronto como sea posible. El mecanismo más común para recuperar SEUs en la memoria de configuración de FPGAs es el scrubbing (sección 4.3).

Los **errores entre dominios**, o también CDE (Cross-Domain Errors), suponen otra de las mecánicas de fallos que afectan a los diseños con implementación TMRs. Éstos suceden cuando un único error provoca un funcionamiento erróneo en dos de las réplicas. Este efecto se detalla en profundidad en [13], donde se asegura que son la causa principal de fallos en este tipo de esquemas redundantes. Los CDEs se deben principalmente a SEUs que afectan a los recursos de rutado. Mientras que resulta bastante sencillo aislar los recursos lógicos de los diferentes dominios mediante restricciones en el emplazamiento (pblocks), esto no es así para el rutado. Es bastante difícil impedir que señales de diferentes dominios puedan ser encaminadas por matrices de interconexión contiguas. Incluso pudiera ocurrir que una matriz de interconexión concreta encamine señales de diferentes dominios a través de sí. Los CDEs pueden ser producidos por SBUs, cuando el error afecta a un bit compartido por ambos dominios, o por MCUs, cuando un error múltiple afecta a bits adyacentes pertenecientes a diferentes dominios.

La figura 4.2 representa un ejemplo de CDE. Aquí, dos señales pertenecientes a dominios TMR diferentes (verde y roja) son rutadas a través de dos matrices de interconexión contiguas. En caso de que un SEU afecte a una de ellas (rayo amarillo), una de estas señales puede ser rutada hacia la matriz contigua (rosa-negro). En esta matriz contigua puede colisionar con una señal perteneciente a otro dominio TMR. Cuando esto sucede, un único SEU provoca errores en dos dominios TMR diferentes, lo que se traduce en un fallo del sistema.

La tercera mecánica de fallo en FPGAs con implementación TMR se corresponde con los SEUs que afectan a **elementos comunes**. Estos recursos tienen incidencia en las tres réplicas, por lo que causan un fallo en el servicio cuando

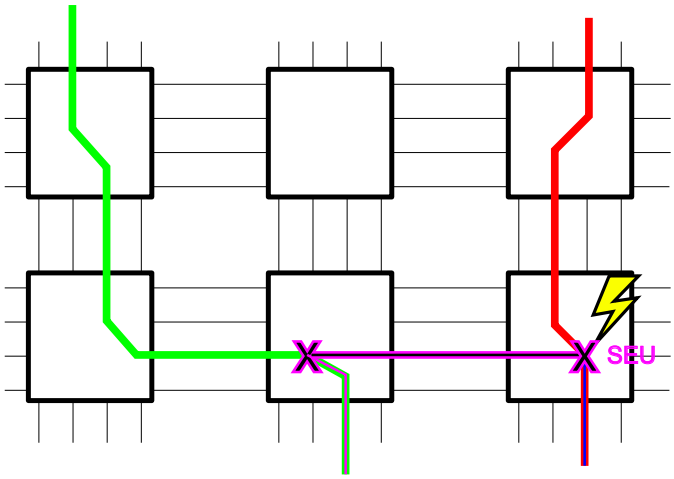


Figura 4.2: Errores entre dominios [13]

son afectados por SEUs. Estos elementos son alimentación, reloj, DCMs, pines de entrada, pines de salida y el votador. También forman parte de esta categoría de fallo los SEUs que afecten al rutado de las señales de entrada y salida desde los pines hasta que estas se bifurcan en tres para atacar las diferentes réplicas.

Por último, los esquemas TMR también pueden fallar por **errores de sincronización** entre las réplicas. Para el correcto funcionamiento del votado por mayoría, es estrictamente necesario que las tres réplicas trabajen perfectamente sincronizadas en el mismo ciclo de reloj. En [14] se explica que los módulos pueden desincronizarse debido a las diferencias en el rutado de señales asíncronas. Este fenómeno se conoce como incertidumbre de muestreo asíncrono (asynchronous sampling uncertainity), y puede hacer que una señal asíncrona se registre en un ciclo de reloj concreto para alguna de las réplicas y en el ciclo de reloj siguiente para el resto. El fenómeno se ilustra en la figura 4.3. Esto puede afectar a las señales de entrada de los pines y a señales que pasen de un dominio de reloj a otro. Si uno de los módulos se desincroniza, los otros dos proporcionan el servicio de manera correcta. Sin embargo, es necesario volverlo a sincronizar para evitar que los errores puedan acumularse. Para ello pueden aplicarse técnicas de sincronización, que se detallan en la sección 4.4.2. En [14] se presenta un esquema para reducir la probabilidad de desincronización por incertidumbre de muestreo asíncrono añadiendo lógica capaz de detectar este tipo de anomalías.

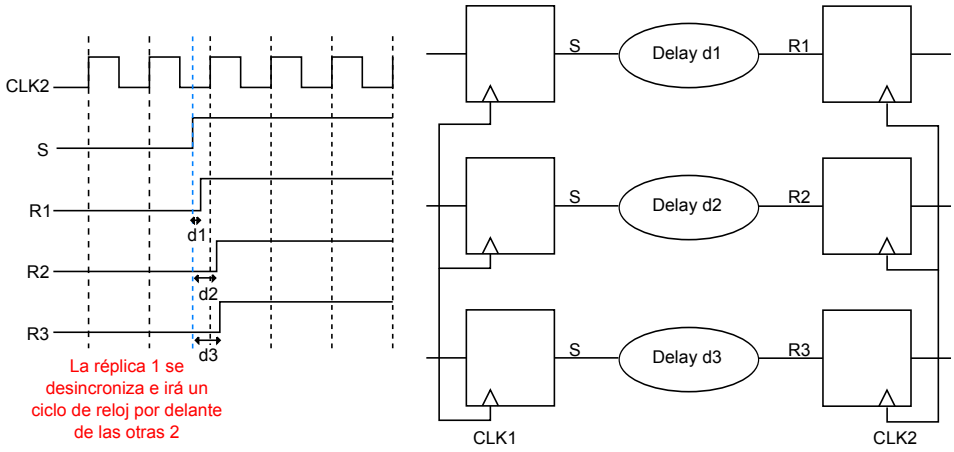


Figura 4.3: Errores de sincrionización en implementaciones TMR con varios dominios de reloj [14]

4.2.2 Implementaciones, granularidades y TMR parcial

Uno de los parámetros que definen una implementación TMR en una FPGA es la **granularidad**; es decir, el nivel de reducción de las réplicas TMR. La implementación de TMR más directa es la conocida como TMR de grano grueso (coarse-grained TMR). Esto se consigue triplicando completamente el diseño. Para ello es necesario bifurcar en tres cada entrada y llevarlas a cada réplica, hacer el votado por mayoría de las salidas y llevar la correcta a los pines. El TMR de grano grueso permite implementar cada módulo en una parcela separada de la FPGA, haciendo mínima la frontera entre dominios y minimizando la posibilidad de errores entre dominios. En esta configuración basta con un solo votador, por lo que los errores en elementos compartidos son también mínimos.

La antítesis de esta implementación es la conocida como TMR de grano fino (fine-grained TMR). Esto consiste en aplicar una implementación TMR a los elementos lógicos más pequeños posibles, que en el caso de una FPGA son las LUTs, los flip-flops y los multiplexores. Esto exige la colocación de un votador por cada elemento triplicado. El votador ha de ser robusto necesariamente, ya que de lo contrario la cantidad de elementos comunes introducidos comprometería dramáticamente la efectividad del mecanismo de tolerancia a fallos aplicado. Para ello se plantea un votador intermedio con salida triplicada en la figura 4.4, que es tolerante a fallos. El votador final es el que va a la salida del circuito, es un elemento común a los tres dominios TMR y por lo tanto, no tolerante a fallos.

Otro esquema de votador seguro se presenta en [82], en donde se plantea añadir señales adicionales de entrada.

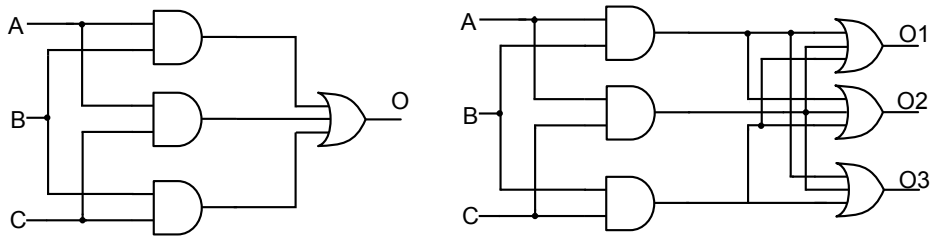


Figura 4.4: Tipos de votadores en TMR. Izda: votador final, Dcha: votador intermedio

El TMR de grano fino protege de manera prácticamente completa frente a la acumulación de errores, ya que es extremadamente improbable que dos errores independientes afecten a réplicas diferentes del mismo dominio TMR al ser los dominios TMR tan pequeños. Sin embargo, este sistema introduce una penalización en frecuencia muy alta. Tampoco es despreciable el aumento del consumo de recursos y de potencia debido a la adición de los votadores. Los errores entre dominios también aumentan debido al incremento de fronteras entre dominios TMR.

Existe una herramienta de Xilinx para las familias Virtex4QV y Virtex5QV que implementa el TMR de grano fino de manera automática [83]. Esto se conoce como XTMR (Xilinx TMR). Aquí se indica que la frecuencia difícilmente puede alcanzar los 100MHz. Esta herramienta no está disponible para las familias de FPGA más modernas.

Como punto intermedio entre estas dos implementaciones TMR se encuentra el conocido como TMR de grano medio (médium-grain TMR). Esto consiste en dividir el diseño total en submódulos y aplicar el TMR a cada submódulo. Esto aumenta considerablemente la protección contra acumulación de SEUs, y tanto el aumento de fronteras entre dominios como la penalización de frecuencia por la inserción de votadores son limitadas. Los votadores empleados entre dominios TMR son votadores intermedios (al igual que en TMR de grano fino). En los casos que exista una separación de dominio de reloj se puede plantear la inserción de votadores finales para evitar la aparición de errores de desincronización. Sin embargo, los votadores finales y el rutado de sus señales son puntos de error comunes en caso de ser afectados por SEUs.

En la figura 4.5 se representan implementaciones TMR de diferentes granularidades para un diseño original compuesto de 6 submódulos. La primera imagen

es el diseño original sin triplicar. La segunda es una implementación de grano grueso, la tercera es una implementación de grano medio y la última es de grano fino. Cada dominio TMR se ha representado con un color (rojo, verde y blanco), y los votadores en azul. En la figura 4.6 se presenta el emplazamiento de estas implementaciones en la FPGA con el objeto de representar las fronteras entre dominios. Esto se ha representado con una línea de color violeta. Aquí se aprecia que para la implementación de grano grueso la frontera es mínima. Es algo mayor en la implementación de grano medio, pero aumentar en exceso. Para la implementación de grano fino no se ha pintado la linea violeta entre dominios para no complicar aún más el dibujo, pero se aprecia que hay mucha frontera entre dominios. Por esto hay muchos más errores de tipo CDE en TMR de grano fino que en grano grueso.

En [84] se presenta un procedimiento de emplazamiento enfocado a la reducción de errores entre dominios TMR. Aquí se asegura que la principal causa de fallos en implementaciones TMR está relacionada con los CDEs, por lo que la reducción de las fronteras entre dominios es una estrategia adecuada para la mitigación de errores en este tipo de diseños.

Por último cabe destacar que algunos diseños utilizan implementaciones de TMR parcial, que consiste en triplicar únicamente algunas partes del diseño. De esta forma se consigue robustecer únicamente los elementos más críticos, pero no es necesario triplicar completamente el diseño. Esto permite la obtención de un buen grado de tolerancia a fallos sin tener un coste adicional tan elevado. Ejemplos de aplicación de esta técnica son [85–88].

4.3 Scrubbing

El scrubbing es una técnica de tolerancia a fallos muy utilizada en FPGAs, ya que las familias modernas incorporan hardware específico para implementarlo de forma sencilla. Esta técnica no enmascara errores, por lo que se clasifica como mecanismo de detección y recuperación. Su rango de alcance se limita a la memoria de configuración. Es decir, mitiga SEUs provocados por radiación en la memoria de configuración. Cabe destacar que la aplicación de esta técnica no tiene ningún efecto sobre posibles SEUs en elementos secuenciales a nivel de circuito como BRAM, Flip-flops o DSPs. Tampoco sirve para detectar errores permanentes en el hardware.

El scrubbing consiste en el refresco continuo de la memoria de configuración [89]. De esta forma se hace volver a su estado original a los bits que hayan podido

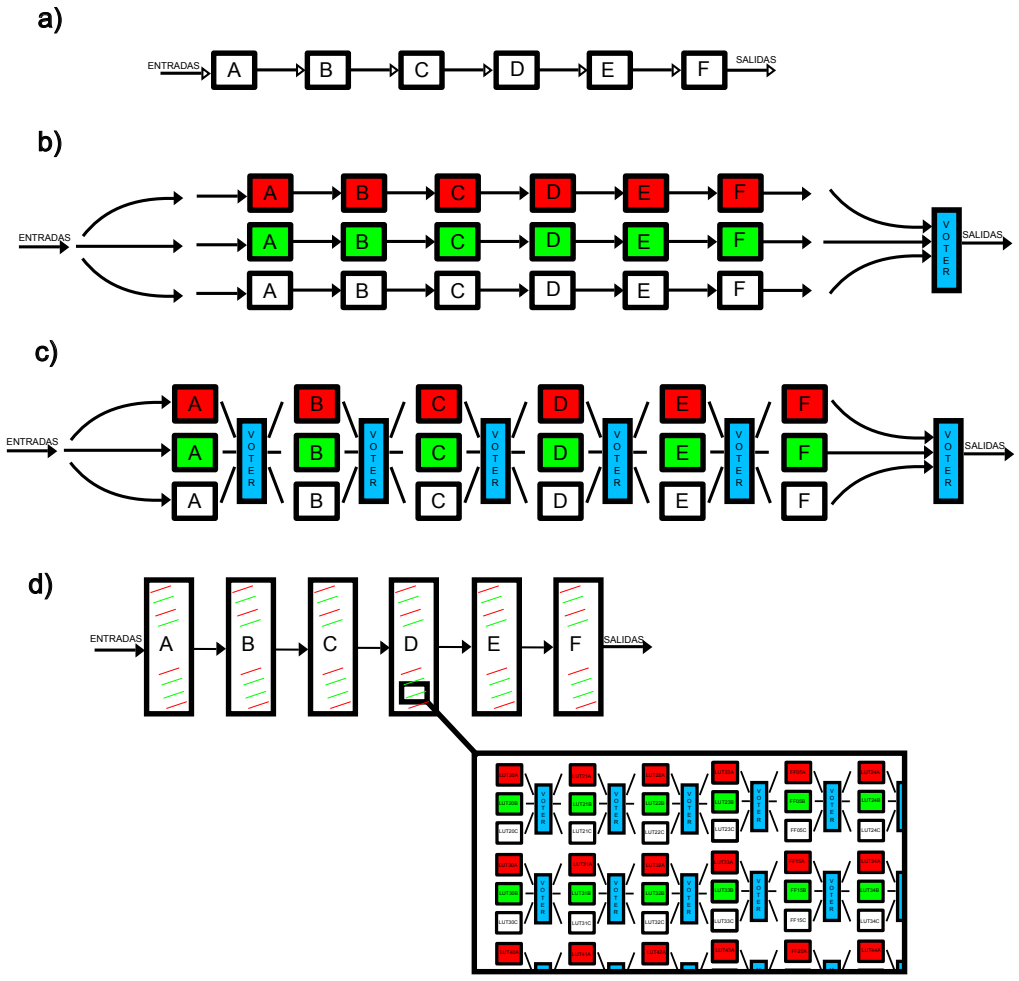


Figura 4.5: Implementaciones TMR de diferentes granularidades. a) implementación simple b) TMR de grano grueso, c) TMR de grano medio, d) TMR de grano fino

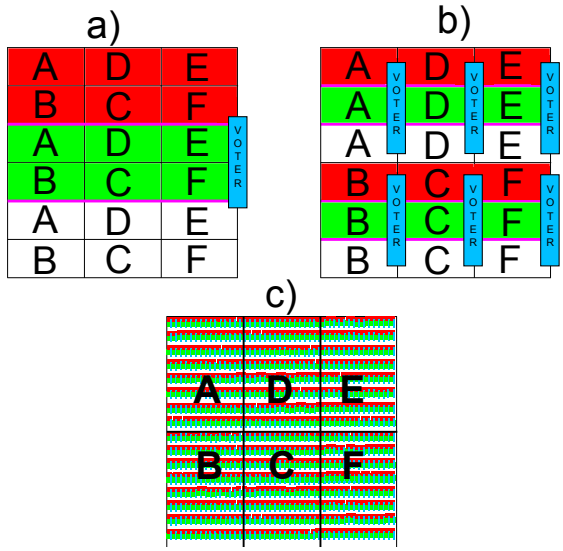


Figura 4.6: Emplazamiento de los dominios TMR para diferentes granularidades. a) TMR de grano grueso, b) TMR de grano medio, c) TMR de grano fino

ser afectados por SEUs. Hay dos enfoques principales para implementar este mecanismo, denominados en [90] como "blind scrubbing" y "readback scrubbing".

4.3.1 Blind Scrubbing

La técnica denominada como "blind scrubbing" se basa en cargar periódicamente el bitstream completo desde una memoria externa. Es una técnica de recuperación únicamente, ya que no se aplica ningún tipo de técnica de detección de errores. Es un mecanismo sencillo de implementar, debido a que únicamente se requiere la carga periódica de un fichero a través de una interfaz de configuración. Según se comenta en [91], el periodo óptimo de refresco es dependiente del tipo de aplicación y de la probabilidad de SEU.

El aspecto más negativo de esta técnica es la necesidad de resetear completamente el estado interno del dispositivo, perdiendo cualquier avance y dato guardado. Esto limita al blind scrubbing a un rango de aplicaciones concreto. Se trata de aplicaciones que realicen repetidamente un cálculo concreto, que dicho cálculo no se extienda demasiado en el tiempo y que los datos se almacenen en algún elemento externo no volátil. En este caso, el blind scrubbing se puede conside-

rar como una acción de mantenimiento programado, que requiere un reseteo del sistema.

Otro aspecto a tener en cuenta es la necesidad de elementos externos para llevar a cabo el blind scrubbing. Es necesaria una memoria en la que almacenar el bitstream, que ha de ser necesariamente robusta. En [91] se propone el uso de una MRAM para esta tarea debido a su robustez. También se requiere de algún tipo de procesador que haga las tareas de scrubbing manager. Es decir, de controlar la interfaz de configuración durante el proceso de carga del bitstream. Este elemento también ha de ser suficientemente robusto. En caso de utilizar un SoC FPGA como el Zynq (Sección 1.5), el blind scrubbing se puede implementar completamente en el ARM sin necesidad de hardware externo adicional.

4.3.2 Readback Scrubbing

Por otro lado, el readback scrubbing consiste en leer frame a frame el contenido de la memoria de configuración y comprobar si el contenido es correcto. Las frames de la memoria de configuración incorporan códigos ECC (Error Correction Codes), que permiten detectar y recuperar errores. Las FPGAs de Xilinx incorporan un módulo hard para realizar estas detecciones y correcciones de forma eficiente.

El readback scrubbing se realiza a través del ICAP. Podría realizarse desde algún interfaz externo, pero requeriría hardware adicional y no añadiría ninguna ventaja. La detección de errores puede realizarse online, sin interrumpir el funcionamiento del sistema, lo que supone una clara ventaja respecto del blind scrubbing. Tras la lectura de una frame se aplica una máscara para descartar los bits referentes a elementos secuenciales del circuito tales como flip-flops o registros de desplazamiento, debido a que estos elementos cambian de valor durante la ejecución y que invalidan el cálculo del ECC si son tenidos en cuenta.

En [91] se destaca que el tiempo necesario para realizar el readback scrubbing de la totalidad de la FPGA es en torno a 6 veces el tiempo de blind scrubbing. Para cada operación de readback es necesario enviar un bitstream de un frame con la orden de lectura y luego recibir el bitstream con el contenido leído. Hay que tener en consideración que cada bitstream cargado o leído tiene una serie de cabeceras que no son carga útil. Entre tanto, el tiempo de software para la gestión de bitstreams también ha de considerarse como tiempo no útil de transferencia de datos.

El readback scrubbing es un algoritmo que permite tanto detectar como recuperar errores. Sin embargo, la aplicabilidad como método de recuperación es discutible, ya que no es posible determinar qué ha sucedido en el sistema desde que el

error ha aparecido hasta que ha sido subsanado, por lo que es necesario llevar al sistema a un estado conocido. Esto puede hacerse mediante un reseteo o mediante técnicas más compejas basadas en checkpointing y rollback (Subsección 4.4.2). Otra posibilidad es la carga del bitstream competo original. También hay que poner en duda los resultados y salidas del sistema durante el tiempo que ha estado fuera de control.

Es precisamente por esto por lo que el scrubbing complementa perfectamente la arquitectura TMR [92, 93]. Este esquema garantiza mediante redundancia que la salida es correcta en todo momento. Cuando una de las tres réplicas es afectada por un SEU, las otras dos se encargan de proveer la salida correcta. En ese momento, el sistema es consciente de qué módulo ha fallado, pero no qué bit en concreto. Entonces se pone en marcha el sistema de readback scrubbing, que detecta el bit afectado y lo corrige. El paso final es el de sincronizar los tres módulos, para lo que se pueden utilizar los mecanismos detallados en 4.4.2. Esta combinación entre scrubbing y TMR se propone en numerosos estudios como [94–97].

Las últimas tendencias en scrubbing apuntan a la reducción del tiempo medio de detección de error. El esquema básico de scrubbing consistiría en comprobar frames empezando por el principio de la memoria hasta llegar al final. En este caso el tiempo medio de detección es la mitad del tiempo total. Este esquema puede mejorarse haciendo algo tan simple como empezar en el punto de la FPGA donde mayor concentración de recursos haya, que son las zonas de mayor probabilidad de ocurrencia de error.

En [98] se propone reducir el tiempo medio de detección de error haciendo scrubbing únicamente en las frames en las que hay algún bit crítico (bit crítico: ver sección 5.2). En [89, 99, 100] se propone optimizar el emplazamiento para que el diseño ocupe la menor cantidad posible de frames. En [101, 102] se propone combinar el scrubbing con un mecanismo modular de detección de errores. Este mecanismo modular indica cuál es el módulo erróneo, aplicando scrubbing únicamente sobre este módulo. Otra posibilidad basada en un scrubbing no uniforme se propone en [103]. En este caso se realizan operaciones de scrubbing sobre las frames con mayor probabilidad de error con mayor frecuencia.

Por otro lado, en [104] se propone un esquema de scrubbing doble, con un ciclo externo de scrubbing y un ciclo interno. El ciclo interno consiste en aplicar el built-in readback scrubbing contenido en las FPGA modernas. En el momento en el que se produzca un error no recuperable por los códigos SEC-DED (Single Error Correction - Double Error Detection), se pone en marcha un ciclo de scrubbing externo para recuperar el error. La tabla 4.1 muestra una comparativa entre blind scrubbing y readback scrubbing

Tabla 4.1: Comparativa entre blind scrubbing y readback scrubbing

Blind scrubbing	Readback scrubbing
Solo recuperación	Detección y recuperación, aunque la recuperación ha de ser complementada con algún otro mecanismo
Necesita hardware externo	No requiere hardware externo
Offline, requiere el reseteo total del sistema	Online, en paralelo con la ejecución del sistema
Rápido	El escaneo completo es 6 veces más lento que blind scrubbing

4.4 Otros mecanismos de tolerancia a fallos en FPGAs

En esta sección se abordan otras técnicas de tolerancia a fallos. Entre ellas se va a analizar el BIST en detalle, ya que se trata de un aspecto especialmente relevante a lo largo de esta tesis. Además, se analiza la existencia de otras técnicas como DMR, lockstep y sincronización.

4.4.1 BIST (Built-in Self Test)

BIST (Built-in Self Test) es una metodología de testeo de sistemas basada en añadir un subsistema de test al propio sistema. Este subsistema es capaz de tomar el control del sistema, generando una serie de estímulos en las entradas y analizando las salidas para determinar si el estado del sistema es erróneo o no. Esta metodología exige la interrupción del sistema para llevar a cabo el proceso de testeo, por lo que se clasifica como mecanismo de detección de errores "offline". En contraposición con el testeo externo, tiene la ventaja de que la accesibilidad a los diferentes puntos del sistema es muy superior. Además de las entradas y salidas, el subsistema de BIST puede acceder a puntos intermedios, lo que se traduce en una cobertura de test mucho más elevada. La desventaja es la necesidad de construcción de dicho subsistema, que no es necesaria en el caso de testeo externo. Sin embargo, en sistemas basados en FPGA, resulta bastante sencillo implementar este tipo de subsistemas dentro de la lógica programable.

Un subsistema BIST se compone de tres partes: el generador de entradas, el analizador de salidas y la unidad de control [105]. El generador de entradas se encarga de proporcionar una secuencia de estímulos que trasladen al sistema a

estados representativos de su funcionamiento. Esta secuencia va a ser siempre la misma, y se conoce como secuencia "golden". Al ser la FPGA un elemento determinista, la respuesta a la secuencia de entrada golden va a ser siempre la misma, mientras no existan errores en el sistema.

Las secuencias de entrada pueden ser de tres tipos: secuencias almacenadas, secuencias pseudoaleatorias o combinaciones de ambas. Las secuencias almacenadas se implementan en las BRAMs. Son una solución bastante buena en algunos casos; sin embargo, a medida que se pretende llevar a cabo un test más exhaustivo, las necesidades de memoria aumentan exponencialmente, lo que hace su implementación impracticable.

Para la generación de secuencias pseudoaleatorias se utiliza un generador de números aleatorios que es iniciado siempre con la misma semilla. De esta forma, la secuencia generada va a ser siempre la misma. Esta solución permite que el test pueda prolongarse en el tiempo sin necesidad de utilizar recursos de memoria para almacenar patrones. El problema es que en ocasiones los sistemas requieren que las entradas tomen unos valores concretos en algunos momentos para realizar un test suficientemente representativo, lo que imposibilita el empleo de este tipo de patrones. La implementación de generadores de números aleatorios no supone ningún tipo de problema en FPGAs, ya que la cantidad de recursos requerida es pequeña.

El analizador de salidas es el encargado de determinar si la secuencia de salida es la correcta o no. Este proceso se lleva a cabo generalmente mediante el cálculo de firmas. Una firma es una función hash, que calcula una palabra cuyo valor es función de todos y cada uno de los bits de la secuencia de salida. De esta forma no es necesario el almacenamiento del stream de salida completo para las entradas golden. Basta con almacenar la firma que el sistema produce en ausencia de errores para que sea comparada con la firma generada en presencia de posibles errores.

Finalmente, el sincronizador se implementa mediante una máquina de estados sencilla. Es el encargado de iniciar y detener el test, sincronizando el generador de entradas con el analizador de salidas. También tiene una interfaz de comunicación con el exterior, que sirve para que algún ente externo pueda iniciar el test y para que el BIST responda a dicho ente con el resultado del mismo.

En FPGAs el BIST tiene dos líneas de aplicación principales, BIST independiente de la aplicación y BIST específico [106]. El objetivo del BIST independiente es hacer un chequeo del dispositivo en busca de defectos en el hardware [107]. Para ello se interrumpe el funcionamiento del sistema y se reconfigura el dispositivo con un BIST independiente. No es posible hacer el testeo completo con una

única configuración, por lo que ha de configurarse el dispositivo varias veces. Esta necesidad de reconfiguración hace que no puedan ser detectados SEUs a nivel de memoria de configuración. En [108] se plantea una arquitectura denominada como Roving STARS, que permite llevar a cabo un BIST independiente de manera parcial, sin detener en ningún momento la operación del sistema.

Por otro lado, el BIST específico es aquel que está especialmente diseñado para testear una aplicación en concreto [109, 110]. No requiere reconfiguración del dispositivo, por lo que es capaz de detectar fallos tanto a nivel de circuito como a nivel de CRAM. Sin embargo, sí se requiere la detención de la ejecución del sistema. Tanto las entradas como las salidas externas son desconectadas del sistema. A las entradas se conectan las entradas del generador de estímulos del BIST, y las salidas se conectan al analizador de respuestas del BIST. En [111] se combinan BIST específico y scrubbing.

4.4.2 DMR, lockstep y sincronización

DMR (Dual Modular Redundancy), a veces referido como DWC (Duplication With Comparison), es una posible alternativa a TMR. Pero en este caso haciendo el duplicado del módulo en lugar del triplicado. Ambos procesadores reciben las mismas entradas y han de proporcionar las mismas salidas, de forma que se puede detectar de forma automática la existencia de errores. Aquí, a diferencia de en TMR, no se puede determinar en qué módulo se ha producido el error, por lo que la ejecución del sistema ha de ser necesariamente interrumpida. Por lo tanto, DMR es una técnica de detección de errores, y no de enmascaramiento como TMR. El coste adicional de esta arquitectura es del 100 %, en contraste con el 200 % introducido por TMR [112–114].

Cuando la arquitectura DMR se aplica a procesadores, se conoce como lockstep [115–117]. Se implementan dos procesadores idénticos corriendo el mismo software y sincronizados en el mismo ciclo de reloj. La particularidad de esta arquitectura es que define la sincronización entre procesadores mediante checkpointing y rollback. Checkpointing consiste en almacenar toda la información de los elementos secuenciales del sistema (BRAMS y flip-flops) en un instante de tiempo concreto, en el cual no haya presencia de errores. Rollback consiste en cargar este estado previo almacenado en el momento que un error es detectado. La combinación entre checkpointing y rollback también puede emplearse para sincronizar sistemas TMR.

4.4.3 Otros mecanismos

Los códigos de detección y corrección de errores son uno de los mecanismos más utilizados en cualquier tipo de sistemas electrónicos para aumentar su robustez. Estos códigos se implementan en FPGAs para proteger la información almacenada en los elementos secuenciales a nivel de circuito implementado tales como flip-flops o BRAMs. Las FPGAs son elementos idóneos para la implementación de este tipo de códigos, ya que basta con colocar unas pocas puertas lógicas junto a estos recursos, algo fácil de hacer en dispositivos programables. Este tipo de códigos han sido y son profundamente estudiados para su aplicación en sistemas electrónicos de todo tipo.

Otra técnica empleada para robustecer los sistemas es la conocida como ABFT (Algorithm Based Fault Tolerance). Es una técnica de tolerancia a fallos que emplea algoritmos específicos de la aplicación. Su aplicabilidad se reduce a aplicaciones muy concretas, generalmente relacionadas con el cálculo matricial. En [118, 119] se plantean algoritmos de tolerancia a fallos específicos para diferentes aplicaciones de cálculo matricial.

4.5 Conclusiones

En este capítulo se han estudiado los diferentes mecanismos de tolerancia a fallos que pueden ser aplicados en sistemas basados en FPGAs. En primer lugar se lleva a cabo una discusión acerca de las diferentes posibilidades para la implementación de mecanismos de prevención de fallos y de eliminación de fallos (la previsión de fallos se deja para el próximo capítulo). Posteriormente se analiza en profundidad el esquema de redundancia triple TMR, para estudiar después el mecanismo conocido como scrubbing. Finalmente, se abordan el BIST y otros mecanismos de tolerancia a fallos.

TMR es una técnica de tolerancia a fallos muy empleada, que consiste en triplicar el sistema original y hacer un votado por mayoría a la salida. Cuando TMR se aplica en una FPGA, existen cuatro mecanismos de fallo: la acumulación de errores, los errores entre dominios (CDE), los errores en elementos comunes y los errores de sincronización. El efecto de la acumulación y los CDE puede modularse mediante la granularidad. A mayor granularidad menos errores por acumulación, pero más CDE. Los errores en elementos comunes van a ser probabilísticamente muy pocos. Sin embargo, resulta difícil reducirlos aún más, quedando como prácticamente única alternativa el empleo de configuraciones multi-device o multi-PCB. Finalmente, es necesario tener precaución con las entradas asíncronas en

dominios TMR, ya que pueden generar problemas de meta-estabilidad. El TMR es especialmente relevante durante la tesis, ya que la solución propuesta ha sido empleada para testear configuraciones de este tipo.

La configuración TMR permite enmascarar errores en el sistema, ya que éste continúa proporcionando la salida correcta en presencia de errores. Sin embargo, el punto débil de este mecanismo es recuperar los módulos erróneos, para que pueda preservarse la posibilidad de enmascarar nuevos errores. Para ello es preciso que TMR sea combinado con alguna técnica de recuperación. La técnica más idónea para complementar el TMR es el scrubbing, que consiste en el refresco de la memoria de configuración. Durante el capítulo se han analizado las dos principales maneras de implementar el scrubbing, que son blind scrubbing y readback scrubbing. Las ventajas de blind scrubbing son la sencillez y la velocidad de refresco, mientras que readback scrubbing permite la detección online de errores, así como la recuperación de errores sin perder la información de estado del sistema.

Finalmente, se presenta el BIST, que si bien es una técnica menos utilizada que TMR junto con scrubbing, es de gran relevancia para el desarrollo de esta tesis, ya que el subsistema de verificación propuesto sigue este esquema.

Capítulo 5

Emulación de SEUs en FPGA

Este capítulo se centra en la evaluación de la tasa de fallo debida SEUs en circuitos implementados en FPGA. En el capítulo 2 se han introducido los conceptos básicos de confiabilidad, y se han detallado los diferentes tipos de mecanismos que se pueden aplicar para el desarrollo de sistemas confiables. Dichos mecanismos se han clasificado en 4 grupos: prevención de fallos, eliminación de fallos, tolerancia a fallos y previsión de fallos. En el capítulo anterior se han discutido en profundidad los de tolerancia a fallos, y se ha hecho una reseña sobre los mecanismos de prevención y eliminación.

El punto de partida de este capítulo es la discusión de los mecanismos de previsión de fallos desde un punto de vista general. El objetivo de estos procedimientos es la cuantificación de la tasa de fallo del sistema, para poder justificar así el cumplimiento de los requisitos de confiabilidad. Posteriormente, estos métodos genéricos se particularizan para el caso de SEUs en FPGAs, discutiendo las diferentes posibilidades de evaluación de la tasa de fallo en este tipo de dispositivos. En este punto, se destaca la **emulación de SEUs** como una estrategia sencilla, económica y eficaz para llevar a cabo la evaluación de la tasa de fallo.

Las siguientes secciones desglosan las diferentes etapas de este proceso: inyección de errores, verificación y recuperación. Posteriormente, se presentan los esquemas de emulación de SEU existentes más relevantes. Se analizan en detalle las soluciones que se han planteado para la implementación de las diferentes etapas de la emulación, para finalmente extraer conclusiones. Estas conclusiones son el

punto del que se va a partir para el desarrollo del emulador de SEU planteado en esta tesis.

5.1 Evaluación de la tasa de fallos provocados por SEUs en FPGAs

La previsión de fallos (fault forecasting) es el cuarto grupo de los mecanismos para el desarrollo de sistemas orientados a confiabilidad, de acuerdo con la clasificación presentada en la sección 2.1.6 (prevención, eliminación, tolerancia y previsión). El propósito de los mecanismos de previsión de fallos es el de conocer con qué frecuencia y con qué severidad van a producirse interrupciones de servicio. Para ello, es preciso llevar a cabo una evaluación de la respuesta del sistema en los casos de activación de eventos potencialmente peligrosos. Esto permite la obtención de la tasa de fallo global del sistema, conocida la probabilidad de ocurrencia de cada uno de estos eventos.

Esta evaluación tiene dos aspectos a tener en cuenta, el aspecto cualitativo y el cuantitativo. El primero de ellos se centra en identificar, clasificar y jerarquizar todas las combinaciones posibles de eventos, fallos individuales de componentes y condiciones ambientales. El aspecto cuantitativo de la evaluación tiene como objetivo el cálculo de valores numéricos para los diferentes atributos de la confiabilidad, para poder certificar que se encuentran dentro de los márgenes asumibles. La evaluación ha de estar presente en el desarrollo de cualquier sistema orientado a confiabilidad, ya que es la manera de justificar que la tasa de fallo es razonablemente baja.

Los mecanismos de evaluación de fallos se dividen en tres grupos:

- Análisis
- Modelado
- Test

El objetivo de los mecanismos de análisis es la obtención de la tasa de interrupciones del servicio partiendo de la tasa de fallo individual de los componentes atómicos [54]. Esto se lleva a cabo mediante diagramas de árbol que ilustran las relaciones e interacciones entre los diferentes componentes. Existen dos tipos principales de mecanismos de análisis, que son el FTA (Fault Tree Analysis) [55] y FMEA (Failure Modes and Effects Analysis) [120]. FMEA consiste en un análisis ascendente, elaborando un árbol probabilístico que parte de la probabilidad de

fallo de cada componente atómico, hasta llegar a la tasa global de interrupciones de servicio.

Por otro lado, en el procedimiento FTA se lleva a cabo un análisis descendente. El punto de partida es un modo de fallo concreto, y partiendo de éste se realiza un árbol, descendiendo a diferentes niveles de módulos, submódulos y componentes que están relacionados con dicho modo de fallo. Ambos mecanismos pueden utilizarse de manera combinada, ya que se complementan entre sí, tal y como se muestra en [121, 122]. Debidoa que los métodos de análisis son generales para los sistemas de ingeniería pueden aplicarse a los sistemas basados en FPGA, tal y como se propone en [53, 123].

Los métodos de análisis necesitan los datos de la tasa de fallo de cada uno de los componentes que conforman el sistema. Estos valores pueden obtenerse por dos vías: modelado y test. El modelado consiste en la construcción de un modelo del sistema (usualmente mediante software de ordenador) para que le sean aplicadas las condiciones de estrés que el componente va a experimentar durante su vida útil. En [124, 125] se proponen modelos de circuitos CMOS para simulación, en los que se simula la generación de pares e-h que se corresponde con un evento de SEU concreto.

En cambio, el test se basa en la monitorización del comportamiento del sistema bajo unas condiciones determinadas. Se considera más aproximado a la realidad que el modelado, ya que las medidas se realizan sobre el sistema real, en lugar de obtenerlas desde un modelo generado por ordenador. Sin embargo, el test requiere la existencia del componente físico construido, del que solamente se puede disponer en fases tardías de la etapa de desarrollo. Los tests pueden ser en tiempo real o acelerados.

Mientras que los tests en tiempo real consisten en observar el funcionamiento del sistema en el entorno en el que va a trabajar, los tests acelerados se basan en generar eventos adversos en un laboratorio, lo que incrementa drásticamente la ocurrencia de fallos. De esta forma se reduce notoriamente el tiempo necesario para obtener datos válidos desde el punto de vista estadístico. En la sección 3.3.2 del capítulo relativo a la radiación se detallan diferentes campañas de test de SEUs en FPGAs, tanto en tiempo real como acelerados.

5.2 Evaluación de los fallos debidos a SEUs en FPGAs

En esta sección se aborda la evaluación de la resiliencia a SEUs de circuitos implementados en FPGAs. Como punto de partida se toma la sección 3.3.2, en la que se ha detallado la metodología para la caracterización de la tasa de SEUs de la memoria de configuración de dispositivos FPGA. De estos estudios de caracterización se obtiene la tasa de SEU por bit del dispositivo (τ_{bit}), que viene dado en unidades de FIT/Mb. Multiplicando este valor por el tamaño de la CRAM se obtiene la cantidad de SEUs que va a afectar a la memoria de configuración en 10^9 horas.

La obtención del valor τ ha sido discutido a lo largo del capítulo 3. Para el caso de FPGAs de Xilinx éste se puede obtener directamente del Reliability Report (Reporte de fiabilidad) [15], o indirectamente, partiendo del valor de σ (sección de cruce) especificado en el mismo reporte y los datos de ϕ (flujo de radiación) obtenidos de SPENVIS [63] para misiones aeroespaciales o de JEDEC-JESD [11] para aplicaciones de aviónica y terrestres.

Por otro lado, tal y como se ha comentado en la sección 1.1, todos los bits de la memoria de configuración no tienen un efecto crítico en caso de ser afectados por un SEU. Por lo tanto, para conocer FR (la tasa de fallo del sistema en FIT) es necesario conocer la cantidad de bits críticos que hay en la memoria de configuración. De esta forma, la tasa de fallo de un diseño concreto implementado en una FPGA se calcula mediante la ecuación 5.1. El número de bits críticos, que se va a representar con las siglas BC, es por tanto un parámetro necesario para el cálculo de la tasa de fallo.

$$FR = BC(Mb) * \tau_{bit}(FIT/Mb) \tag{5.1}$$

Esta sección se centra en el análisis de las diferentes estrategias posibles para la estimación de BC. Tal y como se ha comentado en la sección 1.1, la mayoría de los puntos de rutado no son utilizados por los diseños, por lo que no tienen una incidencia real sobre el comportamiento del circuito en caso de ser afectados por SEUs. Por lo tanto, los bits de la CRAM que hacen referencia a estos elementos no son críticos para el funcionamiento del sistema. Los bits críticos son aquellos bits de la CRAM que hacen referencia a los recursos físicos de la FPGA sobre los que se implementa el diseño, y que en caso de ser afectados por SEUs, provocan interrupciones en el servicio.

La aproximación más sencilla, y a su vez la más pesiminsta para la estimación del

BC de un diseño concreto es considerar todos los bits del bitstream como críticos. El cálculo de la tasa de fallo es inmediato, y se tiene la certeza de que el valor de FIT del sistema va a ser siempre inferior al estimado mediante dicho cálculo. Como ejemplo, el dispositivo Zynq7020 tiene un bitstream de 32Mb, siendo τ = 82FIT/Mb para la ciudad de NY, se puede asegurar que la tasa de fallo del sistema va a ser siempre inferior a 2624FIT.

La siguiente posibilidad (también pesimista) es considerar los datos de ocupación del dispositivo proporcionados por la herramienta de CAD para el diseño basado en FPGA. Es evidente que aquellos recursos que no están siendo utilizados no van a ser críticos (salvo excepciones muy concretas). Por tanto un diseño con una ocupación en torno al 70 % para el dispositivo anteriormente mencionado va a tener una tasa de fallo de 1635FIT.

Otra opción es utilizar el fichero "essential bits" que genera Vivado (para las FPGAs de Xilinx). Este fichero es una aproximación de la cantidad de bits críticos de un diseño concreto realizado por el software de Xilinx. Sin embargo, tal y como se menciona en [126] y se ha comprobado durante este trabajo, los resultados obtenidos son en torno a tres veces el número real de BC.

La opción recomendada por Xilinx [15, 127], es hacer uso de un parámetro conocido como DVF (Device Vulnerability Factor), que indica el porcentaje de bits del bitstream que son potencialmente críticos. Este factor toma valores según Xilinx entre el 2 % y el 10 %. De acuerdo con el fabricante, sería un desafío académico encontrar un circuito con una tasa de fallo superior al 10 % de la memoria de configuración. Aunque no se ha podido probar que no sea posible superarlo, se asegura que los diseños reales nunca van a superar dicho valor. Por lo tanto, Xilinx recomienda estimar el DVF multiplicando la ocupación del dispositivo por el valor de DVF máximo posible (10 %).

$$BC = tamaño\ bitstream * DVF = tamaño\ bitstream * ocupaci\tilde{o}n * DVF_{MAX} \tag{5.2}$$

Sin embargo, esta estimación sigue siendo pesimista e inexacta. Un factor DVF entre el 2 % y el 10 % se traduce en posibles variaciones del 400 % en la estimación de la tasa de fallo del sistema. Además, este método no es válido para la evaluación de implementaciones tolerantes a fallos, ya que en este tipo de diseños un aumento en la ocupación se traduce en una tasa menor de BC. En [2] se plantea un procedimiento de análisis estático más sofisticado basado en el estudio de varios ficheros generados por el software de Xilinx. Este método detecta un número de bits críticos mayor que el real. En torno al 63 % de los bits identificados como críticos lo son realmente. Se consigue acotar más el margen de error, pero sigue

siendo relativamente elevado.

Es por esto que se consideran alternativas para la medida del número de bits críticos del sistema. Una de ellas es la emulación de SEUs, que consiste en programar la FPGA con un bitstream que tenga un bit intencionadamente corrompido, de modo que en la memoria de configuración se almacene un bit erróneo, emulando el efecto de un SEU. Después de la programación, se ha de efectuar una verificación para comprobar si el funcionamiento correcto del sistema ha sido alterado o no. En concordancia con el resultado de dicha verificación, el bit intencionadamente corrompido se etiqueta como crítico o no crítico. Posteriormente es necesario devolver al sistema al estado inicial, para proseguir con las siguientes inyecciones de errores. Una vez testeado un número de bits estadísticamente significativo, se obtiene el porcentaje final de bits críticos.

En la figura 5.1 se presenta el esquema general de una plataforma de emulación de SEUs en FPGAs. El circuito cuyo número de bits críticos se desea evaluar se ha denominado UUT (Unit Under Test), tal y como se hace en [4]. Otros trabajos le dan también nombres similares DUT, FUT, CUT, MUT (Design/Function/Circuit/Module Under Test). Este no forma parte del subsistema de inyección, pero hay que tener presente que va a ser el destinatario de las inyecciones de errores que se puedan realizar. El UUT va a implementarse siempre en el interior de la FPGA, mientras existen diversas posibilidades para la implementación del resto de elementos. Los elementos relativos al subsistema de inyección se representan en color rojo, en verde los relativos a la verificación y el amarillo el UUT.

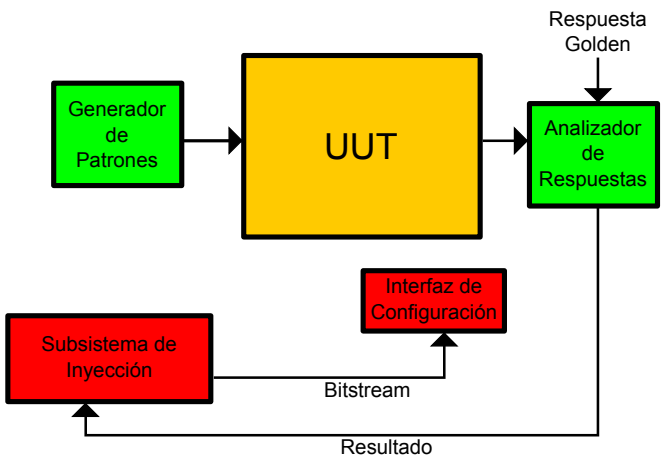


Figura 5.1: Esquema general de una plataforma de emulación de SEUs en FPGAs

Cabe destacar que la emulación solamente es adecuada para evaluar el número de bits críticos en la memoria de configuración. Los bits relativos al contenido de flip-flops y BRAMs no pueden ser analizados mediante esta mecánica, ya que en caso de ser sobrescritos durante la verificación antes de ser leídos no generan ningún tipo de alteración en el sistema. Estos bits siempre son críticos, ya que son parte del sistema. La cuantificación del peso de estos elementos es sencilla, ya que viene indicada en los reportes generados por los programas de CAD. El número de bits críticos relativos a flip-flops no son de gran relevancia en el BC global, sin embargo, el número relativo a BRAM sí que puede ser alto, y ha de ser tenido en cuenta. En [128] se propone una metodología para la inyección de errores en los flip-flops de una FPGA.

Las siguientes secciones de este capítulo profundizan en cada una de las etapas de la emulación de SEU: inyección, verificación y restablecimiento del estado inicial. En cada una de estas secciones se valoran las diferentes alternativas propuestas por otros métodos de inyección de errores.

5.3 Inyección de errores

El proceso de inyección de errores consiste en la modificación del contenido de un bit de la memoria de configuración de la FPGA, emulando así el efecto de un SEU. Para llevar dicho proceso a cabo, es necesaria la implementación de una serie de elementos, cuyo conjunto se conoce como "subsistema de inyección". Este subsistema se compone de los siguientes elementos:

- Software de inyección
- Bitstream
- Interfaz de configuración

El subsistema de inyección se implementa en forma de software, cuyo cometido principal es el de alterar convenientemente el bitstream y cargarlo a través de la interfaz de configuración. Adicionalmente, este programa se comporta como maestro del sistema global de emulación, ya que se encarga de iniciar el proceso de verificación una vez el error haya sido inyectado. Además, recoge el resultado del proceso de verificación y genera estadísticas sobre errores inyectados y fallos del sistema detectados para la computación final del número de bits críticos. Este software se encarga también de la finalización del test, así como de la presentación de los resultados finales. En algunos casos, la memoria interna del procesador que corre este software se emplea para el almacenaje del bitstream.

El bitstream es el fichero que hay que cargar en la memoria de configuración de la FPGA para que ésta realice una función concreta. Tal y como se ha explicado en la sección 1.6, hay 3 tipos de bitstream: el completo, el parcial y el de 1 frame. El completo es el que contiene la totalidad del diseño, y obligatoriamente ha de cargarse desde una interfaz externa. Las FPGAs modernas pueden reconfigurar algunos submódulos mientras el resto del dispositivo trabaja de forma normal, lo que se conoce como RPD (Reconfiguración Parcial Dinámica) (DPR en inglés). Los bitstreams parciales requieren un tiempo menor para su carga, pero pueden generar problemas de meta-estabilidad si las fronteras con las zonas no reprogramandas no están adecuadamente delimitadas. El bitstream de 1 frame es el bitstream más pequeño que se puede generar, siendo su tiempo de carga el más rápido posible.

La interfaz de configuración es el puerto desde el que se carga el bitstream en el dispositivo. Las FPGAs de Xilinx tienen tres: ICAP, SelectMAP y JTAG. ICAP (Internal Configuration Access Port) necesita la implementación de lógica adicional en el interior de la FPGA. SelectMAP es una interfaz externa paralelo, puede cargar el bitstream a gran velocidad, pero requiere de hardware específico en el PCB para este cometido. Finalmente el JTAG permite cargar el bitstream desde un PC externo, pero es muy lento. Ver sección 1.6 para más información.

Los dispositivos de Altera, el otro fabricante principal de FPGAs de tecnología SRAM, tienen opciones de configuración similares a las de Xilinx. Contienen una interfaz de configuración externa, que puede configurarse en modo FPP (Fast Passive Parallel), equivalente a SelectMAP, en modo JTAG o en modo serie. Para la reconfiguración parcial interna existe un bloque denominado PRCB (Partial Reconfiguration Control Block), equivalente al ICAP.

Dependiendo de la interfaz utilizada, existen dos opciones principales para la implementación del subsistema de inyección: inyección interna e inyección externa. Las diferentes metodologías de inyección interna se basan en la utilización del ICAP para la carga de los bitstreams modificados. Por otro lado, los métodos de inyección externa utilizan las interfaces externas (SelectMAP y JTAG para dispositivos de Xilinx) para este cometido.

5.3.1 Métodos de inyección interna de SEUs

Los métodos de inyección interna de SEUs son aquellos que utilizan interfaces de configuración internas para la carga del bitstream modificado. La utilización de estos interfaces (ICAP en Xilinx y PRCB en Altera) requiere de la implementación de lógica adicional en la propia FPGA. Esta lógica va a ser la encargada del envío de los bitstreams y del control de la propia interfaz.

La ventaja principal de este tipo de métodos es la alta velocidad en la inyección de errores debido a que el bitstream empleado es de 1 frame, que es el más reducido posible (contiene en torno a 3000 bits dependiendo de la familia de FPGA), y su tiempo de carga es muy bajo. Además, no se requiere de ninguna memoria para el almacenaje del bitstream, ya que éste se genera mediante readback y su tamaño es muy reducido. Todos los componentes se implementan en la propia FPGA que se está testeando, por lo que no se requiere ningún tipo de modificación a nivel de PCB ni ningún equipamiento externo adicional.

Una de las principales desventajas de este enfoque es la intrusividad, ya que la adición de elementos dentro de la FPGA produce un aumento en la tasa de fallos medida. Además, la compatibilidad de esta metodología con la validación en V (ver sección 2.1.4) es muy cuestionable. Este equipamiento de test ha de ser eliminado a la hora de ser embebido en un entorno de trabajo real, ya que no se permite la existencia de módulos que no tengan ninguna funcionalidad. Pero el hecho de eliminar un submódulo exige una nueva implementación que a su vez necesita ser validada. El modelo en V exige una validación a un nivel de abstracción superior una vez superada la validación a nivel concreto. No se permiten modificaciones de diseño después de la primera validación.

Por otro lado, la lógica relativa a la inyección de errores puede resultar afectada por las inyecciones de errores que se están realizando. En este caso, el sistema no es capaz de recuperarse de los errores introducidos, quedando en una situación de bloqueo. En [6] a este efecto se le da el nombre de "side effects". Por lo tanto, es necesario tomar acciones para tratar de minimizar este efecto en la medida de lo posible. En la sección 5.6 se presentan las propuestas de diferentes esquemas de emulación para reducir el impacto de los side effects.

En algunos casos se propone la implementación del UUT en un pblock para minimizar este efecto. De esta forma, el dominio de test se reduce a dicho pblock, siendo la única zona del dispositivo en la que se van a inyectar errores. Así, se garantiza que el subsistema de inyección no va tomar recursos lógicos (LUTs, MUX, FF, etc) de dicho pblock. Sin embargo, no se puede garantizar que dicho pblock no vaya a rutar señales pertenecientes al subsistema de inyección. Asimismo, el UUT puede rutar sus señales a través de la parte estática, y quedarían fuera del dominio de test. En la figura 5.2 se representa una configuración general para la inyección interna de errores.

- Ventajas:
  - Simple: no requiere modificaciones a nivel de PCB.
  - Muy Rápido

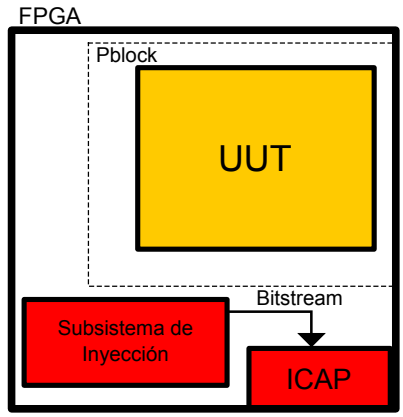


Figura 5.2: Esquema de emulación mediante inyección interna de errores

- Desventajas:
  - Intrusivo: mayor tasa de fallos medida de la real
  - Incompatible con modelo en V
  - Bloqueo por inyecciones en el propio sistema de inyección: Side-effects

5.3.2 Inyección externa

Las metodologías de inyección externa de errores son aquellas que utilizan interfaces externas para la carga del bitstream intencionadamente corrompido. Esto implica que el subsistema de inyección esté ubicado en elementos externos a la FPGA. Aquí se distinguen dos bloques de metodologías de inyección externas. Por un lado, las que utilizan una interfaz de configuración paralelo e implementan el subsistema de inyección en el PCB; y por otro lado, las que utilizan el JTAG y utilizan un PC para este cometido. De esta forma, los inconvenientes relativos a la inyección interna (intrusividad y bloqueo) quedan solventados.

La utilización de un interfaz paralelo, como el SelectMAP de las FPGAs de Xilinx, permite la carga del bitstream a gran velocidad. Esto posibilita que puedan emplearse tanto bitstreams completos como parciales sin que el tiempo de carga sea excesivo. El problema radica en que la gestión de esta interfaz ha de ser controlada por circuitería implementada en el propio PCB. Debido a esto, la emulación de SEUs por interfaces externos paralelo tiene dos vertientes: modificación de PCB y construcción de plataforma dedicada.

La primera de ellas consiste en modificar el PCB original para integrar los componentes relativos al subsistema de inyección. Sin embargo, en una casuística general en la que el sistema ya esté construido, o no se disponga de los recursos necesarios, una implementación de este tipo resulta prácticamente inviable.

Como alternativa, puede plantearse la construcción de una plataforma de test con una configuración de este tipo [7]. En este caso, se realiza una primera implementación de test para ser cargada en dicha plataforma. Una vez obtenida la medida de BC, el diseño se re-implementa para la tarjeta que va a trabajar en el entorno real. Esto plantea una serie de inconvenientes, ya que la FPGA de la plataforma de test va a ser diferente de la FPGA del sistema, lo que requiere volver a ejecutar los algoritmos de place and route. Una nueva implementación va a tener una cantidad diferente de BC, ya que esta medida es muy dependiente de los recursos de rutado empleados. Además, esto puede considerarse incompatible con el modelo de verificación en V, ya que después del test de validación no se permite la generación de nuevos ficheros de diseño.

Por otro lado se encuentran los métodos externos de inyección de errores que utilizan el interfaz JTAG [1, 129]. Este interfaz permite configurar la FPGA desde un PC de forma sencilla, y es este PC el que se utiliza para implementar el subsistema de inyección. Sin embargo, este protocolo de comunicaciones es lento ya que se trata de un protocolo serie, (una única línea de datos). Además, el protocolo introduce tiempos muertos que retrasan más el proceso de configuración. La carga de un bitstream completo requiere de varios segundos. La ventaja es que no se requieren modificaciones de hardware, y el PC ofrece infinitas posibilidades para implementar cualquier tipo de software.

En la figura 5.3 se representa una configuración general para la inyección externa de errores. Tal y como puede apreciarse en la figura, el UUT se implementa en el interior de la FPGA sin ningún tipo de restricción, mientras que el subsistema de inyección no es susceptible de ser afectado por las inyecciones realizadas.

- Ventajas:
  - No intrusivo
  - Compatible con modelo en V
- Desventajas:
  - SelectMAP: Complejidad a nivel de PCB, rigidez
  - JTAG: Muy lento para bitstreams completos

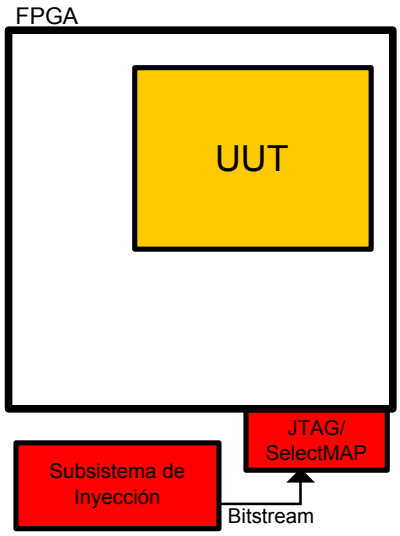


Figura 5.3: Esquema de emulación mediante inyección externa de errores

5.4 Verificación

El objetivo de la etapa de verificación es chequear la funcionalidad del circuito una vez que el error ha sido introducido. Se compone de dos partes: generación de patrones y análisis de respuestas. La primera va a ser la responsable de la generación de los vectores de test que van a ser enviados al circuito. La segunda se centra en comparar las respuestas a los estímulos aplicados con las respuestas obtenidas en ausencia de inyección de errores.

Tanto el generador de patrones como el analizador de respuestas pueden implementarse tanto internamente como externamente a la FPGA. Las figuras 5.4 y 5.5 representan esquemas de verificación interna y externa respectivamente.

5.4.1 Generación de patrones

Los patrones de test son necesarios para verificar el correcto funcionamiento de cualquier sistema. Esto consiste en aplicar unos valores específicos a las entradas para verificar la mayor cantidad de estados y funcionalidades en el menor tiempo posible. Un patrón de test es una secuencia de entradas que es generado de manera idéntica en repetidas ocasiones. La primera vez se ejecuta el test en ausencia de

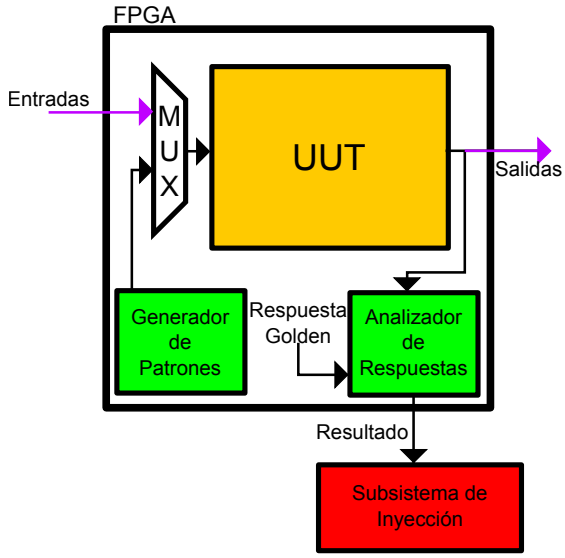


Figura 5.4: Verificación interna

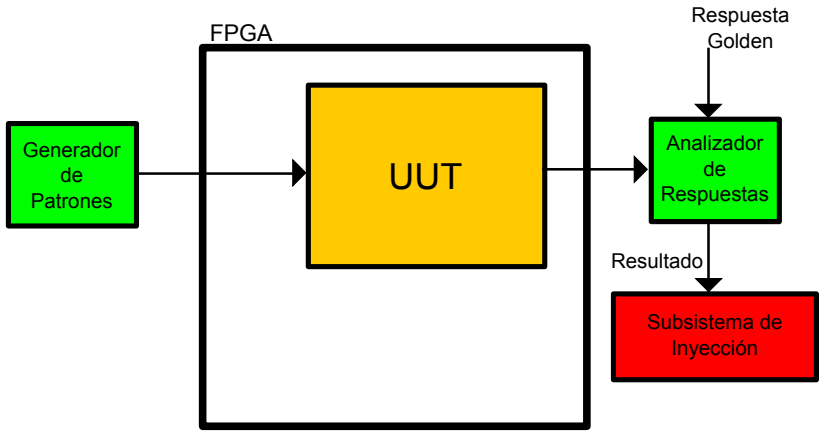


Figura 5.5: Verificiación externa

errores y se almacena el resultado. Este patrón se repite una y otra vez para valorar la criticidad de cada bit de la memoria de configuración.

Para efectuar una verificación exhaustiva, el test ha de cumplir dos características: tiempo y calidad de los patrones de entrada. El tiempo es un factor necesario para permitir que las variaciones en las entradas se propaguen a través del circuito hasta alcanzar las salidas. Sin embargo, el tiempo se entiende como una limitación, ya que se pretende que cada proceso de verificación sea lo más rápido posible.

La calidad de los patrones está relacionada con llevar al sistema a diversos estados internos, para así chequear sus diferentes modos de operación. El testeo por fuerza bruta, que consiste en analizar todas las combinaciones de entradas posibles para cada estado interno, es inviable debido a la complejidad de los sistemas de hoy en día. Por ello, los vectores de entrada han de ser representativos del funcionamiento del sistema. Es decir, han de provocar transiciones en el mayor número de puntos posibles. Dichas transiciones han de propagarse hasta las salidas del circuito o los puntos de test, por lo que el test ha de tener una duración adecuada. En la práctica los vectores de test son semejantes a las entradas de un entorno real, pero con una frecuencia de ocurrencia de eventos mucho mayor.

Relacionado con estos conceptos se define el concepto de "cobertura de test" (test coverage), que indica el número de errores detectables respecto del número de errores reales. Un test con cobertura del 100 % es aquel en el que todos los errores son detectados. Para lograr una elevada cobertura es precisa la utilización de patrones de test de calidad durante un periodo de tiempo suficiente.

5.4.1.1 Tipos de patrones de entrada

Los patrones de entrada se pueden clasificar de acuerdo a su naturaleza, pudiendo éstos ser pseudoaleatorios, almacenados o combinaciones de ambos. Los patrones **pseudoaleatorios** se generan mediante un generador de números aleatorios, pero utilizando siempre la misma semilla y partiendo del mismo estado. De esta forma siempre se genera la misma secuencia, a fin de realizar siempre el mismo test. Las entradas generadas tienen una distribución equilibrada de unos y ceros, y están incorreladas entre sí.

Este tipo de patrones es adecuado para sistemas que no requieran ningún tipo de formato especial en las entradas. Sean ejemplos de esto los circuitos aritméticos o los módulos de cifrado, ampliamente empleados en entornos académicos. Sin embargo, los sistemas reales de hoy en día pueden implementar muchas funcionalidades diferentes, y este tipo de patrones suele no ser adecuado. Es relativamente

sencillo de implementar en una FPGA y ocupa pocos recursos.

Por otro lado, los patrones **almacenados** son aquellos que se almacenan en una memoria para después ser enviados a las entradas del circuito. De esta forma se pueden solventar las exigencias de formato que los patrones pseudoaleatorios no son capaces de afrontar, ya que la información almacenada va a cumplir con los requisitos de calidad del patrón de entrada. El problema de este tipo de patrones es que el tiempo de test es proporcional al tamaño de la memoria, y una verificación suficientemente larga puede requerir gran cantidad de recursos de almacenaje. Puede plantearse la implementación de este tipo de patrones mediante las BRAM de la FPGA siempre y cuando no sean muy largos.

En tercer lugar está la posibilidad de combinar patrones almacenados y pseudoaleatorios. Esto puede aplicarse a circuitos de comunicaciones, en los que los bits con un formato obligatorio (cabeceras de los paquetes, direcciones, etc+ 6*969rd) se obtienen de una memoria, y desde un generador pseudoaleatorio los bits de carga que no van a ser procesados, o que su procesamiento no va a condicionar el modo de funcionamiento general del sistema. De esta forma se solventan las limitaciones de ambos enfoques, reduciendo drásticamente los requisitos de almacenaje y permitiendo la utilización de patrones pseudoaleatorios. Sin embargo, se requiere un mayor esfuerzo de diseño para el generador de patrones.

La cuarta posibilidad es la de no emplear ningún tipo de patrón de entrada. Esto puede hacerse en casos muy puntuales. Por ejemplo, cuando el circuito a verificar está basado en un procesador soft, ya que se ejecuta un programa concreto almacenado en una memoria interna para acabar generando un resultado final al concluir la ejecución. Sin embargo, rara vez van a trabajar estos procesadores en un entorno real sin recibir ningún tipo de dato de entrada. En este caso, la ausencia de patrones no permite analizar los bits críticos que pudiera haber en los distintos controladores de entrada/salida. Obviamente, este procedimiento no es válido para un caso genérico, pero puede aplicarse en casos puntuales.

5.4.1.2 Implementación del generador de patrones

En el contexto de un sistema de evaluación de SEUs en FPGAs, el generador de patrones puede ser implementado en:

- En la propia FPGA
- En otros elementos del PCB
- En un PC externo
- En un entorno real

- No implementar generador de patrones

Una de las posibilidades para la implementación del generador de patrones es la implementación **interna en la FPGA**. Para ello, las entradas del entorno real se multiplexan con las generadas por este módulo. Ésta es una solución intrusiva, y distorsiona la medida de bits críticos. Los bits críticos pertenecientes a los IOBs relativos a las entradas no se detectan. También queda fuera del alcance del test el rutado desde los pines de entrada hasta el multiplexor del generador de patrones, lo que se ha representado en color morado en la figura 5.4. Por otra parte, los bits respectivos al generador de patrones se contabilizan como críticos, aunque no vayan a provocar interrupciones de servicio cuando el sistema esté funcionando en un entorno real. En contraposición con el subsistema de inyección, cuando un SEU emulado afecta al generador de patrones, el sistema no se bloquea, sino que el bit modificado se contabiliza como un bit crítico más.

Con el objeto de reducir la intrusividad, la implementación interna del generador de patrones no debe tomar demasiados recursos de la FPGA. Esto limita el uso de patrones almacenados demasiado largos. Aún así, en [3] se propone una implementación de este tipo utilizando las BRAM. Las combinaciones de generadores pseudoaleatorios y patrones almacenados son adecuadas en este caso. La implementación interna del generador de patrones permite una gran testeabilidad, ya que los patrones pueden dirigirse tanto a las entradas del circuito como a puntos intermedios.

En [7, 8], los estímulos se generan en una FPGA auxiliar localizada en **el propio PCB**. En este caso se dispone de una FPGA auxiliar para la generación de patrones. Esto permite la generación de cualquier tipo de estímulo (pseudoaleatorio, almacenado o combinado) sin ningún tipo de restricción. Tal y como se ha descrito previamente, el problema de este tipo de implementaciones es que resulta complicado implementar esto en la tarjeta final, por lo que se procede a realizar una primera implementación para el test y una segunda para la tarjeta que funcione en el entorno real, lo que plantea incompatibilidades con la verificación en V.

Otra de las posibilidades es utilizar un **PC externo** para este propósito. En este caso no se pueden enviar estímulos sincronizados con el reloj de la FPGA, ya que habría que transmitirlos a través de uno de sus puertos de comunicaciones. Este sistema no garantiza que la entrada sea repetible para cada ciclo de reloj, por lo que no se puede considerar una solución universal para todos los casos. En [6] el circuito implementado en la FPGA es una cadena de módulos de cifrado, a cuya entrada se envía una palabra y se espera hasta que la computación finalice y se obtenga una respuesta. Es una solución que puede ser válida en entornos académicos, pero su aplicabilidad a sistemas reales es reducida.

En [1] las entradas se generan desde **un entorno de trabajo real**. La complicación aquí reside en garantizar la repetibilidad del patrón de entrada. Además, en un caso general va a requerirse más tiempo para llegar a estados representativos del funcionamiento del sistema, ya que no se plantean entradas especialmente preparadas para este propósito. Tal y como se ha mencionado, el testeo de procesadores soft no requiere la utilización de un generador de estímulos, tal y como sucede en [4] [5] [2].

5.4.2 Analizador de respuestas

La etapa final del proceso de verificación es el análisis de la respuesta, que va a consistir en la comparación entre las salidas en presencia de emulación de SEUs, con las salidas en ausencia de ésta (golden), para poder así discernir si el bit de la memoria de configuración sobre el que se ha realizado la inyección del error es crítico o no.

Los mecanismos de análisis de respuesta se dividen en dos grupos:

- Verificación ciclo a ciclo o verificación universal
- Verificación funcional

La **verificación universal** o verificación ciclo a ciclo consiste en comparar las salidas del circuito con las salidas golden en cada ciclo de reloj. Esto propicia que el sistema de verificación no dependa en nada de la aplicación que está siendo validada. Por lo tanto, los mecanismos de emulación que implementen la verificación mediante este esquema pueden ser adaptados para diferentes sistemas sin ningún tipo de esfuerzo de diseño extra. Por esto, se ha nombrado como verificación universal. En este caso, el UUT se entiende como una caja negra, ya que no es necesario conocer su funcionamiento para implementar el analizador de respuestas.

Existen tres vías para implementar un esquema de verificación ciclo a ciclo:

- Duplicación
- Memoria
- Firma

Como su propio nombre indica, la **duplicación** consiste en duplicar el UUT. Mientras los errores se inyectan en una de las copias, la otra queda libre de errores. Las salidas de ambos módulos se comparan ciclo a ciclo, y en caso de detectar alguna disparidad se considera que la inyección se ha llevado a cabo sobre un bit crítico. Esta técnica exige que ambos módulos compartan señal de

reloj, y que las entradas estén perfectamente sincronizadas entre sí. En este caso, no es estrictamente necesario que los patrones de test se repitan constantemente, ya que la respuesta golden se genera en el mismo instante que la respuesta del UUT bajo inyección de errores. Por esto, se permiten las entradas generadas desde PC externo o entorno real.

En [2, 7] los módulos golden y UUT están ubicados en FPGAs separadas, lo que no añade ninguna complicación, más allá de la necesidad de compartir reloj. Por otro lado, en [8] [1] ambos módulos se ubican en la misma FPGA. Esto plantea el mismo tipo de problemas de intrusividad que en el caso de implementar internamente el subsistema de generación de patrones, con el agravante de que el módulo golden incrementa el tamaño del diseño en un 100 %. En sistemas FPGA reales esto no suele ser posible, ya que es frecuente que la ocupación de las FPGAs supere el 50 %, y por tanto, no haya espacio físico para duplicar el UUT. Cabe destacar que el objetivo de los trabajos [1, 8] es el de comparar diferentes arquitecturas, más que el de obtener el valor real de bits críticos.

La verificación de **memoria** consiste en almacenar en una memoria los valores de las salidas del circuito en cada ciclo de reloj. Esto se conoce como respuesta golden, y se va a comparar con la salida del UUT en presencia de errores. Es estrictamente necesario que el patrón de entrada sea exactamente el mismo, y que esté perfectamente sincronizado para cada proceso de verificación. Ésta técnica se aplica en [4], donde se utiliza la memoria BRAM del dispositivo para este propósito. En este caso, la problemática es la misma que para el generador de patrones almacenado en memoria ya que a medida que la duración de la verificación aumenta, crecen las necesidades de capacidad de almacenaje.

La tercera posibilidad es la verificación mediante **firma**, de la misma forma que se lleva a cabo en los sistemas de BIST (ver sección 4.4.1) [105]. Esto consiste en comprimir el stream de salida en una única palabra de memoria, que se denomina firma. Se genera la firma golden en ausencia de errores, y posteriormente se compara con las firmas generadas en presencia de errores. Esto se puede implementar de manera sencilla en una FPGA, reduciendo considerablemente el efecto de la intrusividad.

En contraposición con la verificación ciclo a ciclo, la verificación **funcional** consiste en desarrollar un test específico para la aplicación que se está testeando. Por ejemplo, en [6] se testea una aplicación de cifrado. Para ello, un valor no cifrado se envía al sistema y se recibe la respuesta cifrada, que se compara con el resultado teórico. Los tests sobre procesadores soft, utilizan este tipo de verificación, que consiste en chequear que el programa ha alcanzado un determinado punto y que sus variables tienen unos valores concretos.

Este tipo de verificación se emplea generalmente en aplicaciones del ámbito académico, en el que se implementan circuitos especiales que difícilmente van a trabajar en un entorno real. Los sistemas reales suelen implementar funcionalidades múltiples y muy complejas, lo que dificulta sobre manera la aplicación de un test sencillo de este tipo. La verificación funcional es específica para cada aplicación, y requiere empezar de cero a diseñar el proceso de verificación cuando se plantea validar distintos sistemas.

5.5 Recuperación

El objetivo del proceso de recuperación es el de hacer volver al circuito al estado inicial libre de errores. Una vez que un error ha sido inyectado en la memoria de configuración, y se ha verificado si es crítico o no para el funcionamiento, hay que hacer volver al sistema a la situación inicial para testear la criticidad de otro nuevo bit.

En el caso de aplicar verificación ciclo a ciclo, la secuencia de salida ha de ser coincidente en todo momento con la respuesta golden. Esto exige que el patrón de entrada sea siempre idéntico. Pero no solo eso, sino que el estado del sistema ha de ser idéntico cada vez que se inicia una verificación para garantizar la repetibilidad de la respuesta. Por lo tanto, los elementos secuenciales del circuito han de ser llevados a su estado inicial. En caso de que la verificación sea funcional, la recuperación de los elementos secuenciales puede no ser necesaria, dependiendo de la aplicación.

El mecanismo más común para hacer volver a todos los elementos secuenciales a su estado inicial es la señal de reset. Esto se plantea en la gran mayoría de sistemas de emulación de SEU analizados en la literatura. Sin embargo, esto solamente puede hacerse cuando todos los flip-flops del sistema estén conectados a la misma señal de reset, o a señales de reset fácilmente accesibles desde el exterior. Esta no es una casuística general, debido a que los sistemas implementan cada vez funcionalidades más complejas. En ocasiones, la señal de reset de algún submódulo se genera en entidades internas, tales como maquinas de estado no accesibles desde el exterior. Otros flip-flops ni siquiera hacen uso de la señal de reset, ya que el uso de esta señal ha de ser evitada a menos que sea estrictamente necesaria, tal y como se sugiere en el documento de Xilinx [30] a fin de facilitar las operaciones de emplazamiento y rutado.

Además de esto, los circuitos implementados en FPGAs contienen elementos secuenciales que ni siquiera tienen señal de reset. La memoria RAM es otro ejemplo de esto, ya sea implementada en las BRAM o en las LUTs como memoria distri-

buida. En caso de ser utilizada en procesadores, esto no supone ningún problema, ya que siempre se inicializa la memoria de datos con contenido válido antes de utilizarse, mientras que la memoria de programa es de solo lectura y sus valores nunca cambian. Sin embargo, cuando esta memoria se utiliza con otros propósitos (por ejemplo buffer FIFO para sincronizar dominios de reloj), el retorno al estado inicial no es tan inmediato y puede ocasionar problemas. Otro ejemplo claro de elemento secuencial que no contiene reset son los registros de desplazamiento implementados en las LUTs.

En general, la etapa de recuperación no está cubierta por los métodos estudiados en la literatura. En [8] se comenta que en ocasiones, la señal de reset no va a ser suficiente para hacer volver al sistema al estado inicial. En estos casos, es obligatoria una reconfiguración completa del dispositivo, ya que el bitstream completo contiene los valores iniciales de todos los elementos secuenciales del circuito.

El proceso de recuperación es también responsable de revertir la inyección realizada en la memoria de configuración antes de realizar una nueva inyección. En caso de que la inyección se haga mediante bitstreams parciales, es necesario revertir el bit inyectado antes de modificar un nuevo bit. En este caso, el proceso de recuperación se inicia por la reparación de la memoria de configuración, después sigue la recuperación del estado de los elementos secuenciales y finalmente se realiza una nueva inyección. Por el contrario, en caso de hacer la recuperación mediante bitstream completo, la reparación de la memoria de configuración se hace de manera simultánea con la reparación del estado. El bitstream completo puede aprovecharse incluso para realizar la próxima inyección.

5.6 Métodos de emulación de SEU en FPGAs

En esta sección se presentan los métodos existentes de emulación de SEUs en la memoria de configuración mediante modificación del bitstream. Estos métodos ya han sido citados durante las anteriores secciones, a la hora de analizar las diferentes posibilidades para cada etapa del sistema de emulación. Para cada método se analiza el objetivo para el cual ha sido desarrollado, su arquitectura y cómo se realizan las fases de inyección, verificación y recuperación.

5.6.1 Straka et. al. [1]

El objetivo para el cual este emulador de SEU ha sido diseñado, es la evaluación de una arquitectura de tolerancia a fallos concreta. La arquitectura de tolerancia

a fallos que se plantea en el artículo es una DWC (Duplication With Comparison), que en caso de detectar errores en una de las réplicas, la repara mediante el core de Xilinx SEM [130]. Este core utiliza el ICAP y puede utilizarse tanto para la inyección de errores como para la recuperación de la memoria de configuración.

Para validar dicha arquitectura, los autores plantean una serie de campañas de inyección de errores. Deciden no emplear el SEM con este propósito, ya que es parte del sistema, y consideran inadecuado utilizarla para la inyección de errores. Por ello, diseñan un sistema de emulación de SEUs externo, que hace uso de la interfaz JTAG.

La arquitectura del sistema se presenta en la figura 5.6. El subsistema de inyección se implementa íntegramente en un PC externo. Por lo que el sistema consta únicamente de dicho PC y de la tarjeta que contiene la FPGA corriendo la arquitectura planteada.

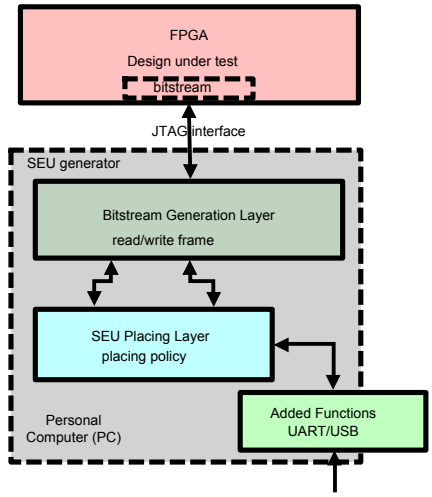


Figura 5.6: Straka

- Inyección: Externa (JTAG), se utilizan bitstreams de 1 frame.
- Verificación: No hay generación de patrones, el sistema trabaja con entradas reales. El análisis de respuesta lo hace la propia arquitectura DWC que se plantea en el artículo.
- Recuperación: La arquitectura tolerante a fallos que se implementa es la encargada de corregir los errores inyectados. Desde el ICAP. No se requiere

recuperar el estado inicial porque se trabaja con entradas reales en lugar de con patrones.

5.6.2 Battezzatti et. al. [2]

En este artículo se plantea un procedimiento de análisis estático, que ya ha sido explicado en la sección 5.2. El objetivo del procedimiento planteado es realizar una estimación de la cantidad de bits críticos del sistema partiendo del bitstream y de los reportes generados por el software de Xilinx. Para validar su propuesta se plantea un sistema de emulación de SEU.

Al igual que en el anterior caso, se trata de un sistema de emulación externo, que hace uso del interfaz JTAG para la carga de bitstreams desde un PC externo (cable USB en la figura). La arquitectura del sistema es muy sencilla, el sistema únicamente consta del PC externo y del PCB con la FPGA (figura 5.7). La aplicación implementada en el UUT es un procesador soft. En este caso, existe un segundo PCB con otra FPGA en la que se implementa el UUT en ausencia de errores, a fin de generar las salidas golden.

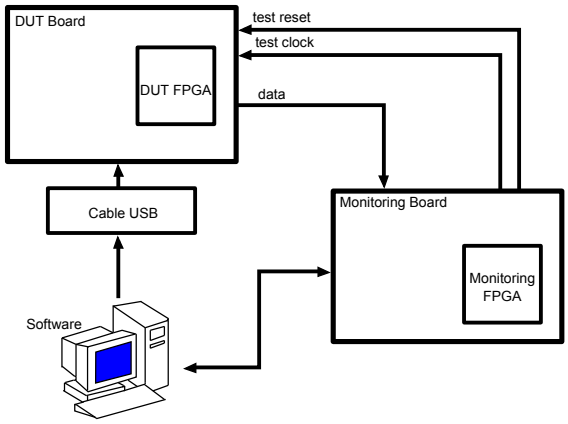


Figura 5.7: Battezzatti

- Inyección: Externa (JTAG), se utilizan bitstreams completos.
- Verificación: Se implementa un procesador soft. No hay generación de patrones. Se comparan las salidas con las de una FPGA idéntica libre de errores.

- Recuperación: La inyección del próximo bit mediante bitstream completo revierte la anterior inyección y recupera el estado inicial.

5.6.3 FIRED, Nunes et. al. [3]

En este artículo se plantea propiamente un sistema de emulación de SEU cuyo objetivo es la estimación del número de bits críticos de un UUT concreto implementado en una FPGA. En este caso se implementa una aplicación real "PID-based cruise control system". La UUT se implementa en un pblock, y la inyección de errores se hace internamente desde el ICAP mediante bitstreams de 1 frame. En la figura 5.8 se puede apreciar la arquitectura general de este sistema.

El subsistema de inyección se implementa en la propia FPGA, en un módulo llamado IRC, (Injection Runtime Controller) basado en MicroBlaze. La verificación es funcional y se lleva a cabo desde un PC externo. El sistema de verificación implementado es bastante básico, el PC recibe el resultado del UUT y lo compara con el golden run. En el artículo se comenta que este mecanismo de verificación no puede aplicarse para un caso general, ya que solamente es aplicable a UUTs deterministas. Los errores inyectados en la CRAM se recuperan mediante el ICAP desde el IRC que ha hecho la inyección. El estado se recupera mediante un reset.

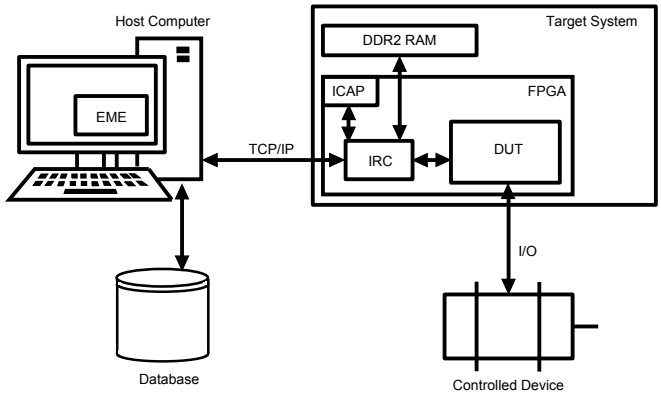


Figura 5.8: FIRED

- Inyección: Interna (ICAP), se utilizan bitstreams de 1 frame.
- Verificación: Funcional. se implementa en un PC externo.
- Recuperación: La CRAM se recupera desde el ICAP. El estado del UUT mediante reset.

5.6.4 Sterpone et. al. [4]

En este artículo se presenta un sistema de emulación de SEU bastante similar al presentado previamente. En este caso, el objetivo es el de comparar la resiliencia al SEU de diferentes IPCores. Tal y como se puede observar en la figura 5.9, la inyección de errores se sigue haciendo desde el ICAP mediante bistreams de 1 frame, sobre una UUT implementada en un pblock. En este caso, el subsistema de inyección se implementa internamente en el dispositivo, pero en un procesador hard (PowerPC), que no se ve afectado por las inyecciones de SEUs que se puedan realizar. Sin embargo, el PowerPC se comunica con el ICAP a través del bus OPB, un bus soft que puede ser afectado por las inyecciones y bloquear así el susbsistema de inyección.

El sistema cuenta con un PC externo, que actúa como elemento maestro, y se comunica con el software del PowerPC para indicarle qué campañas de inyección tiene que realizar. El subsistema de verificación se implementa de manera interna. La generación de patrones se lleva a cabo en una unidad llamada "timing unit". Esta unidad genera únicamente un reloj y un reset, que son los estímulos que ponen en marcha la UUT. Los resultados generados por la UUT se envían a través del bus OPB al PowerPC, en donde se comparan con los datos golden, almacenados en una memoria dedicada para este procesador.

La recuperación de la memoria de configuración se hace a través del ICAP, al igual que en el caso anterior. Para recuperar el estado inicial del UUT existe un reset. En el artículo se afirma que en el caso de que dicha UUT contenga algún tipo de memoria interna, es necesaria una fase de pre-inicialización.

- Inyección: Interna (ICAP), el subsistema de inyección se implementa internamente en un procesador hard.
- Verificación: Se implementa internamente, hay un generador de patrones interno muy básico.
- Recuperación: La CRAM se recupera desde el ICAP, el estado del UUT mediante reset y una etapa de pre-inicialización.

5.6.5 Ghaffari et. al. [5]

En este caso se propone un sistema de inyección similar a los anteriores. Las inyecciones se hacen desde el ICAP, controladas por un sistema de procesador soft interno y la UUT se implementa en un pblock. El aspecto más novedoso de este sistema es la aplicación de la metodología IDF (Isolated Design Flow)[131]. La aplicación de esta técnica garantiza que el rutado de la parte estática no haga

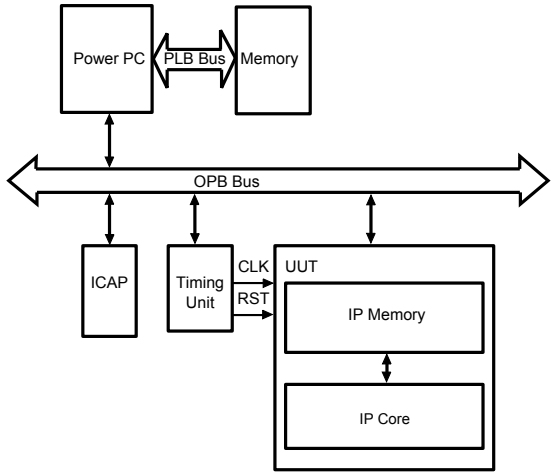


Figura 5.9: Sterpone

uso del pblock en el que se implementa la UUT. Sin embargo, esto se consigue a un coste muy alto, ya que esta metodología exige que una serie de zonas de la FPGA queden vacías a modo de barrera, lo que reduce la capacidad útil del dispositivo. Además, los recursos de rutado válidos para cada módulo disminuyen notablemente. La aplicación del IDF es además muy restrictiva con el uso de pines de entrada y salida, así como con las señales que comunican módulos aislados entre sí. De esta forma, se evita que el sistema de inyección pueda inyectar errores sobre sí mismo.

En el artículo no se hace mención alguna acerca de la verificación. La aplicación testeada es un contador binario, cuyo funcionamiento se evalúa de forma funcional, y cuyo estado inicial se recupera mediante un reset.

- Inyección: Interna (ICAP), Se hace uso de la metodología de aislamiento IDF. Se evitan las inyecciones de errores sobre el subsistema de inyección.
- Verificación: Funcional, la aplicación que se testea es un contador binario.
- Recuperación: La CRAM desde el ICAP, el estado del UUT mediante reset.

5.6.6 Kretzschmar et. al. [6]

En este trabajo también se propone un método de inyección interno basado en el ICAP mediante bitstreams de 1 frame. Se utiliza el core de Xilinx “SEU con-

troller" [132], que contiene un procesador Picoblaze para gestionar la interfaz de configuración. Este core se comunica con un PC externo a través de una interfaz de comunicación serie (RS-232), desde donde se le indica en qué puntos de la memoria de configuración ha de realizar las inyecciones de errores.

No hay ninguna restricción de localización para la UUT, ya que no se utilizan pblocks. Por lo tanto, el sistema de inyección puede inyectar errores sobre sí mismo, bloqueando el sistema. Para solucionar esto, se prevé un ciclo de reconfiguración externo, basado en cargar el bitstream completo desde el PC a través de la interfaz JTAG.

El proceso de verificación se lleva a cabo desde un PC externo y se hace de forma funcional. La aplicación implementada es una cadena de módulos de cifrado, en cuya entrada se introduce una palabra sin cifrar y se recibe cifrada a la salida. No hay necesidad de recuperación del estado interno del sistema, ya que se trata de un sistema aritmético. Los errores introducidos en la CRAM se revierten desde el mismo punto desde el que se han inyectado (el ICAP). Cada cierto tiempo se realiza una reconfiguración completa desde el JTAG para evitar los bloqueos debidos a las inyecciones en el subsistema de inyección. En la figura 5.10 se muestra un diagrama del sistema en el que se indican las casuísticas que exigen una reconfiguración desde el ICAP (inner loop), o desde el JTAG (outer loop).

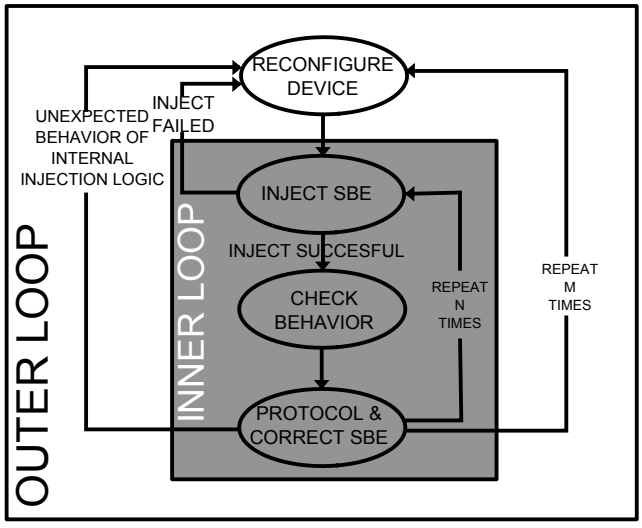


Figura 5.10: Kretzschmar

- Inyección: Interna (ICAP), mediante el core Xilinx SEU Controller.

- Verificación: Funcional, desde un PC externo, a aplicación implementada se trata de una cadena de módulos de cifrado.
- Recuperación: La CRAM desde el ICAP, reconfiguración completa desde el JTAG para desbloquear el subsistema de inyección. No hay necesidad de recuperación del estado interno.

5.6.7 FT-UNSHADES2, Mogollon et. al. [7]

En este artículo se presenta una plataforma para la evaluación de la robustez de los sistemas frente a SEEs. Esta plataforma, que cuenta con el respaldo de la ESA (European Space Agency), se compone de dos tarjetas idénticas. Una va a emplearse para la implementación del UUT, mientras en la otra se va a correr una réplica exacta en ausencia de errores, para generar así la secuencia golden en tiempo real. Cada una de estas tarjetas consta de dos FPGAs, una de ellas para la implementación del UUT, y la otra para la generación determinista y sincronizada de patrones de entrada.

El sistema dispone también de una tarjeta base, a la que se acoplan estas dos. En dicha tarjeta se implementa la comparación de salidas. Además, actúa como elemento maestro, ordenando las inyecciones a realizar coordinando las tarjetas en las que se implementa el test. También se realizan funciones de interfaz de usuario. La inyección de errores se lleva a cabo a través de la interfaz SelectMAP. La arquitectura de este sistema se representa en la figura 5.11.

La plataforma puede emplearse para la validación tanto de ASICs como de diseños implementados en FPGAs. En el primero de los casos, el punto de partida es la descripción del ASIC en formato netlist. Dicha descripción se implementa en ambas tarjetas y se procede a las inyecciones de errores en una de ellas. En este caso, las inyecciones se realizan principalmente en los elementos secuenciales del circuito (flip-flops y memorias), ya que los ASICs no contienen memoria de configuración. Se pueden plantear inyecciones en la CRAM para emular defectos a nivel hardware.

Cuando la plataforma se emplea para la validación de circuitos implementados en FPGAs, se parte del diseño sintetizado en netlist, y las inyecciones se efectúan sobre la CRAM. El sistema de inyección es de alta velocidad (bitstreams de 1 frame a través de SelectMAP), y no hay problemas de bloqueo por inyección de errores sobre sí mismo. La verificación no es intrusiva, ya que se realiza íntegramente exteriormente a la FPGA en la que se encuentra el UUT.

El procedimiento de verificación es de caja negra, comparando ciclo a ciclo de reloj el valor de las salidas del UUT con las salidas del módulo golden. De esta

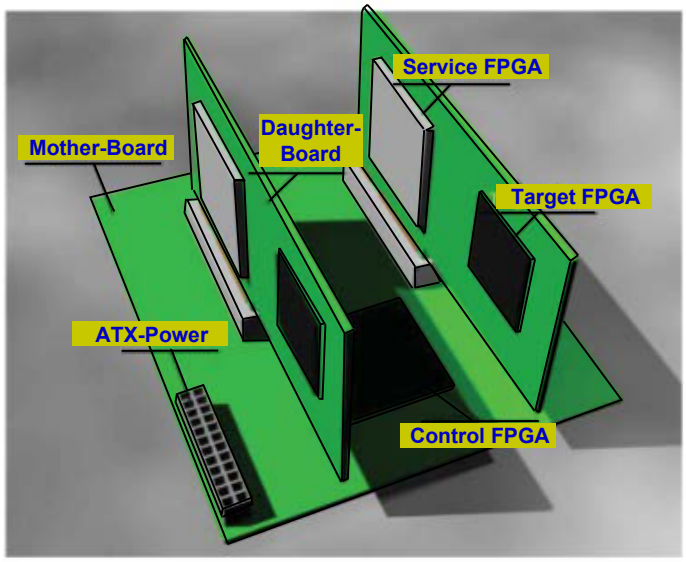


Figura 5.11: FT-UNSHADES2

forma el sistema es fácilmente adaptable para el testeo de diferentes aplicaciones. El punto más débil de esta plataforma es la necesidad de re-adaptar el diseño para adaptarlo a la tarjeta en la que debe trabajar, ya que esto plantea incompatibilidades con el modelo de validación en V. La recuperación del estado no se aborda en el artículo, aunque puede llevarse a cabo de múltiples maneras debido a la gran cantidad de posibilidades que ofrece una plataforma tan completa.

- Inyección: Externa, Bitstreams de 1 frame a través del SelectMAP.
- Verificación: Universal, ciclo a ciclo, externa, generación de patrones en FPGA auxiliar.
- Recuperación: No se aborda en el artículo.

5.6.8 XRTC-V5FI, Harvard et. al. [8]

(XRTC-V5FI) Xilinx Radiation Test Consortium Virtex5 fault injector

Aquí se presenta una plataforma de test cuyo procedimiento de inyección de errores es similar al anterior. Las inyecciones se realizan a través de la inter-

faz SelectMAP, desde un PCB anexo al que contiene la FPGA con el UUT. El objetivo que se plantea es la estimación de bits sensitivos de procesadores soft. Debido a que la verificación puede hacerse de forma funcional y sencilla no se requiere la FPGA de servicio de la anterior plataforma ni la tarjeta golden para la comparación. La aplicación que se testea es el benchamrk "towers of Hanoi".

Los puntos fuertes y débiles del sistema de inyección son los mismos que en el caso anterior. La mayor ventaja es que la inyección se realiza desde fuera de la FPGA, lo que impide que pueda bloquearse al ser afectado por un error. Por otro lado, el inconveniente es la incompatibilidad con el modelo en V, ya que se requiere de una nueva implementación para la tarjeta en la que va a trabajar el diseño. El esquema de la plataforma se representa en la figura 5.12.

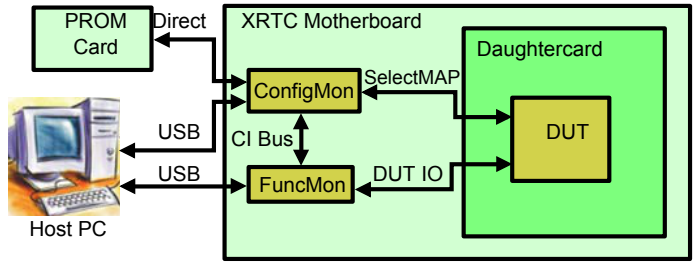


Figura 5.12: Harvard

- Inyección: Externa, Bitstreams de 1 frame a través del SelectMAP.
- Verificación: Benchmark towers of Hanoi, no hay generación de patrones. Verificación externa.
- Recovery: Reset, se avisa que no es siempre suficiente, que en algunos casos puede ser necesaria una reconfig. completa

5.7 Conclusiones

En primer lugar se han introducido los diferentes mecanismos de previsión de fallos para orientar los sistemas a confiabilidad desde un punto de vista general. Éstos se fundamentan en una evaluación de la tasa de fallo global del sistema, para poder así certificar que éste no vaya a fallar más de lo que está establecido en las especificaciones de confiabilidad.

Después, se han particularizado las opciones de evaluación para el estudio de la tasa de SEU en la memoria de configuración de una FPGA. Se ha concluido

que este estudio consta de dos partes. Por un lado están los tests de radiación, que proporcionan la tasa de SEU global del dispositivo, y por otro la estimación del número de bits críticos del sistema. Esta estimación puede realizarse bien por métodos analíticos, partiendo de los reportes generados por el software de desarrollo de FPGA, o bien mediante emulación de SEU, que consiste en la configuración errónea del dispositivo emulando el efecto de un SEU.

Posteriormente se ha profundizado en las diferentes fases de los sistemas de emulación de SEU. De esta forma se ha realizado un análisis pormenorizado de todas y cada una de las diferentes etapas de las que constan este tipo de sistemas. Finalmente, se han estudiado diferentes alternativas propuestas para la emulación de SEUs. En este punto, se ha discutido la implementación de cada una de las fases del sistema de emulación para cada uno de estos mecanismos, a fin de poder identificar los puntos fuertes y débiles de cada uno de ellos. Gracias a este análisis se extraen las siguientes conclusiones:

- La mayoría de los mecanismos analizados se centran principalmente en la fase de inyección. Se presentan diagramas detallados de la arquitectura del sistema, y describen en detalle el proceso de inyección de errores. Por otra parte, la mayoría de los sistemas analizados rehúyen abordar en detalle el proceso de verificación, centrándose en el estudio de una aplicación concreta. En muchos casos se implementan procesadores soft, para los que se puede implementar un esquema de verificación muy sencillo. Esto supone un problema a la hora de plantear un emulador de SEU universal, que pueda ser fácilmente adaptado a diferentes aplicaciones complejas.

- Los esquemas que plantean una verificación universal se basan en duplicar la UUT, de modo que los errores se inyecten en una de las réplicas, dejando la otra como golden. Implementar internamente la UUT golden no es viable, a menos que el sistema final plantee una arquitectura DWC. La inclusión de una segunda réplica en el mismo dispositivo requiere duplicar los recursos empleados. Además, no sería viable eliminarla en la implementación final debido a que esto quebrantaría el esquema de validación en V. Otra posibilidad es implementar la segunda réplica en una tarjeta externa, idéntica a la que contiene la UUT. Pero esto tampoco es compatible con la validación en V a menos que la tarjeta de test sea la misma que la tarjeta que trabaje en el entorno real.

- La problemática de la recuperación del estado no se aborda en la mayoría de los esquemas analizados, ya que casi todos estos implementan aplicaciones sencillas, que recuperan el estado inicial mediante un simple ciclo de reset. Sin embargo, esto no puede considerarse como un procedimiento universal. Una alternativa posible es la reconfiguración completa del dispositivo.

- La inyección de errores puede llevarse a cabo internamente desde el ICAP mediante lógica implementada en la propia FPGA. Esto plantea problemas de auto-inyección. Para evitar esto no basta con implementar la UUT en un simple pblock, ya que esto no garantiza que las líneas de rutado de la parte estática no vayan a cruzar dicho pblock. Sería necesaria la aplicación de una metodología de diseño muy exigente como la IDF (Isolated Design Flow).
- El problema anterior puede solventarse realizando la inyección de errores desde una interfaz externa. En caso de utilizar la interfaz SelectMAP, pueden requerirse modificaciones a nivel de PCB, algo que no siempre es viable. La interfaz JTAG puede ser una alternativa viable en caso de utilizar bitstreams de 1 frame, pero resulta demasiado lenta en caso de utilizar bistreams completos.

Parte II

Aportaciones

Capítulo 6

Método propuesto para la emulación de SEUs

Una vez terminada la parte del estado del arte (capítulos 1-5) se procede a la descripción del trabajo realizado y de las diferentes aportaciones de la tesis. La principal es la definición de una metodología de emulación de SEUs en la memoria de configuración de dispositivos que combinan FPGA con sistema procesador. Este capítulo se centra en la presentación de dicho método.

El objetivo es desarrollar un procedimiento de emulación universal, que pueda ser adaptado de una manera sencilla a diferentes sistemas. Es decir, que no sea dependiente de un hardware concreto, siendo el único requisito del sistema el de contener un SoC FPGA del tipo Zynq o similar. Adicionalmente, se pretende que el sistema pueda ser adaptado a diferentes diseños de manera sencilla, independientemente del tipo de circuito que se haya cargado en la lógica programable. De esta forma, tiene que ser posible adaptar el sistema al testeo de diseños industriales reales que puedan implementar múltiples funcionalidades complejas.

En primer lugar se presenta el esquema general de la metodología de emulación propuesta, para posteriormente concretar las implementaciones particulares de los diferentes subsistemas y fases como la inyección, la verificación o la recuperación del estado. Finalmente, se llevan a cabo campañas de emulación para justificar que el método propuesto cumple con los objetivos que se han descrito. Las pruebas de emulación se llevan a cabo en un PCB industrial real, y se testean diseños de diferentes funcionalidades y complejidades.

6.1 Esquema general de la metodología de emulación propuesta

En esta sección se presenta la arquitectura general del método de emulación propuesto en esta tesis. El dispositivo empleado es de la familia Xilinx Zynq7000. Aún así, esta metodología de emulación puede ser aplicada a cualquier SoC que combine FPGA y procesador de otros proveedores, siempre y cuando la FPGA pueda ser configurada desde el microprocesador. El diagrama global se presenta en la figura 6.1.

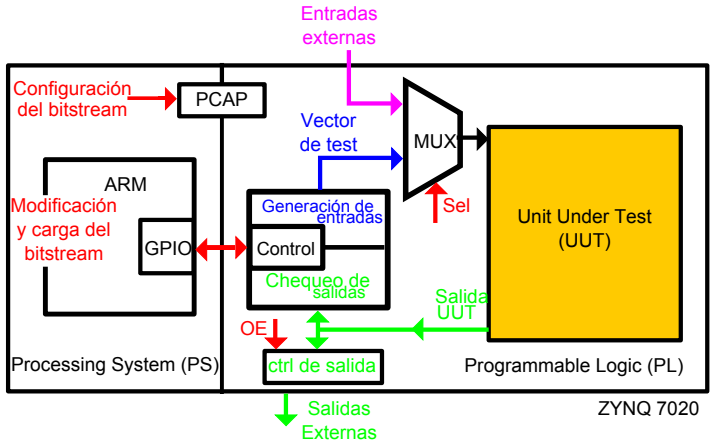


Figura 6.1: Diagrama general de la herramienta de emulación SEU propuesta

El subsistema de inyección (Configuración de bitstream y Software de modificación y carga del bitstream) se ubica en el sistema procesador. De acuerdo con la clasificación de los métodos de emulación de SEUs realizada en el capítulo anterior, el propuesto en este documento utiliza la inyección externa de fallos. Esto permite la inyección por reconfiguración completa y previene los denominados como "injection side-effects", que consisten en el bloqueo del sistema debido a inyecciones de errores en el propio subsistema de inyección.

Sin embargo, este subsistema se ubica en el sistema microprocesador, dentro del mismo chip. De esta forma, es posible utilizar el PCAP, que es una interfaz de configuración paralelo de alto throughput. Asimismo, no son necesarias modificaciones hardware a nivel de PCB, tal y como sucede con la interfaz SelectMAP. Debido a la alta velocidad del PCAP es posible realizar las emulaciones de SEUs mediante bitstreams completos en un tiempo razonable. La utilización de bits-

treams completos también permite llevar a cabo la recuperación del estado inicial de una forma universal.

El subsistema de verificación se implementa internamente en la FPGA. De este modo se lleva a cabo una verificación universal ciclo a ciclo sin necesidad de añadir elementos al PCB. Para la fase de generación de patrones de entrada, pueden emplearse patrones pseudoaleatorios y patrones específicos de la aplicación. Este esquema permite implementar patrones almacenados en las BRAMs. Sin embargo, no es algo que se recomiende, ya que el área ocupada puede ser grande, distorsionando significativamente los resultados de test. La etapa de análisis de respuestas se implementa mediante firma.

Una de las ventajas de los sistemas de emulación de SEUs respecto de los métodos de estimación directa es la posibilidad de testear configuraciones tolerantes a fallos. Estas configuraciones se basan en añadir circuitería a fin de minimizar el efecto de la existencia de errores en el sistema. Mientras que los métodos de estimación directa traducen el aumento de recursos utilizados en un incremento en la tasa de interrupciones de servicio, solamente los emuladores de SEU van a ser capaces de comprobar la eficacia de este tipo de mecanismos.

Entre los esquemas de tolerancia a fallos implementados en FPGAs destaca la arquitectura TMR (Triple Modular Redundancy). Para el testeo de este tipo de configuraciones se propone el triplicado del sistema de verificación, tal y como se presenta en la figura 6.2. La cantidad de bits críticos de una implementación TMR es reducida, comparable al número de bits críticos del subsistema de verificación. Es por esto que se propone triplicar también el subsistema de verificación para que el impacto de éste sobre la tasa de fallo siga siendo mínimo.

6.2 Fortalezas del sistema propuesto

En esta sección se van a reseñar las principales fortalezas de la metodología de emulación de SEUs propuesta en esta tesis. Las características que diferencian a este método del resto de propuestas estudiadas en la literatura son las siguientes:

- Universalidad
- No requiere modificaciones en PCB
- Intrusividad baja
- Compatible con validación en V
- Diferenciación de las tres fases de la emulación de SEUs

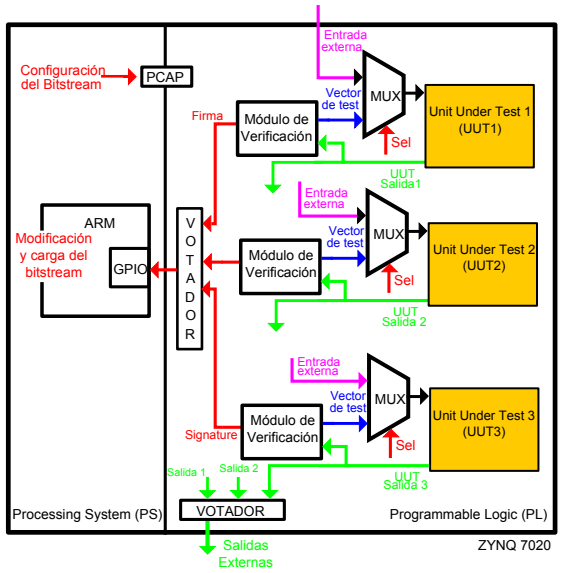


Figura 6.2: Diagrama de la herramienta de emulación SEU propuesta para configuraciones TMR

- Recuperación del estado inicial tras cada inyección

El método propuesto es un método de emulación **universal**, lo que significa que es fácilmente adaptable al testeo de cualquier sistema implementado en un SoC que combine FPGA con sistema procesador. El único requisito para poder ser aplicado es precisamente ese, que tenga un chip tipo Zynq o equivalente. La única etapa que requiere una adaptación dependiente del tipo de circuito es la generación de patrones del proceso de verificación. Esto se debe a la necesidad de enviar al circuito que se está testeando estímulos representativos de su funcionalidad, para poder así testear el correcto funcionamiento de los diferentes módulos y submódulos. Esta característica hace posible el testeo de sistemas industriales complejos, mientras que la mayoría de los emuladores estudiados en la literatura se limitan a circuitos académicos simples.

El análisis de respuestas mediante firmas también facilita esto, ya que no es necesario conocer el funcionamiento de la unidad bajo test para decidir si el funcionamiento es correcto o no. Las salidas del circuito bajo inyección de errores se comparan ciclo a ciclo de reloj con las salidas en ausencia de errores, y en caso de haber discrepancias se considera que el bit modificado es crítico para el funcionamiento correcto del diseño. En la literatura se ha observado que la

mayoría de entornos de emulación propuestos propone un esquema de verificación dependiente de la aplicación que se está verificando.

El hecho de realizar la recuperación del estado inicial mediante reconfiguración completa es un aporte a la universalidad de los esquemas de emulación. La problemática de la recuperación del estado ya ha sido identificada por algunos de los emuladores analizados en la literatura, ya que se considera que un ciclo de reset no es una solución universal para hacer volver al sistema al estado inicial. Esto sucede especialmente cuando las LUTs de los CLBs se utilizan como RAM distribuida o registros de desplazamiento, ya que no tienen pines de reset y su única posibilidad de inicialización es el bitstream.

El método propuesto es adecuado para el testeo de sistemas ya construídos, ya que no se requiere ningún tipo de **modificación a nivel de PCB**. La velocidad de inyecciones es adecuada, llegando a hacerse hasta 26 inyecciones por segundo para el Zynq7000. Esto posibilita que en unas pocas horas pueda haberse llevado a cabo un número representativo de inyecciones. El tiempo necesario para testear la CRAM completa es de 7 días.

La **intrusividad** derivada de la adición de los componentes necesarios para el test es baja. El subsistema de inyección no plantea ningún tipo de problema de este tipo, ya que se implementa íntegramente fuera de la FPGA. Esto plantea una mejora sustancial en comparación con el resto de métodos de inyección interna, ya que todos éstos requieren de lógica de control implementada en el interior de la FPGA.

El subsistema de verificación sí que añade intrusividad al sistema, ya que se implementa dentro de la FPGA. Ésta es la contrapartida de plantear un esquema de verificación ciclo a ciclo sin añadir hardware adicional en el PCB. Sin embargo, ocupa una cantidad de recursos muy reducida dentro de la FPGA, por lo que la distorsión introducida en la medición es pequeña.

La metodología planteada es compatible con el esquema de **validación en V**. La implementación testeada es la misma que va a operar después en un entorno real. Otros métodos de emulación plantean la utilización de hardware específico para el test. Esto requeriría una implementación para dicho hardware, y una vez obtenidos los resultados, sería necesaria una nueva implementación para adaptar el diseño al PCB que trabaja en el entorno real.

Además de esto, las auditorías de safety recomiendan que todos los módulos implementados en la FPGA tengan alguna funcionalidad durante la etapa de operación. Debido a que eliminar el subsistema de verificación sería contrario a la validación en V, se plantea utilizarlo como herramienta adicional de tolerancia a fallos. Este módulo puede utilizarse como un BIST durante la fase de operación,

llevando a cabo operaciones de verificación programadas para chequear el estado del sistema.

Entre todos los esquemas analizados en la literatura, el presentado en este trabajo es el primero que identifica y diferencia las **tres fases existentes** en el proceso de emulación de SEUs: inyección, verificación y recuperación del estado inicial. La práctica totalidad de todos ellos se focaliza en la descripción del proceso de inyección. En muchos de los casos se plantean esquemas de verificación dependientes de la aplicación implementada. La problemática de la recuperación del estado es identificada en alguno de estos métodos, pero no se plantean las acciones que han de ser tomadas para llevar a cabo dicha recuperación del estado.

6.3 Implementación del método en un dispositivo Zynq7000

En esta sección se profundiza en los diferentes aspectos relativos a la adecuación de este método a una tarjeta que contiene un dispositivo de la familia Zynq7000. Para ello se realiza un análisis pormenorizado de las decisiones tomadas para cada una de las fases de las que consta el proceso de emulación de SEUs en FPGAs. En primer lugar se discute la implementación del subsistema de inyección, para luego continuar con el de verificación y terminar explicando el proceso de recuperación del estado inicial.

6.3.1 Subsistema de inyección de errores

El subsistema de inyección consiste en un software implementado en el PS del Zynq, que está compuesto por un sistema procesador hard basado en ARM. Este software es responsable de generar el bitstream corrupto, además de gestionar la interfaz de configuración para cargarlo en la CRAM. Posteriormente, se encarga de iniciar el proceso de verificación, dando la orden de inicio al subsistema responsable de este apartado. Después va a recibir el resultado del subsistema de verificación, que indica si la inyección realizada ha provocado un fallo del sistema o no. Finalmente, almacena dicho resultado, genera estadísticas y las presenta al usuario. La figura 6.3 presenta el flujo del subsistema de inyección.

La primera operación realizada por este software es almacenar el bitstream en memoria direccionable por el procesador. El usuario lo guarda en una tarjeta SD y el software lo almacena en la memoria DDR3. Los dispositivos de la familia Zynq tienen un controlador de tarjeta SD en el PS, además de un controlador DDR

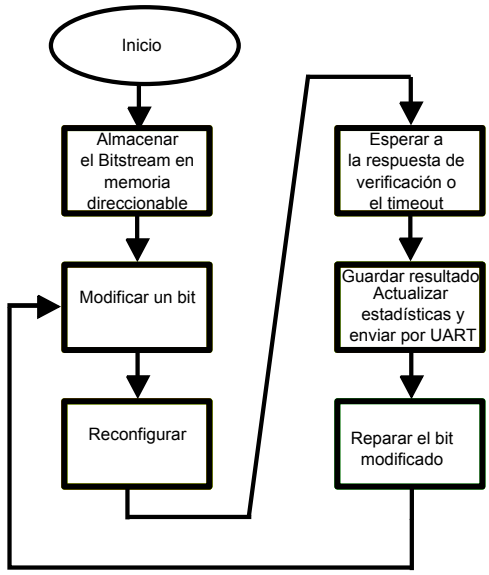


Figura 6.3: Flujo del software de inyección de fallos

para expandir la memoria interna. La tarjeta SD y la memoria DDR3 externa son elementos habituales en los sistemas basados en Zynq. Sin embargo, estos elementos no son estrictamente necesarios. En caso de no existir esto, se ha de plantear un mecanismo alternativo para almacenar el bitstream en la memoria del sistema.

Una vez que el bitstream está almacenado en memoria direccionable, se modifica un bit y se envía el fichero a la interfaz de configuración. Para ello se dispone de librerías que facilitan enormemente este proceso. Cabe reseñar que el fichero de bitstream tiene un campo de CRC, que permite certificar la integridad de dicho fichero y evitar la carga de bitstreams corruptos. Esto obliga a conocer la estructura del fichero de bitstream, para poder así identificar el punto en el que se encuentran los comandos relativos al CRC y eliminarlos. A tal fin, la estructura del fichero bitstream ha sido introducida en la sección 1.6. La utilización de bitstreams corruptos sin CRC se limita a la fase de test, y no compromete ni reduce la confiabilidad del sistema. Cuando el sistema opere en un entorno real se van a utilizar bitstreams normales, sin eliminar el CRC ni modificar ningún bit.

El proceso de configuración del dispositivo es la operación que va a fijar el tiem-

po del test, especialmente cuando las inyecciones se llevan a cabo mediante bitstreams completos, como es el caso. El biststream completo del Zynq7020 tiene un tamaño de 32Mb. La carga de dicho fichero requiere de un consumo de tiempo que ha de ser tenido en cuenta, tanto por la operación de transferencia de datos en sí, como por los tiempos muertos introducidos por las librerías de software.

En el caso de esta implementación concreta, se consigue una velocidad de 26 tests por segundo. Esto significa que el tiempo necesario para analizar todos los bits de configuración de un dispositivo Zynq 7020 es de 14 días. Sin embargo, de estos 32Mb solamente en torno a 18Mb van a almacenarse en la memoria de configuración, ya que el resto se corresponden con bits de valores iniciales de BRAMs o con espacios vacíos, tal y como se ha explicado en la sección 1.6. Cabe recordar que el objetivo del emulador es cargar bits erróneos en la memoria de configuración, y por lo tanto no tiene sentido inducir errores en bits que no van a almacenarse en dicha memoria.

Por otra parte, no es estríctamente necesario testear la criticalidad de todos y cada uno de estos bits, sino que se pueden obtener resultados estadísticamente representativos verificando una muestra más pequeña. Para cada diseño se han realizado 226968 inyecciones de errores, que suponen 1/80 bits del bitstream. Considerando una distribución binomial, el intervalo de confianza del 95 % de que un bit sea crítico viene dado por la ecuación 6.1:

$$Intervalo\ de\ 95\,\%\ de\ confianza = p \pm *z_{95}\sqrt{\frac{p(1-p)}{n}} \tag{6.1}$$

Considerando un valor típico para p = 0.1, (10 %) el intervalo de confianza del 95 % es p = 0.1 ± 0.00123. Esta expresión proporciona la relación entre el número de inyecciones y el intervalo de confianza. Se puede apreciar que al aumentar el número de inyecciones (n) se consigue un margen de confianza más estrecho. El parámetro z_{95} se corresponde con el valor de la distribución normal que comprende el 95 % de la campana de Gauss.

La reconfiguración del dispositivo se realiza a través de la interfaz PCAP (Processor Configuration Access Port), cuya construcción es hard y está ubicada físicamente fuera de la región reconfigurable. Por lo tanto, en ningún caso va a resultar afectado por los errores inyectados.

Después de la finalización del proceso de configuración, el subsistema de inyección da la orden al subsistema de verificación para que comience con el proceso de verificación. En ese instante, el software de inyección queda en espera hasta que el subsistema de verificación proporcione un resultado válido, notificando que la verificación ha finalizado. Cabe la posibilidad de que el subsistema de verificación

se bloquee debido a la inyección de errores, ya que está implementado en la lógica programable. En ese supuesto, salta un temporizador en el software de inyección que indica el bloqueo del proceso de verificación, considerando como crítico el bit en el que se ha hecho la inyección del error.

Finalmente, el bit corrompido es reparado, dejando al sistema preparado para llevar a cabo nuevas inyecciones. No es necesario volver a configurar el dispositivo con el bitstream correcto. Con cargar el bitstream relativo a la próxima inyección es suficiente para recuperar el estado inicial.

Se ha observado la existencia de side-effects en casos muy puntuales cuando las inyecciones de errores se realizan en IOB (bloques de entrada y salida). Si se inyectan errores en los pines de entrada que son esenciales para el PS (por ejemplo, pines de memoria DDR3 externos), el microcontrolador se bloquea y necesita un reseteo para volver al funcionamiento. Cuando esto sucede se detiene el test y se pierden los datos que han sido obtenidos. Se ha comprobado que, de los 19 millones de bits que pueden analizarse, algo así sucede únicamente en menos de 100 bits, lo que se corresponde con un porcentaje ínfimo. Esto se resuelve no inyectando errores en los IOB. Además, los bits relativos a los IOBs quedan fuera del dominio de verificación, tal y como se explica en la sección 6.3.2, por lo que los bits críticos de estos bloques nunca serían detectados.

6.3.2 Subsistema de verificación

Se propone ubicar todo el sistema de verificación en el interior de la FPGA, ya que ésta es la única forma de conseguir altos niveles de testeabilidad sin añadir ningún tipo de hardware adicional. En coherencia con el modelo de validación en V, no es factible eliminar este sistema después de efectuar la verificación, ya que esto supondría una nueva implementación. Este subsistema puede emplearse durante la fase operativa como herramienta de tolerancia a fallos. Se implementa mediante un esquema BIST (Built-In Self Test), ya mencionado en la sección 4.4.1, y puede utilizarse para chequear el correcto estado del sistema.

BIST es una técnica de detección de fallos offline, que requiere necesariamente la detención de la operación del sistema. Permite tanto la detección de defectos en la capa de hardware como de errores en la memoria de configuración. Mientras que el scrubbing únicamente detecta fallos en la CRAM, arquitecturas basadas en redundancia como DWC o TMR introducen un sobrecoste de área del 100-200 %.

La implementación interna del subsistema de verificación tiene dos contrapartidas principales. La primera de ellas es la intrusividad. Los bits de la memoria de configuración relativos a este subsistema se identifican como críticos, aunque

realmente no interfieran en el servicio que está siendo provisto a los usuarios. Sin embargo, esta intrusividad no es bloqueante, ya que el subsistema de inyección recupera el control al saltar el timeout. Por otro lado, cabe recordar que los errores que afectan al subsistema de inyección sí que son bloqueantes.

La segunda contrapartida es el hecho de que los bits críticos relativos al rutado hacia los IOBs y los bits de configuración de los propios IOBs no van a ser detectados como tal. Esto va a hacer que la cantidad de bits críticos medida sea inferior a la real, contrarrestando el efecto de la intrusividad introducida por el sistema de verificación. Los bits críticos que no son detectados, y que sí afectan al servicio, se compensan con los relativos al subsistema de verificación, que sí se consideran críticos pero no interfieren en el servicio. Por este motivo no se van a realizar inyecciones en los IOBs, ya que no van a detectarse bits críticos, además de que estas inyecciones pueden bloquear el procesador ARM.

El peso de estos dos efectos va a ser reducido. Por un lado, el número de señales que son rutadas hacia los pines en un diseño basado en FPGA está limitado por el propio encapsulado y la construcción del PCB. Difícilmente van a superar la(s) centena(s), mientras que un diseño estándar en FPGA puede contener 50000-100000 señales fácilmente. Esto representa un error en torno al 1 % del total. Por otro lado, el subsistema de verificación ocupa menos de 100 LUTs, que en comparación con las 53200 del total del dispositivo Zynq7020 es un porcentaje menor, teniendo en cuenta que lo habitual es que los diseños superen el 50 % de ocupación del dispositivo. En caso de necesitar reducir la criticalidad de este subsistema se le puede aplicar una configuración TMR o similar.

A continuación se presentan las implementaciones concretas que han sido llevadas a cabo tanto para el generador de patrones como para el analizador de respuestas.

6.3.2.1 Generación de patrones

El generador de patrones es el único elemento de toda la metodología presentada que es dependiente de la funcionalidad del sistema que se está testeando. Por lo tanto, requiere ser adaptado cada vez que se testea un diseño diferente. Tal y como se ha explicado previamente, la calidad de los patrones de entrada está directamente relacionada con el concepto de **cobertura de test** (también denominado"test coverage"), que representa el porcentaje de errores detectables respecto del total de errores.

Se requiere un patrón de entrada intensivo, que haga llevar al sistema a estados que sean suficientemente representativos de la funcionalidad del sistema. En otras palabras, que genere actividad a lo largo de todo el sistema. Por actividad en el

sistema se entiende provocar transiciones en la mayor cantidad posible de señales. Además, es necesaria una duración del tiempo de test suficiente para que dicha actividad se propague hasta las salidas del sistema o hasta los puntos de test.

Para explicar mejor el concepto de actividad de las señales, se propone el ejemplo de un sistema que procese tramas Ethernet. Un sistema de este tipo tiene en la entrada un receptor de paquetes, que verifica si la entrada tiene el formato de paquete Ethernet. Si llega un stream de bits (unos y ceros) que no tenga el formato de trama no se va a recibir ningún paquete, por lo que no hay ningún motivo para iniciar el procesamiento. En esta situación, el único punto del sistema en el que hay actividad es en el detector de tramas, mientras que los módulos dedicados al procesamiento de paquetes están a la espera. Estos módulos van a tener por lo general un valor constante en los nodos que lo componen, y no van a enviar ninguna información a las salidas. Por lo tanto, el patrón de entradas no puede considerarse como exhaustivo, y no es adecuado para el test.

En este ejemplo, el patrón que se envía al sistema tiene que tener necesariamente el formato de tramas Ethernet. De esta forma sí que se van a recibir paquetes, que van a ser enviados a los módulos encargados de su procesamiento. En estos módulos, las señales van a tener cambios en sus valores, y sus resultados van a ser transferidos a las salidas. Este tipo de ejemplos va a ser una casuística muy común en diseños implementados en FPGAs, ya que es habitual que estos dispositivos implementen sistemas completos. Los sistemas se comunican habitualmente con los usuarios y con otros sistemas mediante protocolos de comunicaciones e interfaces de comunicaciones. Por lo tanto, los generadores de patrones con unos valores de cabeceras fijos y un payload de contenido aleatorio van a ser una casuística bastante genérica.

Basándose en el concepto de actividad de las señales, en este trabajo se propone una metodología para medir el "test coverage" de los patrones de entrada. Para ello se utiliza la herramienta de medición de potencia de Vivado [133]. Esta herramienta basa su cálculo de potencia consumida en que los dispositivos de tecnología CMOS solamente consumen en las transiciones. El simulador de Vivado indica la actividad de cada señal del diseño para un conjunto concreto de entradas. La finalidad principal de esto es hacer una estimación del consumo de potencia del dispositivo. Sin embargo, se propone utilizar estos datos para la medición del test coverage. Para ello, se dispone a cuantificar el número de señales que no tienen transiciones respecto del total de señales, por lo que la cobertura de test viene dada por la ecuación 6.2.

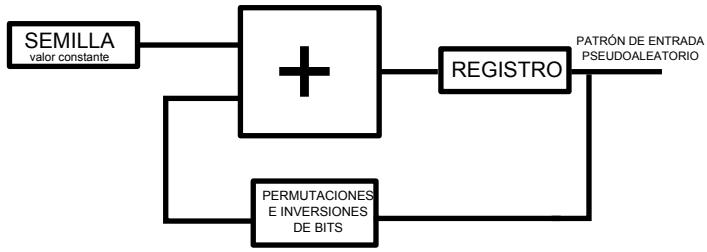


Figura 6.4: Generador de patrones pseudoaleatorios

$$Cobertura\ de\ test = \frac{Errores\ detectados}{Errores\ totales} = \frac{Nets\ con\ consumo\ de\ potencia}{Nets\ totales} \tag{6.2}$$

A continuación se van a presentar las implementaciones particulares de generadores de patrones llevadas a cabo para los circuitos testeados en el presente trabajo. En la figura 6.4 se presenta un generador de números aleatorios con un peso equilibrado de ceros y unos, que es válido para cualquier sistema que no tenga exigencias de formato para las entradas. La secuencia de entrada va a ser siempre la misma, en función de la semilla que haya sido introducida. La implementación concreta consiste en un sumador realimentado con una red de inversiones y permutaciones de bits (figura 6.4). La ocupación de este módulo es de 13 LUTs y 18 flip-flops, por lo que la intrusividad es mínima (el total de LUTs del dispositivo es 53200, y el de flip-flops es de 106400).

En la figura 6.5 se presenta una implementación de un generador de tramas Ethernet. Se trata de una máquina de estados finitos (FSM) que multiplexa la cabecera MAC almacenada, el generador de direcciones MAC, el payload generado en el generador de números aleatorios y el CRC calculado en el generador de CRC. La máquina de estados toma algunos bits del generador de números aleatorios para realizar paquetes de tamaño aleatorio. También hay aleatoriedad en el generador de direcciones MAC. Al igual que en el caso anterior, la intrusividad de este generador de patrones es limitada, ya que contiene 73LUTs y 119 flip-flops.

6.3.2.2 Analizador de respuestas

Se propone un esquema de análisis de respuestas universal, por lo que no es dependiente de la funcionalidad del sistema ni requiere ningún tipo de adaptación cuando se testean sistemas diferentes (más allá de las anchuras de buses de datos,

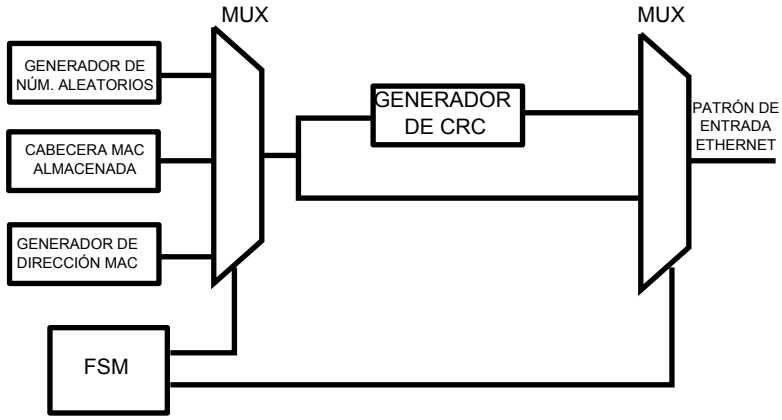


Figura 6.5: Generador de patrones ethernet

que es un parámetro configurable del IPCore). En primer lugar se obtiene el stream de datos de salida en ausencia de errores, al que se le va a dar el nombre de respuesta dorada (golden response). Después, se inyecta el error y se obtiene el stream de salida para el mismo patrón de entrada. Se compara este último con la respuesta dorada en cada ciclo de reloj, y en caso de detectar discrepancias en un solo bit de las salidas, en un solo ciclo de reloj, se considera que el error inyectado ha modificado el servicio, por lo que se etiqueta como crítico.

Realizar una verificación universal externa es complicado. Se requiere una sincronización perfecta entre el generador de patrones y el analizador de respuestas. En los trabajos [7] [8] se consigue esto añadiendo hardware adicional al PCB. Esta posibilidad se descarta en este trabajo, ya que se pretende llevar a cabo el test sin introducir modificaciones a nivel de hardware. Por lo tanto, la alternativa más sencilla es incluir el subsistema de verificación en el interior de la FPGA. Esto exige un analizador de respuestas liviano, que no introduzca excesiva intrusividad en las mediciones de bits críticos.

En las aplicaciones concretas testeadas en este trabajo se han llevado a cabo tests de hasta 20000 ciclos de reloj. Esto supondría almacenar una respuesta golden de 20000 muestras de 30 bits (en el caso que el bus de salida sea de 30 bits), lo que ocuparía mucho espacio en memoria. Por lo tanto, para mantener una intrusividad baja se utiliza un esquema de verificación basado en firmas.

Una firma es una manera de comprimir el stream de datos de salida. Las 20000 palabras de datos de salida se comprimen en una única palabra, por lo que el ahorro de memoria es evidente. Esta palabra ha de cambiar de valor en caso de que

un solo bit del stream de salida altere su valor. Obviamente, no es una compresión como tal, ya que no es posible reconstruir el stream completo partiendo de dicha firma. Es algo muy parecido a una función CRC, un mecanismo habitual para confirmar la integridad de un archivo o paquete. Y es ese precisamente el objetivo de la verificación basada en firmas, confirmar que no hay cambios en el stream de salida a fin de identificar si el error introducido afecta al servicio.

La implementación particular que ha sido llevada a cabo se presenta en la figura 6.6, y ocupa 49 LUTs y 44 flip-flops, lo que indica un grado de intrusividad muy bajo. La base del módulo es un sumador cíclico, que suma el valor del stream de salida y lo acumula en un registro, que se realimenta mediante una red de permutaciones e inversiones de bits. Por otro lado, un contador lleva la cuenta del número de ciclos de reloj que dura el test, y cuando llega al final envía al sumador cíclico la orden para que se detenga. En ese momento, la firma es enviada a través del EMIO (ver sección 1.5) al PS. El valor de firma se compara con el golden, y si hay discrepancia se considera que el bit de configuración modificado es crítico. En esta implementación concreta se ha llevado a cabo desde el EMIO, pero cabe destacar que también se puede hacer desde otras interfaces PS-PL, como por ejemplo, el AXI.

Este mismo circuito se emplea para la obtención de la secuencia golden. Para ello se ejecuta el test en ausencia de inyecciones, y el valor transferido desde el registro al PS por el EMIO va a ser la secuencia dorada que se va a comparar con la firma generada por cada test.

Para que el test se efectúe correctamente, es vital una sincronización perfecta entre el contador que determina el final del test y el generador de patrones. Ambos módulos se inician al mismo tiempo desde el subsistema de inyección.

6.3.3 Recuperación del estado inicial

Al concluir el proceso de verificación, los elementos secuenciales del sistema almacenan unos valores diferentes de los que tenían en el momento de inicio del test. Es bien sabido que las salidas de un sistema dependen de las entradas y del estado, por lo que un patrón de entrada idéntico no garantiza una salida idéntica si los flip-flops y memorias empiezan con valores iniciales diferentes. Por lo tanto, es estrictamente necesario hacer volver a todos los elementos secuenciales al estado inicial para proseguir con inyecciones sucesivas.

En general, este punto se obvia en la mayoría de mecanismos de emulación de SEUs observados en la literatura. Por un lado, los mecanismos que hacen la verificación por duplicación [1] [2] pueden partir desde un estado inicial diferente,

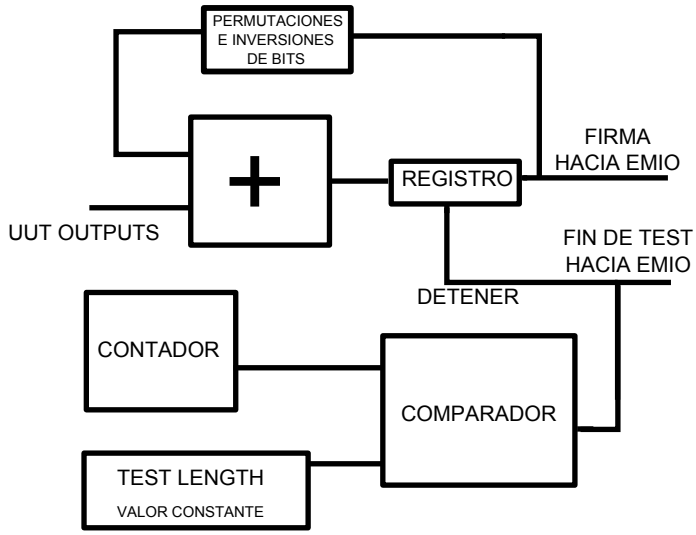


Figura 6.6: Esquema del analizador de respuestas

siempre que este sea común para ambas réplicas. Por otro lado, los métodos que utilizan verificación funcional [3] [5] [6] solo necesitan coincidir con el valor golden en el resultado final, siendo menos relevante la sucesión de estados por los que el sistema ha transitado.

La forma directa de implementar esta etapa es mediante un reset global, que tenga acceso a todos los flip-flops del sistema. De esta forma, el UUT se puede inicializar fácilmente desde el subsistema de inyección. Sin embargo, esto no puede considerarse una casuística universal, apta para cualquier escenario. En primer lugar, puede haber flip-flops cuyo reset se origine en unidades de control intermedias, y no desde el reset global. Además, no es posible inicializar el contenido de las BRAM desde el reset, lo que exige inicializar las posiciones una por una de manera individual. Y por último, hay circuitos secuenciales que ni siquiera tienen reset, como son los registros de desplazamiento implementados en LUTs, cuyo único modo razonable de inicialización es el bitstream (podrían inicializarse mediante escritura secuencial en serie, pero resulta complicado plantear una implementación de este tipo).

Está claro que pueden diseñarse estrategias de inicialización concretas dependiendo de la aplicación que está siendo testeada. Pero como el objetivo es el de un método universal de emulación de SEUs, hay que buscar una técnica de inicialización más genérica. En el método propuesto en este trabajo se decide la

inicialización desde el bitstream, ya que este fichero contiene los valores iniciales de todos los elementos secuenciales del diseño. Además, el bitstream completo se puede utilizar para hacer la inyección del bit erróneo, por lo que con una simple reconfiguración se consigue hacer dos fases de la emulación: la inyección y la inicialización.

El método propuesto utiliza bitstreams completos a fin de inicializar los elementos secuenciales. A través del PCAP se pueden llevar a cabo las inyecciones mediante bitstreams parciales sin demasiada complejidad. El problema es que no habría forma de plantear un mecanismo universal de inicialización. El hecho de utilizar una interfaz interna de alta velocidad como el PCAP permite poder hacer inyecciones de errores por bitstream completo en un tiempo razonable, algo que no sería factible utilizando el JTAG.

6.4 Limitaciones de la metodología de emulación propuesta

El esquema de inyección de fallos presentado en este trabajo tiene como objetivo determinar la tasa de fallo debido a SEUs en la memoria de configuración en SoCs que combinan FPGA y procesador. Cabe destacar que este no es el único modo de fallo de este tipo de dispositivos, aunque sí es el más relevante. En [15] se apunta que la tasa de fallo hard es de 11FIT para los dispositivos de Xilinx de 28nm, con rangos entre 6FIT y 12FIT para el resto de dispositivos de Xilinx. Este valor recoge la probabilidad de fallo abrupto debido a otros factores como altas temperaturas, ciclos térmicos, humedad y otros modos de fallo. Esta barrera de 11FIT [15] va a ser infranqueable, y ningún mecanismo de tolerancia a SEUs de los mencionados en el capítulo 4 va a permitir lograr una tasa de fallo inferior. Para bajar de ahí la única alternativa es la redundancia basada en múltiples dispositivos.

Además de los errores hard, los errores soft que afectan al procesador tampoco son cuantificables mediante este método. El sistema ARM tiene las memorias caché L1 y L2 y la OCM (On-Chip-Memory), además de los registros del sistema, que también son susceptibles a SEUs. Adicionalmente, la memoria del sistema procesador puede ser ampliada mediante DDR externa, también susceptible a SEUs, al igual que los flip-flops y registros de los controladores de entradas y salidas. La susceptibilidad a SEUs en sistemas procesadores es una disciplina ampliamente estudiada, que queda fuera de esta tesis. Como muestra de esto se encuentra el trabajo [134], que estudia la tasa de fallo debido a SEUs en el PS de la familia Zynq7000. Aquí se consideran unas tasas de fallo entre 5FIT y 100FIT

según los diferentes modos de operación.

Los SEUs pueden afectar también a los bits de los elementos secuenciales de la FPGA, tales como flip-flops, BRAMs y LUTs que funcionen como RAM distribuida o registros de desplazamiento. Estos bits siempre son críticos para el funcionamiento del sistema, al menos en los casos que contengan información útil. Su contabilización es inmediata, ya que el software de CAD para diseño en FPGA informa en sus reportes acerca de cuántos de estos bits están siendo utilizados por el diseño. En general el peso de estos elementos en la tasa de fallo va a ser bajo. De los 20 millones de bits de configuración del Zynq7020 solamente cien mil se corresponden con flip-flops, que raramente van a ocuparse todos. Sin embargo, en los diseños con mucho uso de BRAMs sí que pueden acercarse en orden de magnitud a los relativos a la CRAM.

Este esquema de emulación no es adecuado para comprobar si los bits relativos al contenido de los elementos secuenciales son críticos. Los SEUs que afectan a estos elementos no son permanentes, por lo que su criticalidad depende del instante concreto en el que sucede el error. El presente método de emulación solamente permite la inyección de errores en el instante t=0. Lo que va a determinar si un error de este tipo se detecta o no es la próxima acción llevada a cabo sobre el elemento afectado. Si el bit erróneo es leído o transmitido a otro módulo del sistema, el funcionamiento se va a ver afectado. Por el contrario, si después de la inyección se realiza una operación de escritura/inicialización sobre el bit afectado, el sistema de verificación no va a ser capaz de etiquetar ese bit como crítico.

La forma de abordar esta problemática va a ser la siguiente. La cantidad de flip-flops se va a sumar directamente a la cantidad de bits críticos medida. Aunque algunos de los bits de los flip-flops hayan sido identificados como críticos, es preferible contarlos dos veces que dejarlos sin contar. De todas formas, esto va a suponer un porcentaje pequeño respecto del total de bits críticos. Los bits del bitstream relativos al contenido de las BRAMs están claramente definidos, y no se van a hacer inyecciones sobre ellos, por lo que sumando la cantidad total de bits de BRAM utilizados se obtiene su implicación sobre el SEU.

Otra de las limitaciones es la ya comentada intrusividad del subsistema de verificación por el hecho de integrarlo en el interior de la propia FPGA. Esto va a hacer que los bits de la CRAM relativos al sistema de verificación vayan a ser considerados como críticos sin afectar realmente al servicio, mientras algunos bits de rutado y de configuración de IOBs no van a ser considerados críticos a pesar de serlo. Debido al reducido tamaño del sistema de verificación y al limitado número de entradas y salidas no se espera una distorsión en la medida superior al 1 %. Además, ambos efectos de intrusividad van en sentido contrario.

Los efectos que realmente van a limitar la precisión del sistema son el error de τ_{bit} y el coverage del patrón de entrada. Tal y como puede observarse en la tabla 3.1, los valores de FIT/Mb tienen un margen de error en torno al 20 %. La estimación de este valor depende de la caracterización de cada familia de dispositivos, y desde la emulación de SEUs no hay control sobre este valor. El valor de τ es multiplicado por el valor de bits críticos, obtenido de los test de emulación, según la ecuación 5.1. Por lo tanto, por muy preciso que sea el mecanismo de emulación, hay una barrera en la precisión del sistema fijada por el proceso de caracterización del dispositivo.

Otro factor que puede condicionar la precisión del sistema es la calidad de los patrones de entrada, que viene definida por el coverage. En este caso el error es siempre por defecto, detectando menos bits críticos de los que en realidad lo son. Tal y como se ha descrito previamente, es posible realizar una estimación de éste mediante la herramienta de simulación de potencia de Vivado, pudiendo realizar una compensación de la medición de bits críticos partiendo de ella.

6.5 Setup experimental y resultados

Las campañas de emulación han sido llevadas a cabo en la tarjeta FTZ (Fault Tolerant Zynq), que es un módulo electrónico industrial real desarrollada por SoC-e [135] para aplicaciones de Ethernet industrial (figura 6.7). El elemento central es un dispositivo Zynq7020, que controla 6 puertos Ethernet, para implementar arquitecturas Ethernet custom basadas en redundancia, principalmente en base a los protocolos HSR (Higly-available Seamless Redundancy) [136] y PRP (Parallel Redundancy Protocol) [137], y sincronización. Esta placa se elige para ilustrar que el planteamiento de emulación de SEUs propuesto, pero se puede implementar en cualquier tarjeta que contenga un SoC que combine procesador y FPGA.

Los UUTs sobre los que se han realizado los experimentos de caracterización de bits críticos son los siguientes:

- Cadena de sumadores
- Cadena de módulos CORDIC
- Diseño MicroBlaze
- SoC-e UES (Unmanaged Ethernet Switch)

Los primeros dos circuitos (cadenas de sumadores y CORDIC) son circuitos matemáticos sencillos que se pueden implementar en diferentes tamaños para llenar toda la FPGA. La figura 6.8 muestra el esquema electrónico de estas cadenas. La

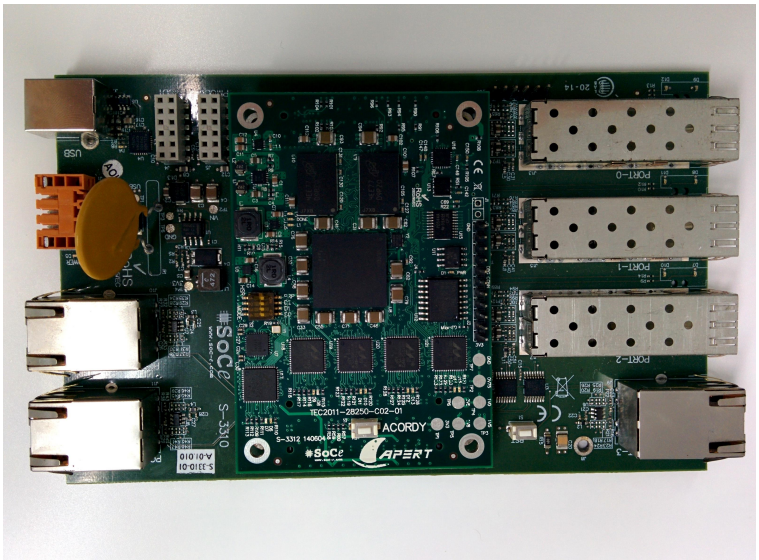


Figura 6.7: Tarjeta FTZ

cadena de sumadores se implementa utilizando datos de 28 bits, y los datos de 16 bits se utilizan para la cadena CORDIC. Estos circuitos no requieren datos de entrada específicos para generar salidas representativas. Por lo tanto, se utiliza un generador de patrones pseudoaleatorios.

El estudio de la sensibilidad a SEU de procesadores soft es un tema recurrente en la literatura. Para verificar la aplicabilidad del método a este tipo de sistemas, se ha realizado el test sobre un diseño basado en MicroBlaze ejecutando un algoritmo "Towers of Hanoi". Como patrón de entrada se genera únicamente un pulso de reset, mientras que la salida analizada es un bit que indica la correcta ejecución del programa.

Finalmente, se prueba un IPcore industrial real como caso de estudio más relevante de este trabajo. De esta manera, se demuestra que el sistema de prueba propuesto puede ser adaptado a cualquier sistema real, sin necesidad de modificaciones a nivel de hardware. Este core está compuesto por un UES (Unmanaged Ethernet Switch) de 3 puertos, cuya funcionalidad es la de enrutar cada una de las tramas de Ethernet por la interfaz de salida adecuada hacia su destino.

La generación de patrones se lleva a cabo colocando un generador de tramas Ethernet (figura 6.5) a la entrada de cada interfaz. Algunos bits que pertenecen a

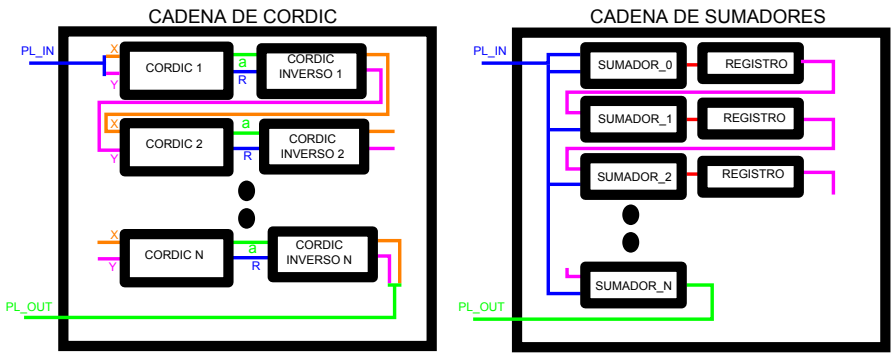


Figura 6.8: Circuitos implementados como UUT

las direcciones MAC de origen y destino se establecen como constantes, mientras que otros se asignan aleatoriamente. De esta forma, cada generador de patrones tiene un pequeño conjunto de direcciones MAC posibles.

Uno de los puntos de aplicación más relevante de los sistemas de emulación es el testeo de arquitecturas basadas en redundancia. Siendo la arquitectura TMR la más relevante en este aspecto, se han realizado campañas de inyección para implementaciones TMR de grano grueso.

Los resultados de las campañas de emulación llevadas a cabo se presentan en las tablas 6.1 y 6.2. Los valores FIT se calculan mediante la ecuación 5.1, utilizando el valor CB de la columna (CB total) y un valor de $\tau_{bit}(FIT/MB)$ de 82 FIT/Mb, que es el que se recoge en [15]. La primera de las tablas presenta la tasa de bits críticos en los recursos combinacionales y de rutado , mientras que en la segunda se aplica la compensación de los bits secuenciales y del coverage.

Como se muestra en la tabla, los valores de FIT son menores que el valor de 164 FIT estimado para todo el dispositivo con un DVF del 10 %, con la excepción del circuito de 1500 sumadores. Los resultados también muestran que la tasa de FIT no aumenta linealmente con la ocupación en todos los casos, como sí ocurre con 400 sumadores y 1500 sumadores.

También se demuestra, por medio de esta herramienta de inyección, que la tasa de fallo de diseños TMR de grano grueso debido a SEUs es mucho menor que la tasa de fallo hard del dispositivo (11FIT). Esto significa que el cuello de botella de la tasa de fallos no está en los SEU de PL si se aplica un esquema de TMR a los entornos terrestres.

Tabla 6.1: Resultados de las campañas de emulación

Diseño	Ocupación (LUTs) 53200 (100 %)	Coverage	Inyecciones	Errores	CB (combinacional + rutado)
200 sumadores	6802 (12.7 %)	100 %	453920	3137	125480
400 sumadores	12128 (22.7 %)	100 %	453920	6559	262720
1500 sumadores	42881 (80.6 %)	100 %	453920	49898	1995920
47 cordic	24963 (47 %)	100 %	453920	39718	1588720
Microblaze	1261 (2.3 %)	100 %	453920	1303	52120
SoC-e UES	6824 (12.8 %)	91.6 %	453920	2878	115120
400 sumadores TMR	33776 (63 %)	100 %	453920	26	1040
SoC-e UES TMR	17118 (32 %)	91.6 %	453920	31	1240

Tabla 6.2: Resultados de las campañas de emulación (2)

Diseño	Flip-flops	BRAM (36Kb)	CB (secuencial)	CB total	FIT
200 sumadores	5573	0	5573	131053	10.74
400 sumadores	11373	0	11373	274133	22.48
1500 sumadores	42172	0	42172	2038092	167.12
47 cordic	42090	0	42090	1630810	133.72
Microblaze	5754	2	77754	129874	10.65
SoC-e UES	7223	27	979223	1094343	89.73
400 sumadores TMR	33968	0	0	1040	0.084
SoC-e UES TMR	21671	81	0	1240	0.10

6.6 Conclusiones

Se ha demostrado la universalidad del entorno de test. Mientras la mayoría de métodos estudiados en la literatura se centran en la caracterización de una única aplicación, el presente método ha sido adaptado al testeo de sistemas diversos sin introducir modificaciones sustanciales. El único punto que requiere adaptación al cambiar de diseño es el generador de patrones.

El único requisito del método para ser aplicado es que el PCB contenga un SoC FPGA de tipo Zynq o similar. No es necesario añadir ningún tipo de modificaciones hardware a nivel de PCB. Como prueba de ello, los tests han sido llevados a cabo sobre una tarjeta industrial ya construida (Zynq FTZ), mientras que las otras posibilidades de emulación de SEUs propuestas en la literatura (UNSHADES y Hardward) requieren de una plataforma de test específica para realizar el testeo.

A pesar de incluir el subsistema de verificación en el interior de la FPGA, la intrusividad del sistema es baja. El subsistema de inyección se ubica fuera del PL, por lo que en ningún caso va a bloquearse debido a inyecciones realizadas sobre sí mismo. Las inyecciones sobre el sistema de verificación no son bloqueantes, y únicamente van a suponer un ligero aumento en la tasa de fallos medida. Las únicas inyecciones bloqueantes se deben a los IOBs (bloques de entrada y salida), bits que están delimitados en el bitstream y que basta con no realizar inyecciones sobre ellos para evitar el bloqueo.

El esquema de emulación propuesto en este trabajo es perfectamente compatible con el modelo de verificación en V. La implementación que va a ser cargada para funcionar en el entorno real es exactamente la misma que se está testeando. Esto contrasta con los métodos que proponen adaptar el diseño a un entorno de test específico para posteriormente readaptarlo al PCB que va a operar en el entorno real.

Se ha dado una solución universal a la problemática de la inicialización, que es un punto clave para la adaptabilidad sencilla a diferentes diseños. Este hecho justifica la utilización de bitstreams completos para la inyección de fallos. Para que la inyección pueda realizarse en un tiempo razonable es necesaria una interfaz de configuración de alta velocidad, algo que puede conseguirse con el PCAP, pero que difícilmente puede llevarse a cabo desde el JTAG. La utilización de la interfaz SelectMAP puede requerir modificaciones a nivel de PCB.

Entre todos los trabajos estudiados en la literatura éste es el único que diferencia las tres fases del proceso de emulación de SEUs (inyección, verificación y recuperación). La mayoría de los trabajos estudiados se centra en explicar el proceso de

inyección, dejando un esquema de verificación funcional difícilmente adaptable a otros diseños.

El sistema de emulación propuesto se ciñe a la caracterización del número de bits críticos almacenados en la memoria de configuración de la FPGA. Quedan fuera del estudio los bits relativos a elementos secuenciales (flip-flops y memorias). El presente método tampoco cuantifica la tasa de fallos soft del sistema procesador. Además de esto, hay que tener en cuenta la tasa de fallo hard del dispositivo (11 FIT), además de la tasa de fallo del resto de componentes del PCB y otros elementos del sistema.

Los factores que condicionan la precisión de la medida son la intrusividad de la memoria de configuración y el coverage del patrón de entrada. El primero de estos factores va a tener una incidencia limitada siempre y cuando la implementación del sistema de verificación ocupe pocos recursos. Para el segundo de los factores, es posible realizar una estimación del impacto mediante la herramienta de simulación de potencia de Vivado, y realizar una compensación de la medida a partir del mismo. De todas formas, la cantidad de bits críticos ha de ser multiplicada por el valor de τ, que tiene una incertidumbre en torno al 20 % y que no depende de la metodología de emulación.

El método propuesto ha sido empleado para el testeo de arquitecturas TMR. Es en este punto en donde la emulación de SEUs es la única solución posible para la estimación de bits críticos, ya que los métodos de estimación directa no son adecuados en estos casos.

Capítulo 7

Efecto de las diferentes etapas de diseño en la tasa de fallo

En este capítulo se analiza el efecto de algunas de las decisiones tomadas durante las diferentes etapas de diseño en la tasa de fallo de los circuitos implementados en FPGAs. Los tests de emulación se van a llevar a cabo mediante la plataforma presentada en el capítulo anterior. En este caso, se van a realizar leves modificaciones a un diseño concreto para determinar de qué manera afectan dichas modificaciones a la tasa de fallo. Las modificaciones se van a dividir en tres grupos: las relativas a la etapa de generación de fuentes, las relativas a síntesis y las relativas a implementación. Este capítulo está, por tanto, muy relacionado con la sección 1.3, en la que se explican las diferentes fases de diseño de un circuito implementado en FPGA.

El principal objetivo de este capítulo es demostrar que cualquier modificación en el diseño tiene un efecto en su cantidad de bits críticos. Esto está muy relacionado con el concepto de validación en V (sección 2.1.4), que no autoriza ninguna modificación en el diseño después de la validación. Una de las fortalezas de la metodología propuesta en el capítulo anterior es la compatibilidad con la validación en V, y con los resultados obtenidos en este capítulo, se va a subrayar la importancia de cumplir con dicho esquema de validación.

Estos tests van a permitir conocer qué parámetros tienen un impacto positivo en el número de bits críticos y cuales tienen un impacto negativo. Se va a comprobar

que los bits de rutado son los que mayor impacto tienen en este apartado, por encima de los bits relativos a los recursos lógicos. Se va a comprobar que una mayor ocupación a nivel de CLBs no tiene por qué significar una mayor tasa de fallo, algo que sí sucede cuando aumenta la congestión en los recursos de rutado.

7.1 Decisiones tomadas en la generación de fuentes

En esta sección se va a analizar el impacto en el número de bits críticos de algunas decisiones que se toman en el momento de generar la descripción de los circuitos. Concretamente, se van a estudiar tres parámetros: la frecuencia, la linealidad y el efecto la señal de reset. Como UUT se van a utilizar los circuitos aritméticos simples del capítulo anterior (figura 6.8). En este caso la longitud de la cadena de sumadores es de 1000, y la de CORDIC es de 47 módulos.

El primero de los experimentos consiste en implementar dichos circuitos para diferentes frecuencias. En el caso de la cadena de sumadores, la frecuencia máxima que se puede obtener sin producirse errores de implementación es de 110MHz. En la figura 7.1 se representa la implementación de este circuito para 10MHz y para 100MHz. Aquí se puede observar que a medida que la frecuencia aumenta, la implementación del diseño es más compacta. Mientras que los datos de ocupación de recursos lógicos son idénticos, en el caso de 100MHz es necesario rutar la misma cantidad de señales en menos espacio, por lo que la congestión es mayor. Esto se traduce en un aumento de la tasa de fallo a medida que se sube en frecuencia. Tal y como puede observarse en el gráfico de la figura 7.2, la tasa de bits críticos aumenta en un 47 % a 110MHz respecto de la tasa a 10MHz.

Por otro lado, la figura 7.3 ilustra la implementación de la cadena de 47 módulos CORDIC para 50 y 250 MHz, ya que por encima de 250resulta imposible cumplir los requisitos temporales. En este caso se observa que ambos dibujos son similares. El nivel de congestión es muy similar para ambas implementaciones, por lo que la cantidad de bits críticos es similar en ambos diseños. Esto puede observarse en la figura 7.2, donde la tasa de fallos es constante para la cadena de CORDIC. Aún así hay hasta un 10 % de discrepancia en la cantidad de bits críticos, y curiosamente la tasa de bits críticos es inferior para la frecuencia de 250MHz.

La interpretación que se hace de esto es la siguiente. El circuito de la cadena de sumadores tiene muchas nets que son críticas a nivel de timing, lo que exige que la celda origen y la celda destino estén lo más cerca posible. Esto obliga a la herramienta de emplazamiento y rutado a compactar mucho los recursos, hacien-

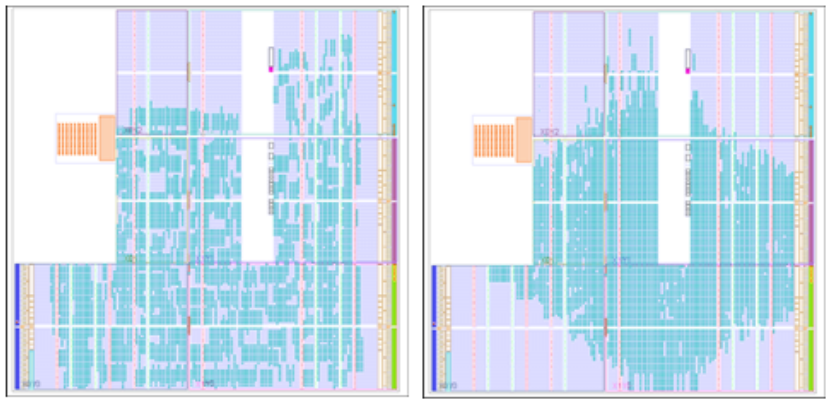
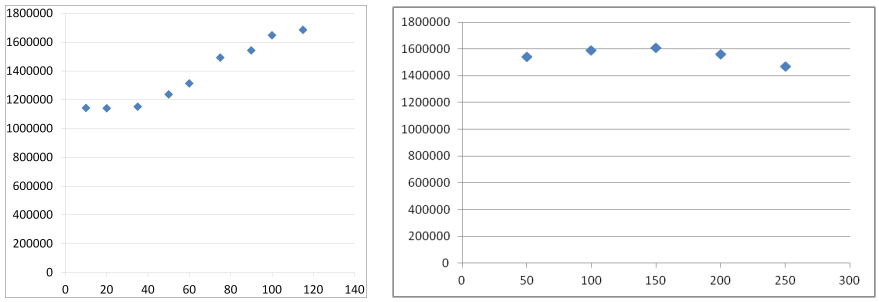


Figura 7.1: Implementación de la cadena de 1000 sumadores, izquierada: 10MHz derecha: 100MHz



(a) Cadena de 1000 sumadores (b) Cadena de 47 CORDIC

Figura 7.2: Bits críticos en función de la frecuencia(Eje X, frecuencia (MHz), eje Y número de bits críticos)

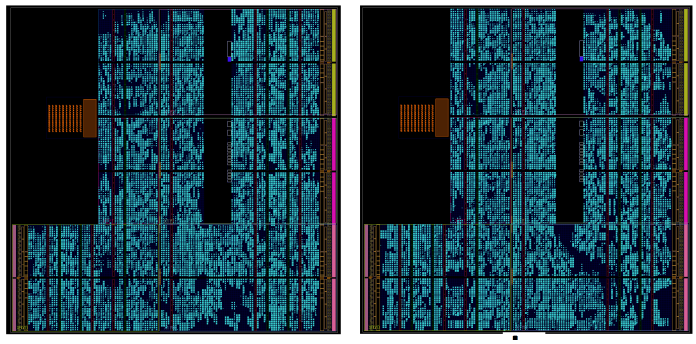
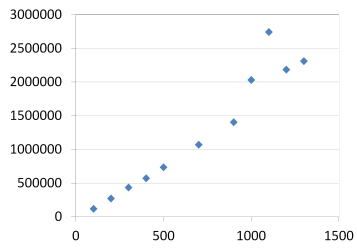


Figura 7.3: Implementación de la cadena de 47 CORDIC, izquierda: 50MHz, derecha: 250MHz

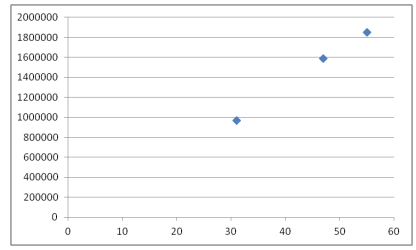
do crecer la congestión. Por otro lado, en la cadena de CORDIC no hay muchas nets que den problemas de timing, por lo que la herramienta software tiende a apelmazar dichas nets problemáticas, mientras el grueso del diseño permanece intacto.

El segundo de los experimentos consiste en implementar circuitos de diferentes tamaños para verificar si la cantidad de bits críticos aumenta linealmente con el tamaño del circuito. Para ello se ha implementado la cadena de sumadores a 100MHz para diferente número de etapas de suma, manteniendo constante el ancho de bus. Estos resultados se presentan en la figura 7.4. Aquí se puede observar que para tamaños pequeños el incremento sí que es más o menos lineal. Sin embargo, cuando el índice de ocupación empieza a ser grande, el efecto de la congestión se hace más evidente. Esto hace que la tasa de bits críticos se pueda volver impredecible. Aún así, la tasa de fallos es siempre superior al comportamiento lineal que se aprecia para tamaños pequeños.

Resulta especialmente llamativo que la cantidad de bits críticos es menor para una cadena de 1200 sumadores que para una de 1100. De hecho, el caso concreto de los 1100 sumadores queda especialmente fuera del comportamiento lineal. Esto se debe a que el proceso de emplazamiento y rutado tiene un componente impredecible debido a la naturaleza iterativa del algoritmo. En este caso, la herramienta ha conseguido una implementación válida para las exigencias de tiempos, y no considera que deba optimizarse. Por otro lado, para el caso de 1200 sumadores se consigue una implementación más optimizada desde el punto de vista de la congestión. El experimento de linealidad también se ha hecho para la cadena de CORDIC, pero como en este caso no hay problemas de congestión la linealidad es prácticamente perfecta.



(a) Cadena de n sumadores



(b) Cadena de n CORDIC

Figura 7.4: Bits críticos en función de la longitud de cadena (Eje X, longitud de cadena, eje Y número de bits críticos)

En el experimento final de esta sección se pretende estimar el efecto de la señal de reset en la tasa de fallo global. En ocasiones esta señal se incluye de forma automática, sin valorar adecuadamente si es estrictamente necesaria o no. En este caso se ha implementado una cadena de 1100 sumadores, que contiene 1100 registros de 28 bits. Por lo tanto, la inclusión del reset global obliga a rutar dicha señal a 30800 flip-flops. Con señal de reset el diseño tiene 1494480 bits críticos, mientras que sin ella la tasa de bits críticos es de 1397160, lo que equivale a un aumento de la tasa de fallo del 7 %. Enlazando este apartado con el capítulo anterior, este test no podría haberse realizado de no utilizarse un mecanismo universal para la inicialización de los flip-flops.

7.2 Parámetros a nivel de síntesis

En este nivel se va a realizar el test sobre dos parámetros, MAX_FANOUT y LUT_COMBINING. El primero de ellos indica el máximo fanout de una señal concreta. Es decir, la cantidad máxima de destinos a los que una señal concreta puede ser rutada. La herramienta de síntesis lleva a cabo esta acción mediante duplicación de hardware. De esta forma, no habrá una única fuente que haga de driver hacia múltiples destinos, sino que serán múltiples fuentes las que se encarguen de este cometido. Esto tiene un coste a nivel de ocupación, debido al incremento de CLBs necesarias para la implementación.

La cadena de 1000 sumadores de 28 bits ha sido testeada con valores de MAX_FANOUT = 25 y MAX_FANOUT = 10000, a una frecuencia de 100MHz. En el primero de los casos la cantidad de bits críticos es 1672502, y para la segunda 2031056, lo que indica un incremento del 21 %. Al igual que en el caso

anterior, un incremento en la cantidad de recursos utilizada (el incremento es mínimo para este caso concreto, inferior al 1 %) supone una reducción de la tasa de fallo. La obligación de llegar a múltiples destinos desde una única fuente exige ubicar los recursos muy próximos entre sí, haciendo crecer la congestión en esa zona.

Este mismo test ha sido llevado a cabo para la cadena de 47 módulos CORDIC a 150 MHz. Los valores de MAX_FANOUT escogidos han sido 25 y 10000, al igual que en el caso anterior. Sin embargo, en este caso el resultado es el contrario. Para MAX_FANOUT = 25 la cantidad de bits críticos es 1780160, y para MAX_FANOUT = 10000 es de 1561920, lo que supone una reducción del 14 %. En este caso no hay señales con un fanout extremadamente elevado, por lo que no se consigue una reducción significativa de la congestión mediante el hardware extra. El valor MAX_FANOUT = 25 es exageradamente pequeño, por lo que se añade bastante hardware redundante que no contribuye a la reducción de la congestión. Este es un ejemplo que indica que el mismo parámetro hace reducir la tasa de fallo para un circuito concreto y hace aumentarla para otro.

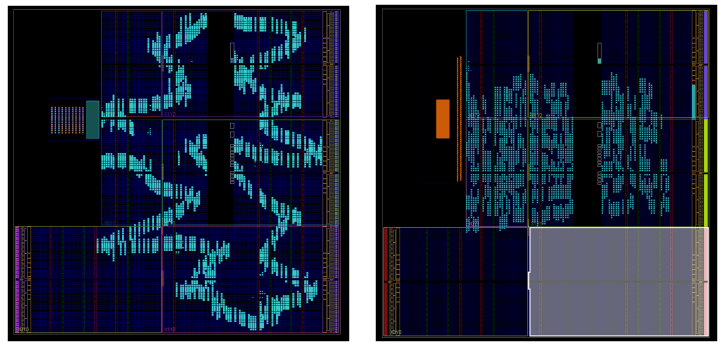
Por otro lado, unos tests similares han sido practicados para el estudio del parámetro LUT_COMBINING. Cuando éste tiene un valor de "true", se permite combinar en una LUT concreta 2 funciones lógicas de 3 o menos entradas para ahorrar espacio. De esta forma, se consigue un descenso en el número total de LUTs a cambio de aumentar la cantidad de señales que requieren ser rutadas en una zona concreta. Es decir, disminuir la cantidad de LUTs ocupadas a cambio de aumentar la congestión.

Los resultados obtenidos en este caso se asemejan a los del parámetro anterior. Con LUT_COMBINING activado se consigue una reducción de los slices ocupados del 2 %. Sin embargo, esto no significa una cantidad de bits críticos inferior, ya que hay más señales que han de ser rutadas en menos espacio. Debido a esto aumenta la congestión, y por consiguiente, también aumentan los bits críticos relativos a recursos de rutado. En este caso, para la cadena de sumadores de 28 bits este parámetro no tiene ningún efecto, ya que por su naturaleza la herramienta de síntesis no considera adecuado realizar implementaciones de este tipo. Sin embargo, si se cambia la anchura de bit de los sumandos a 6 bits sí que se hacen implementaciones de este tipo.

Para una cadena de 1500 sumadores de 6 bits a 100MHz la cantidad de bits críticos es de 477434 con LUT_COMBINING activo y 444176 para cuando no está activado. Por otro lado, para la cadena de 47 módulos CORDIC a 150MHz es de 1588720 cuando está activado y 1513200 cuando no lo está. En este caso el incremento de LUTs empleadas es de un 50 % cuando este parámetro se deshabilita.

7.3 Parámetros a nivel de Implementación

En esta sección se va a analizar cómo afectan las medidas y parámetros a nivel de implementación en la cantidad de bits críticos de un diseño concreto. Lo primero en ser testeado es el algoritmo de implementación. La cadena de sumadores de 28 bits ha sido implementada mediante el Vivado 2015.3 y el PlanAhead 14.7, que es la herramienta previa para diseño con FPGAs de Xilinx. En la figura 7.5 se presentan estas dos implementaciones para cadenas de 400 bits de longitud, y se aprecia claramente que ambas son muy diferentes. La ocupación de ambos diseños es idéntica, por lo que la cantidad de bits críticos en recursos lógicos y CLBs va a ser la misma. Sin embargo, al realizarse el emplazamiento mediante un algoritmo diferente, los elementos han sido colocados en ubicaciones diferentes. Por lo tanto, la interconexión de dichos elementos mediante los recursos de rutado va a ser llevada a cabo por líneas diferentes, haciendo así variar la cantidad de PIPs y matrices de interconexión que forman parte del diseño.



(a) PlanAhead (b) Vivado

Figura 7.5: Implementaciones de una cadena de 400 sumadores en PA y Vivado)

Los resultados obtenidos en este experimento son sorprendentes. En principio, cabría esperar que la herramienta más moderna realizara una implementación más eficiente al estar más depurado el algoritmo. Esto es así para el caso de la cadena de 400 sumadores a 100MHz, con 800352 bits críticos para el caso del PlanAhead y 571920 para el Vivado. Se aprecia que la discrepancia es bastante elevada (40 %). Sin embargo, para el caso de una cadena de 1300 sumadores sucede a la inversa. La cantidad de bits críticos detectada es de 1655325 para PlanAhead y de 2310571 para Vivado, lo que supone una discrepancia del 39 % en este caso favorable al PlanAhead. La conclusión que se extrae de esto es que

el software Vivado realiza implementaciones poco eficientes para cadenas de sumadores grandes a frecuencias altas. Se concluye también que la cantidad de bits críticos es muy dependiente del algoritmo empleado para la implementación, que tiene un efecto impredecible en la medida. Esta imprevisibilidad tiene mayor relevancia cuanto más cerca se está de los límites del dispositivo a nivel de ocupación y/o frecuencia.

La herramienta de implementación de Vivado proporciona a los desarrolladores la posibilidad de optimizar el emplazamiento y el rutado según una serie de parámetros, siendo los más relevantes el área, la congestión, la frecuencia y el tiempo de ejecución. Se ha testeado el impacto de estos parámetros de optimización en la cantidad de bits críticos. Sin embargo, en la mayoría de los casos testeados estos parámetros no tienen ningún efecto en la cantidad de bits críticos, con discrepancias inferiores al 1 % en todos estos casos para los diferentes circuitos analizados. Tan solo en el caso del parámetro Spread_LogicSSL para la cadena de 1300 sumadores a 100MHz se consigue una mejora del 32 %. La implementación que la herramienta Vivado hace para la cadena de 1100 sumadores a 100MHz es el punto más extremo de la figura 7.4, que tiene un nivel de congestión fuera de lo normal. Este parámetro intenta dispersar la lógica por toda la FPGA en la medida de lo posible, lo que contribuye positivamente a reducir el nivel de congestión. En este caso, de 2718066 bits críticos se pasa a 2052248.

El último factor a analizar es el floorplanning, que consiste en forzar a la herramienta de emplazamiento a ubicar los recursos lógicos en una zona concreta de la FPGA (ver sección 1.3.3), que se denomina como pblock. Esto obliga a agrupar mucho los recursos lógicos, especialmente si se fuerza a ubicar una gran cantidad de lógica en un pblock pequeño. Debido a esto, se restringen las opciones de la herramienta de rutado para conectar los recursos lógicos entre sí, teniendo que rutar muchas señales en poco espacio. De esta forma, la inclusión de pblocks en el diseño provoca un aumento en la congestión del sistema, con el consiguiente incremento de la cantidad de bits críticos.

La relevancia del estudio del floorplaning radica en el hecho de que algunos métodos de emulación de la literatura [3–5] proponen ubicar el UUT en un pblock. Además de los problemas mencionados en los dos capítulos anteriores de la intrusividad del subsistema de inyección, cabe destacar que la inclusión de pblocks produce aumentos de la tasa de fallo, algo que no es positivo en sistemas orientados a confiabilidad. Los esquemas de emulación pueden plantear eliminar el pblock después de realizar el test. Sin embargo, no es conveniente modificar la implementación del diseño una vez realizado el test. Esto se ha venido reforzando durante el presente capítulo, debido a la incertidumbre introducida por el proceso de emplazamiento y rutado. Aún y todo, la eliminación del pblock debería tener

siempre un impacto positivo en la cantidad de bits críticos del sistema.

Para demostrar esto, se ha implementado una cadena de 1200 sumadores con y sin pblock, tal y como se muestra en la figura 7.6 (con pblock) y en la figura 7.7. A la izquierda se presenta la distribución de los recursos utilizados a lo largo de la FPGA. A la derecha se muestra el gráfico de congestión, que indica la cantidad de señales que cruzan por cada región del espacio. La congestión baja se representa en azul, la media en amarillo, y la congestión elevada en morado y lila. Se aprecia claramente que la congestión es mayor para el diseño con pblock. Sin pblock la cantidad de bits críticos es de 2183334, y con pblock es de 3204024. Esto supone un incremento del 46 %.

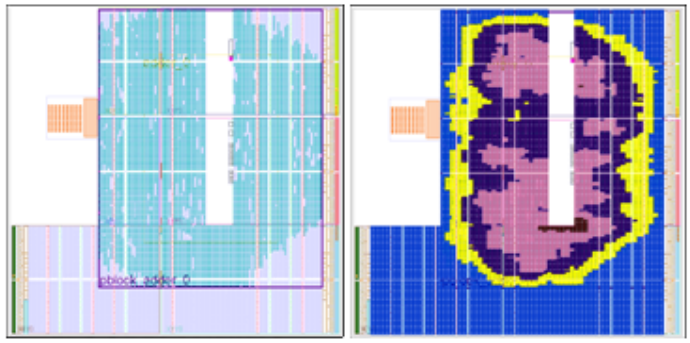


Figura 7.6: Congestión de la cadena de sumadores con pblock

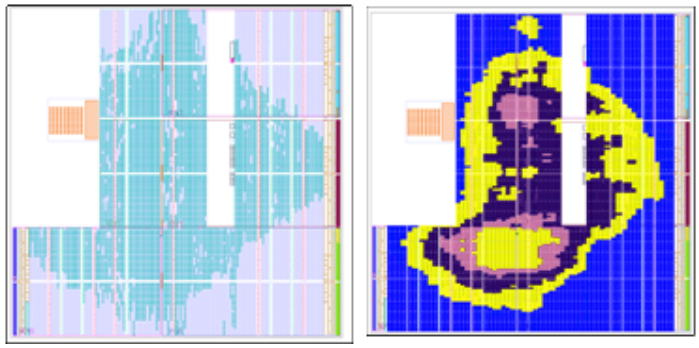


Figura 7.7: Congestión de la cadena de sumadores sin pblock

Sin embargo, existe un caso en el que la utilización de pblocks consigue reducir la cantidad de bits críticos de un sistema. Se trata de las configuraciones tolerantes a

fallos basadas en redundancia, tales como DWC (Duplication With Comparison) o TMR (Triple Modular Redundancy) (ver capítulo 4). En estos casos, el hecho de aislar cada una de las réplicas en un pblock consigue reducir significativamente los errores entre dominios (ver sección 4.2.1). Esto se ha comprobado mediante el test de una cadena de 400 sumadores en configuración TMR. En la figura 7.8 se aprecian ambas implementaciones, con y sin pblock. En estas figuras, a cada dominio TMR se le ha asignado un color (rojo, amarillo y verde) En la figura se aprecia claramente que la frontera entre dominios disminuye considerablemente cuando se utiliza floorlpanning. El diseño con pblocks tiene 19669 bits críticos, y 112350 sin pblocks, es decir, un dato casi 6 veces peor.

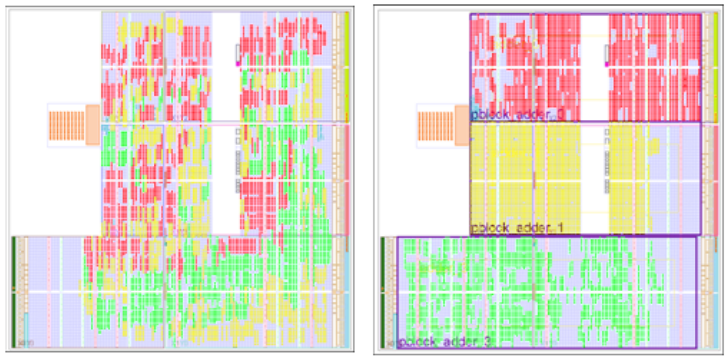


Figura 7.8: Implementaciones TMR sin floorplanning (izquierda) y con floorplanning (derecha)

7.4 Conclusiones

Mediante los tests y pruebas realizadas, se ha demostrado que modificaciones de diseño pequeñas pueden provocar consecuencias importantes en la cantidad de bits críticos del sistema. Esto fortalece la necesidad de una aplicación rigurosa del esquema de validación en V, ya que modificaciones en el diseño posteriores a los tests de validación invalidan los resultados de dichos tests.

Una idea que también se refuerza en este capítulo es el hecho de que una mayor ocupación a nivel de recursos lógicos no implica una mayor cantidad de bits críticos. Como ya ha sido comentado a lo largo de esta tesis, la mayoría de los bits de configuración se corresponden con recursos de rutado. Por lo tanto, va a ser el nivel de congestión en el rutado el que condicione en gran medida la tasa de bits críticos del sistema. Cuantas más señales haya que rutar a través de un espacio definido, más crítica va a ser la influencia de los transistores de rutado. Los transistores que están en corte van a tener una mayor probabilidad de conectar dos señales que transcurran por cables adyacentes. De igual manera, al ser el rutado más complejo, las señales van a tener que cruzar por más puntos de interconexión para alcanzar su destino, por lo que va a haber más transistores en conducción que provoquen fallos cuando su valor de puerta cambie.

Las acciones que reduzcan flexibilidad a la herramienta de rutado van a ser las que, por lo general, hagan crecer la cantidad de bits críticos. Como ejemplo de esto está el incremento en frecuencia, que limita la distancia a la que se pueden ubicar los recursos lógicos y la cantidad de PIPs que una señal puede cruzar. De igual modo, cuando la ocupación del sistema alcanza valores elevados, la congestión aumenta irremediablemente, ya que resulta imposible dispersar la lógica y va a haber muchas señales a ser rutadas a través de unos recursos limitados. Algo similar sucede con el floorplanning, especialmente cuando la ocupación del pblock sea elevada. La utilización del parámetro LUT_COMBINING también incrementa este efecto.

Por otro lado, las acciones que permitan dispersar la lógica a lo largo del dispositivo van a reducir la tasa de fallo. Ejemplo de esto es el parámetro Spread_logicSSL. También obligar a la herramienta de síntesis a reducir el fanout duplicando el hardware consigue un efecto en el mismo sentido.

Otras acciones pueden tener consecuencias imprevisibles. Un ejemplo de esto es el cambio en el algoritmo de implementación. Cuando se cambie de software de diseño o se actualice una versión es conveniente repetir los tests de emulación, ya que esto puede tener un impacto relativamente grande.

En cuanto a arquitectuas tolerantes a fallos basadas en redundancia como la configuración TMR o similares, se ha justificado la importancia del floorplanning para prevenir los errores entre dominios.

Capítulo 8

Análisis de la tasa de fallo en presencia de MCUs

Debido a la reducción continua del tamaño de los transistores en dispositivos electrónicos, cada vez los voltajes de alimentación son más reducidos, lo que implica que la carga almacenada en cada celda de memoria sea más pequeña. Por otro lado, la cantidad de pares e-h generados por el impacto de una partícula de radiación concreta en el silicio se mantiene constante. Esto tiene dos consecuencias: por un lado, es más probable que se rebase el umbral de carga del biestable como consecuencia de un evento, y por otro, se incrementa la posibilidad de que una única partícula afecte a dos celdas diferentes, lo que se conoce como MCU (Multiple Cell Upset). Las partículas más susceptibles de provocar MCUs son los iones pesados (heavy ions), ya que tienen un LET muy elevado que se traduce en una gran cantidad de energía transferida al semiconductor.

Los fabricantes de semiconductores llevan tiempo proponiendo diferentes avances a nivel microelectrónico para mitigar el impacto de la reducción de la carga acumulada en las celdas de memoria. Por ejemplo, la tecnología FinFET SRAM está consiguiendo unos resultados prometedores en SEU y MCU. Sin embargo, la reducción del tamaño de los transistores es una tendencia incuestionable en las últimas décadas, que va a continuar en el futuro con total seguridad, siguiendo probablemente la ley de Moore. Debido a esto, la influencia de los MCUs puede volverse más relevante para las próximas generaciones de semiconductores.

En este capítulo se van a discutir dos cuestiones relativas a los MCUs. Por un lado, se va a plantear la emulación de eventos múltiples desde la modificación

del bitstream. Es decir, discernir qué bits han de ser modificados de manera simultanea para replicar en la medida de lo posible el efecto de un MCU. Además, se discute en profundidad cómo varía la tasa de fallo cuando se tienen en cuenta los MCUs.

A la hora de analizar las herramientas de emulación de SEU estudiadas en la literatura, se ha comprobado que éstas no tienen en consideración la inyección de MCUs. Sin embargo, esto no quiere decir que los errores múltiples no se contabilicen, sino que se tratan como si fueran múltiples errores simples. Es decir, como si fueran SBUs (Single Bit Upsets) independientes e incorrelados entre sí. La diferencia fundamental es que los MCUs siempre se producen en bits físicamente adyacentes, mientras que múltiples SBUs incorrelados quedan dispersos por todo el dispositivo y no tienen por qué afectar a celdas de memoria con relación de proximidad. Y es éste precisamente uno de los objetivos principales de este capítulo, estudiar las diferencias entre ambas maneras de abordar la presencia de MCUs.

El capítulo comienza con un análisis matemático de ambos enfoques. Como punto de partida se toman las ecuaciones presentadas durante los capítulos anteriores, y se desarrollan hasta llegar a una expresión de la tasa de fallo para ambos casos. Posteriormente, se llevan a cabo campañas de emulación, basándose en un estudio previo [16] en el que se analizan las formas de los MCUs en FPGAs de la serie7 de Xilinx, para finalmente presentar hipótesis acerca de las causas de las discrepancias entre ambos modelos.

8.1 SBU independientes y emulación de MCUs

Tal y como se ha explicado en el capítulo 3, un SEU (Single Event Upset) va a provocar en la mayoría de ocasiones un error de un bit, lo que se conoce como SBU (Single Bit Upset). Sin embargo, en un número reducido de ocasiones, dicho evento va a provocar errores en más de un bit. Este fenómeno se denomina MCU (Multiple Cell Upset), y sucede en casos de eventos excepcionales, provocados por partículas de muy alta energía. Otro concepto análogo es el MBU (Multiple Bit Upset), que se corresponde con un evento que afecta a múltiples bits pertenecientes a la misma palabra de memoria [138], entendiendo por palabra la unidad mínima protegida por un código de detección y corrección de errores (EDAC). Un MBU es caso de MCU, pero mucho más severo, ya que errores múltiples en una palabra pueden invalidar el código EDAC (Error Detection And Correction), provocando la pérdida de información irreparable.

Existen múltiples trabajos que miden experimentalmente las secciones de cruce,

la probabilidad y el tamaño de los MCUs en las memorias de configuración de FPGAs de tipo SRAM. En [139], se presenta una metodología para medir las secciones de cruce de protones e iones pesados para MCUs en FPGAs de Xilinx, donde se analiza la familia Virtex4 (90nm). Aquí, el 1 % - 3 % de los eventos producidos por protones de 63,3 MeV son MCU, y para iones pesados de LET elevadas los MCUs pueden alcanzar el 35 %. En [140], se analiza la familia Virtex5 y los MCUs son entre el 6-10 % de los SEUs para este mismo tipo de protones y aproximadamente 60 % de MCUs para iones pesados. Se obtienen datos similares para un dispositivo Spartan3 en [141].

En [16], los ratios de MCU/SBU se analizan para las FPGA de la serie 7 de Xilinx (28nm). En este caso, la tasa de MCUs para partículas de energías medianas es similar a los casos anteriores (1-3 %). Sin embargo, para iones de alta energía la tasa de MCUs no supera el 36 %. Esto se debe a que esta familia de dispositivos incorpora la tecnología FinFET de TSMC, que consigue reducir la cantidad de MCUs, según se refleja en [73, 142].

Para mitigar la posibilidad de que un MCU afecte a bits de la misma palabra de memoria, las FPGAs implementan estrategias de entrelazamiento o interleaving. Éstas consisten en hacer que bits físicamente adyacentes pertenezcan a diferentes frames. Como los errores múltiples afectan a bits físicamente contiguos, disminuye la probabilidad de que un único evento afecte a más de un bit de una frame concreta. De esta forma, se consigue incrementar la eficacia de los códigos EDAC, que protegen las diferentes frames almacenadas en la CRAM del dispositivo. Una posible representación del concepto de interleaving se presenta en la figura 8.1. Aquí se aprecia que los bits de diferentes frames aparecen mezclados entre sí, de modo que si un MCU afecta a bits de frames distintas, se preserva la eficacia del esquema EDAC implementado.

Sin embargo, esto plantea problemas a la hora de realizar la emulación de MCUs, ya que introducir modificaciones en bits contiguos del bitstream no se corresponde con emulaciones de SEUs en bits adyacentes físicamente. Para conocer qué relación lógica existe entre dos bits físicamente adyacentes la única vía es la ingeniería inversa, que es precisamente lo que se lleva a cabo en el trabajo [16].

En este estudio se hacen incidir sobre una FPGA de la serie7 haces de iones de diferentes energías (N, Ne, Si, Ar, Cu, Kr, Xe), y como conclusión se presentan las geometrías en las que se dan los eventos múltiples con sus respectivas probabilidades. Estos resultados se representan en la tabla 8.1. Las formas geométricas de dicha tabla representan la relación lógica entre los bits afectados por un evento múltiple. El eje Y indica el número de frame, mientras que el eje X representa el bit afectado dentro de una frame concreta. De esta forma, la primera fila hace referencia a la cantidad de SEUs que afectan a un único bit. De igual manera,

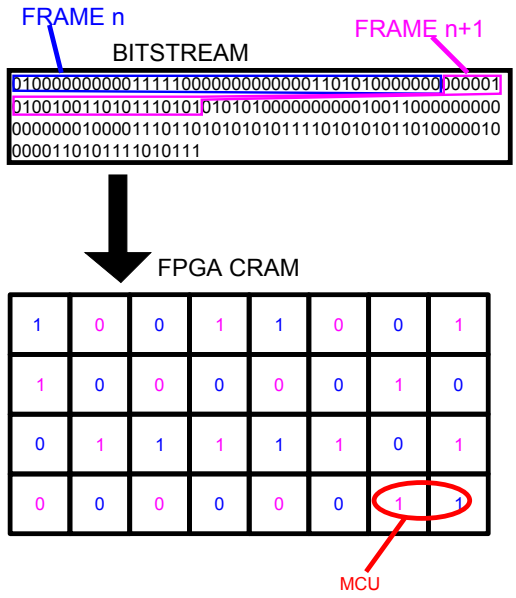


Figura 8.1: Entrelazamiento entre frames o interleaving

Tabla 8.1: Probabilidad de aparición de MCUs de una forma concreta [16]

		$Ion(LET)\frac{MeV\,cm^2}{mg}$						
	Shape	N (1.16)	Ne (2.39)	Si (4.35)	Ar (7.27)	Cu (16.5)	Kr (25.0)	Xe (49.3)
P_{SBU}		98,7	93,1	90,1	84	76,9	78,5	64
P_{21MCU}		0,2	0,9	1,4	2,5	6,4	5	15,1
P_{22MCU}		0,7	4,5	5,2	7,6	6,5	7,6	4,4
P_{23MCU}		0,1	0,9	1	2,6	4,1	3,9	2,7
P_{24MCU}		0,2	1,2	1,1	1,1	1	1	0,8
P_{3MCU}		0	0,6	0,6	0,7	2	1,2	4,1
P_{41MCU}		0	0,1	0,1	0,4	0,5	0,7	1,8
P_{42MCU}		0	0	0	0,2	0,5	0,4	0,6
P_{43MCU}		0	0	0,1	0,1	0,2	0,2	0,6

la segunda fila se refiere a MCUs que afectan a dos bits contiguos de una frame concreta, mientras que la cuarta fila indica la probabilidad de MCUs que afectan a la misma posición de bits de frames contiguas.

En este capítulo se plantea la emulación de MCUs aplicando los datos de dicha tabla. Para ello, se llevan a cabo las inyecciones de errores mediante la modificación de múltiples bits del bitstream siguiendo las formas indicadas en la tabla. El resto del proceso de emulación se mantiene igual. Una vez modificado el bitstream de acuerdo a las formas expuestas en [16], el dispositivo se configura con el bitstream corrompido, se lleva a cabo la verificación y se retorna al estado inicial.

El objetivo final es el de evaluar el impacto que tiene sobre la tasa de fallo del sistema el hecho de tener en cuenta la existencia de MCUs en el proceso de emulación. Para ello, se realiza una comparación entre este planteamiento de emulación de SEUs y la emulación consistente en considerar todos los SEUs como SBUs.

Considerar únicamente la existencia de SBUs no quiere decir que no se cuantifique el impacto de los SEUs que afecten a más de un bit, sino que los eventos múltiples se consideran como una sucesión de eventos únicos (SBUs) independientes e incorrelados entre sí.

8.2 Formulación matemática de ambos modelos

El punto de partida para la descripción de ambos modelos son las ecuaciones 8.1 y 8.2. Estas expresiones, que ya han sido introducidas previamente, permiten el cálculo de la tasa de fallo (FR) en función del entorno de radiación, el tipo de dispositivo y el diseño implementado. Los parámetros que entran en juego son los siguientes.

- El flujo de radiación (ϕ), que depende del entorno físico de radiación en el que va a trabajar el dispositivo. Indica la cantidad de partículas que atraviesan una superficie determinada en un tiempo concreto. El flujo es una función de la energía (LET).
- La sección de cruce (σ), que viene identificada por la tecnología de la familia de FPGA empleada. Representa la probabilidad de ocurrencia de un evento para un flujo concreto. Es dependiente de la energía de la partícula (LET), ya que, a mayor energía, mayor probabilidad de evento potencialmente nocivo.
- El DVF (Design Vulnerability Factor), que indica el porcentaje de bits de

la memoria de configuración que son críticos para el diseño, y por tanto, es dependiente del diseño implementado en la FPGA.

$$FR = \tau_{SEU} * DVF \tag{8.1}$$

$$\tau_{SEU} = \int_{LET} \sigma(LET) * \phi(LET) * dLET \tag{8.2}$$

La clave para el análisis llevado a cabo en el presente capítulo radica en la ecuación 8.2. Esta ecuación permite la obtención de el parámetro τ_{SEU} (tasa de eventos), que indica la cantidad de eventos (SEUs) que afectan al dispositivo por unidad de tiempo. En este punto es en el que se plantea la bifurcación de los dos modelos analizados en este capítulo.

Por un lado, si no se considera la existencia de MCUs, todos los SEUs van a corresponderse con SBUs, es decir, van a afectar solamente a un único bit de memoria. Aquí se plantea una redefinición del parámetro sección de cruce (σ). Este parámetro representa la cantidad de SEUs por unidad de tiempo cuando el dispositivo está afectado por un flujo de radiación de 1 partícula/$m^2 * s$. Es decir, cantidad de eventos por unidad de flujo y tiempo.

Para el primero de los planteamientos (todos los SEUs se consideran SBUs), el parámetro (σ) no indica exactamente la cantidad de eventos por unidad de flujo y tiempo, sino que se aporta una medida de la cantidad de bits afectados por SEUs por unidad de tiempo y flujo. Tal y como se ha comentado en repetidas ocasiones, un evento puede provocar modificaciones en más de un bit, por lo que resulta evidente que el número de eventos es diferente del número de bits modificados.

En aras de distinguir ambas secciones de cruce, se definen (σ_{SEU}) y (σ_{error}). La primera va a dar información del número de eventos por unidad de flujo y tiempo, mientras que la segunda indica la cantidad de errores (bits con modificaciones) por unidad de tiempo y flujo. De estas definiciones se deducen las ecuaciones 8.3 y 8.4. Cabe reseñar que estos dos parámetros (σ_{SEU} y σ_{error}) son dependientes de la energía o el LET de las partículas incidentes, ya que cuanto más energéticas sean dichas partículas mayor es la probabilidad de que un evento cause errores múltiples, lo que incrementa la diferencia entre la cantidad de eventos y la cantidad de errores.

$$\tau_{SEU} = \int_{LET} \sigma_{SEU}(LET) * \phi(LET) * dLET \tag{8.3}$$

$$\tau_{error} = \int_{LET} \sigma_{error}(LET) * \phi(LET) * dLET \tag{8.4}$$

El parámetro que relaciona ambas expresiones es el número medio de errores por evento, que en este texto se va a definir como r(LET) 8.5. Este parámetro se obtiene partiendo de las probabilidades de ocurrencia de SBUs y MCUs de 2, 3 ó 4 bits respectivamente (ecuación 8.6), que a su vez se extraen de la tabla 8.1. En esta tabla se puede observar cómo las probabilidades de ocurrencia de cada forma son dependientes del LET, ya que con la energía aumenta la probabilidad de ocurrencia de eventos múltiples. P_{SBU} viene dada por la primera fila, que representa la probabilidad de que un evento concreto sea un SBU para diferentes LETs. Por otro lado, P_{2MCU} se obtiene sumando las probabilidades de aparición de las formas correspondientes a MCUs de 2 bits (filas 2 a 5). De igual manera, P_3MCU es la fila 6 y P_{4MCU} es el sumatorio de las filas 7 a 10.

$$\sigma_{event}(LET) = \sigma_{error}(LET) * r(LET) \tag{8.5}$$

$$r(LET) = P_{SBU} + 2 * P_{2MCU} + 3 * P_{3MCU} + \cdots = P_{SBU} + \sum_{n} n * P_{nMCU} \tag{8.6}$$

Finalmente, para conocer la tasa de fallo del sistema se precisa la utilización del parámetro conocido como DVF (Design Vulnerability Factor), así como la ecuación 8.1. Este parámetro indica la probabilidad de error de un diseño concreto cuando éste ha sido afectado por un SEU. Si todos los errores se consideran como SBUs, este valor va a ser una constante. Sin embargo, en el caso de que se considere la existencia de eventos múltiples, la probabilidad de que uno de éstos desencadene un fallo del sistema es mayor para energías altas, ya que la probabilidad de que dicho evento haya sido múltiple aumenta con la energía.

Por lo tanto, la expresión del cálculo de la tasa de fallo viene dada por las expresiones 8.7 y 8.8. La primera de ellas define el modelo en el que todos los errores se consideran SBUs, y no es más que la combinación directa de las ecuaciones 8.1 y 8.2. Ésta es la expresión que ha sido empleada en capítulos anteriores para la obtención de la tasa de fallo. Por otro lado, la ecuación 8.8 ilustra el caso de considerar la ocurrencia de eventos múltiples. En este caso, el parámetro DVF no puede sacarse de la integral debido a que éste es dependiente de la energía del evento.

$$FR_{SBU} = DVF_{SBU} * \int_{LET} \sigma_{error}(LET) * \phi(LET) * dLET \tag{8.7}$$

$$FR_{MCU} = \int_{LET} DVF_{MCU}(LET) * \sigma_{SEU}(LET) * \phi(LET) * dLET = \\ = \int_{LET} \frac{DVF_{MCU}(LET)}{r(LET)} * \sigma_{error}(LET) * \phi(LET) * dLET \tag{8.8}$$

Observando ambas ecuaciones se aprecia que son muy similares, exceptuando que para el primero de los casos DVF_{SBU} es constante y puede sacarse de la integral. Mientras que en el segundo, el factor de vulnerabilidad es dependiente de la energía de la partícula y aparece normalizado por el parámetro r(LET). Por lo tanto, la comparativa entre ambos modelos se centra en calcular DVF_{MCU} para diferentes valores de LET, normalizarlos por el valor de "r" correspondiente y compararlo con DVF_{SBU}.

Por último se describe el proceso de obtención del parámetro DVF mediante emulación. En el caso de no considerar MCUs, este valor se obtiene mediante la expresión 8.9, es decir, dividiendo el número de errores detectados por la cantidad de inyecciones realizadas. Tal y como se ha venido destacando en los capítulos anteriores, esta expresión va a ser válida cuando la cantidad de SBUs inyectados es suficientemente alta como para obtener valores estadísticamente representativos.

$$DVF_{SBU} = \alpha_{SBU} = \frac{Detected\ errors}{Injected\ SBUs} \tag{8.9}$$

Por otro lado, se propone la expresión 8.10 para la obtención del $DVF_{MCU}(LET)$ en el caso de considerar la ocurrencia de MCUs. En esta ecuación, la letra "s" hace referencia a cada una de las formas que aparecen reflejadas en la tabla 8.1. P_{sMCU} es la probabilidad de aparición de cada una de estas formas, que se corresponde con los valores de dicha tabla. Nótese que estos valores son dependientes de la energía de la partícula incidente (LET), tal y como se refleja en la tabla. α_{sMCU} indica la probabiliad de que un MCU de la forma "s" provoque un fallo en el sistema. Estos valores han de obtenerse mediante emulación, para después aplicar las ecuaciones 8.9 y 8.10. En este caso se requiere una campaña de emulación para cada una de las formas, lo que implica un cálculo del DVF mucho más tedioso que en el caso de considerar todos los SEUs como SBUs.

$$DVF_{MCU}(LET) = P_{SBU}(LET) * \alpha_{SBU} + \sum_{s} P_{sMCU}(LET) * \alpha_{sMCU} \tag{8.10}$$

En la ecuación 8.10 se puede apreciar que si P_{SBU} es 1 (todos los SEUs son SBUs) y el resto de probabilidades son 0, resulta que DVF = α_{SBU}, lo que es

consecuente con el planteamiento de considerar todos los SEUs como SBUs. A medida que aumenta la energía de las partículas radiantes el peso del resto de probabilidades aumenta en relevancia, incrementando el valor de DVF.

8.3 Emulación de MCUs y obtención de resultados

En esta sección se van a presentar las campañas de emulación llevadas a cabo para efectuar la comparación entre los dos modelos presentados en las secciones anteriores (ecuaciones 8.7 y 8.8). Para ello, es necesario hallar DVF_{SBU} y DVF_{MCU} mediante emulación. El otro parámetro involucrado en la comparativa es $r(LET)$, que identifica el número medio de bits afectados por un SEU, se obtiene directamente con los datos de la tabla 8.1 y la ecuación 8.6.

La comparación se va a efectuar de acuerdo con lo planteado en la sección anterior. En primer lugar, se va a obtener DVF_{SBU}, del mismo modo que se ha venido haciendo a lo largo de los capítulos anteriores mediante la ecuación 8.9. Después se va a obtener DVF_{MCU} utilizando la ecuación 8.10. Dicho valor va a ser normalizado utilizando el parámetro $r(LET)$ obtenido por la ecuación 8.5, para poder comparar el resultado con DVF_{SBU}. De este modo se consigue comparar las ecuaciones 8.7 y 8.8.

Se ha implementado una cadena de sumadores como UUT para las campañas de inyección de fallos. Se trata del mismo circuito sobre el cual han sido llevadas a cabo las campañas de emulación en los capítulos 6 y 7, y su esquema se representa en la figura 6.8. Es una cadena de 1000 sumadores que ocupa 28077 LUTs y 28299 flip-flops, lo que supone el 53 % del dispositivo Z7020. Para cada prueba se realizan 90786 inyecciones, que suponen el 0.5 % del total de bits del bitstream. La desviación máxima de la probabilidad de fallo estimada es del 0.0032 para un intervalo de confianza del 99.9 %.

La estimación de DVF_{SBU} se lleva a cabo exactamente igual que en las secciones anteriores, en las que la posibilidad de MCUs no ha sido tenida en cuenta. Se configura la FPGA con un bitstream que tiene un bit modificado, y posteriormente se verifica el funcionamiento correcto del diseño. Si se observa un funcionamiento erróneo a la salida se etiqueta el bit modificado como crítico. Esta operación se repite un número estadísticamente significativo de veces, para obtener finalmente el DVF dividiendo el número de bits críticos detectados por el número de inyecciones realizadas (ecuación 8.9).

Por otro lado, la estimación de DVF_{MCU} es más compleja. El cálculo de este

Tabla 8.2: Resultados de las campañas de inyección de MCUs de diferentes formas

Shape	Injections	Errors	% Errors	α iBGUs	%iBGUs
	90690	6723	α_{SBU} 7,41		
	90690	1194	α_{21MCU} 13,16	415	3,47
	90690	12307	α_{22MCU} 13,57	1387	11,27
	90690	1196	α_{23MCU} 13,18	384	3,21
	90690	11694	α_{24MCU} 12,89	325	2,77
	90690	14533	α_{3MCU} 16,02	1502	10,33
	90690	20172	α_{41MCU} 22,24	2531	12,54
	90690	19141	α_{42MCU} 21,10	2391	12,49
	90690	18769	α_{43MCU} 20,69	1805	9,61

parámetro mediante emulación requiere de la obtención de las diferentes α_{sMCU}. Esto exige llevar a cabo una campaña de emulación diferente para cada una de las formas de MCUs que están reflejadas en la tabla 8.1. En este caso, el parámetro α_{sMCU} se obtiene según la fórmula 8.10. Al igual que en el caso anterior, se calcula la probabilidad de error dividiendo el número total de fallos entre el número total de inyecciones. Las probabilidades de error α_{sMCU} obtenidas mediante emulación se ponderan con su probabilidad de aparición (P_{SBU} de la tabla 8.1).

Los resultados de las campañas de emulación se presentan en la tabla 8.2. La tercera columna muestra la cantidad de errores detectados en las 90690 inyecciones llevadas a cabo para cada forma. En la cuarta columna se presenta la probabilidad de error para cada una de las formas, obtenida al dividir la cantidad de errores por la cantidad de inyecciones. El concepto de iBGU (columnas 5 y 6) se discute en la siguiente sección.

Finalmente, la tabla 8.3, presenta el cálculo de DVF_{MCU} para los valores de LET de cada uno de los iones de los que se disponen datos, haciendo uso de la fórmula 8.10. Para la aplicación de dicha ecuación se han utilizado los valores de α_{SBU} y α_{sMCU} de la tabla anterior y los valores de P_{sMCU} de la tabla 8.1. En la próxima columna se representan los valores de DVF_{SBU} * r para cada uno de los iones. De esta forma se puede comparar la probabilidad de que un SEU provoque un fallo aplicando los diferentes métodos de medición (SBUs independientes columna 3 y teniendo en cuenta MCUs la columna 4). La siguiente columna muestra los valores de DVF_{MCU} normalizados por el valor r, para compararlos con la tasa de fallo del caso de SBUs independientes.

La última columna compara a modo de porcentaje los valores de la columna

anterior ($\frac{DVF_{MCU}}{r}$) y la probabilidad de fallo para SBUs, que es 7.41 (primera fila de la tabla 8.2).

Tabla 8.3: DVF_{MCU} para diferentes iones y comparación con SBU

Ion(LET)	r (Núm. medio errores por evento)	DVF_{MCU}	$DVF_{SBU} * r$	$\frac{DVF_{MCU}}{r}$	% respecto a SBUs
N(1.16)	1.011	7.50 %	7.49 %	7.41 %	100.11 %
O(1.54)	1.022	7.58 %	7.57 %	7.41 %	100.09 %
Ne(2.39)	1.103	8.14 %	8.17 %	7.38 %	99.59 %
Si(4.35)	1.101	8.11 %	8.16	7.36 %	99.40 %
Ar(7.27)	1.165	8.54 %	8.63 %	7.33 %	98.92 %
Cu(16.5)	1.237	9.02 %	9.17 %	7.29 %	98.40 %
Kr(25.0)	1.223	8.93 %	9.06 %	7.30 %	98.53 %
Xe(49.3)	1.343	9.69 %	9.95 %	7.21 %	97.37 %

En esta tabla se aprecia que DVF_{MCU} es similar a DVF_{SBU} (valor α_{SBU} de la tabla 8.3) para valores de LET bajos, y que a medida que la energía de las partículas se incrementa, DVF_{MCU} es mayor que DVF_{SBU}. Es lógico pensar que la probabilidad de fallo cuando todos los SEUs son SBUs va a ser inferior a la probabilidad de fallo cuando los SEUs pueden ser SBUs o MCUs. Sin embargo, cuando se iguala la cantidad de bits afectados por SEU, (los MCUs pasan a considerarse como múltiples SBUs independientes), la probabilidad de error es mayor para el caso de considerar únicamente SBUs que para el caso de considerar MCUs ($DVF_{SBU} > DVF_{MCU} \div r$).

8.4 IBGUs (i-Bit Group Upsets)

En el experimento descrito en la sección anterior se han efectuado inyecciones de MCUs con unas formas concretas para evaluar si un MCU sobre los bits modificados provoca o no un funcionamiento incorrecto del sistema. Este fallo del sistema puede deberse a dos causas:

- Al menos uno de los n bits modificados es un bit crítico
- Ninguno de los bits modificados es crítico por su cuenta, pero la modificación en grupo provoca un fallo.

El primero de los casos se corresponde con el planteamiento que se ha venido exponiendo a lo largo de la tesis. Cuando un bit crítico es afectado por un evento de radiación, el funcionamiento del sistema se ve interrumpido, con independencia de la naturaleza crítica o no crítica del resto de bits que conforman el evento

múltiple. En el segundo de los casos, por el contrario, los fallos solamente son detectables por medio de la emulación de MCUs, ya que nunca se va a poder inducir un funcionamiento incorrecto de este tipo mediante inyecciones simples.

Al mismo tiempo que se han realizado las emulaciones de MCUs se han ido efectuando las comprobaciones pertinentes para clasificar cada fallo detectado. En el momento en que se detecta que una emulación múltiple produce una alteración del correcto funcionamiento del sistema se analiza la criticalidad de los bits que componen el evento emulado cada uno por separado. En caso de que ninguno de estos bits sea crítico, se considera que es el grupo de bits el que tiene un efecto crítico. En la figura 8.2 se presenta el diagrama de flujo que se ha seguido a la hora de realizar la emulación de MCUs.

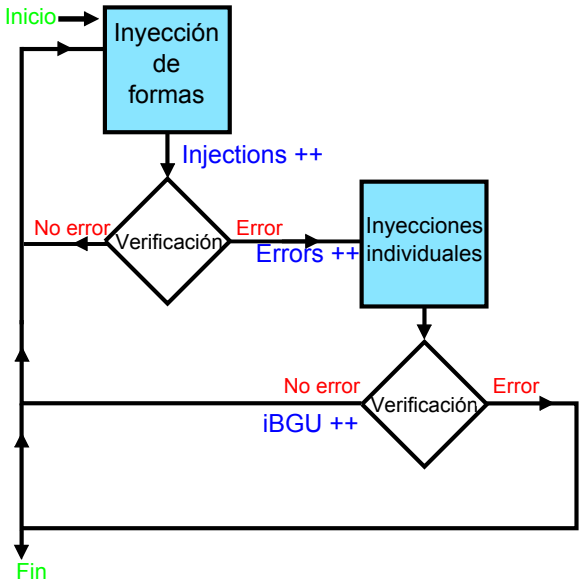


Figura 8.2: Diagrama de flujo del proceso de inyección y verificación de MCUs

Estos grupos de bits, que tomados individualmente no son críticos, pero al ser afectados simultáneamente por un evento producen un fallo del sistema han sido nombrados en este trabajo como iBGUs (i-Bit Group Upsets). Solamente tienen impacto cuando existe proximidad física entre los bits afectados. Se ha comprobado la existencia de iBGUs para bits aleatorios y se ha determinado que la presencia de este tipo de eventos es insignificante en casos en los que no hay cercanía física entre los bits modificados. Por lo que se puede concluir que los iBGUs se dan cuando se modifica un grupo de bits con una forma como las presentadas

en la tabla 8.1. Este hecho refuerza la hipótesis defendida en el artículo [16], que describe las relaciones entre proximidad física y lógica de los bits de configuración de las FPGAs de la serie7. Fuera de dichas formas también se han detectado iBGUs, pero siempre y cuando todos los bits modificados pertenezcan a una frame concreta o a las contiguas. Cuanto más alejados están los bits modificados, menor es la probabilidad de ocurrencia de iBGUs.

La figura 8.3 presenta una posible interpretación física de la ocurrencia de iBGUs. Aquí se representa una matriz de conmutación típica de las FPGAs, que se ha presentado en el capítulo 1. En la subfigura "a" se plantea una situación estándar en la que dos señales cruzan dicha matriz, y programando a"1" los transistores indicados, dichas señales se encaminan hacia otra zona de la FPGA. En la subfigura "b" se aprecia la ocurrencia de un SEU de un bit (un SBU) sobre un bit crítico. Al programarse erróneamente el transistor señalado con una U (Upset) y un rayo rojo, se cortocircuitan ambas señales, alterando el correcto funcionamiento del circuito. En las subfiguras c y d se representa la ocurrencia de un MCU de 2 bits sobre bits físicamente adyacentes. En los dos casos representados se produce un fallo en el sistema.

En el primero de los casos, uno de los bits que componen el evento múltiple incide sobre un transistor crítico, lo que hace que el circuito falle independientemente del efecto que el otro bit pueda acarrear. Sin embargo, ninguno de los bits modificados en la subfigura "d" es de los considerados como críticos, ya que al modificarse por separado no provocan ninguna modificación en el circuito implementado. Pero en el caso de que ambos bits se modifiquen de manera simultánea las señales verde y azul se cortocircuitan. En este caso se considera que ha ocurrido un iBGU de 2 bits, o lo que es lo mismo, un 2BGU.

Esto mismo se ilustra en las subfiguras e y f, pero representando un MCU de 3 bits. En la figura de la izquierda se presenta un caso en el que un MCU de 3 bits influye sobre algunos bits críticos y produce modificaciones determinantes en el diseño. Por otro lado, la figura de la derecha representa un 3BGU, es decir, un grupo de 3 bits que modificándose por separado nunca producirían alteraciones graves en el sistema, pero que al modificarse de manera conjunta producen una alteración en el correcto funcionamiento del sistema.

En la tabla 8.2 se presenta el peso de este tipo de errores para cada una de las formas evaluadas. Aquí se aprecia que los iBGUs suponen entre el 2 % y el 13 % del total de errores detectados.

En principio, cabría esperar que la emulación de MCUs proporcione una tasa de fallo superior a la obtenida mediante emulación de SBUs independientes. La primera permite detectar fallos inducidos por bits críticos y fallos inducidos por

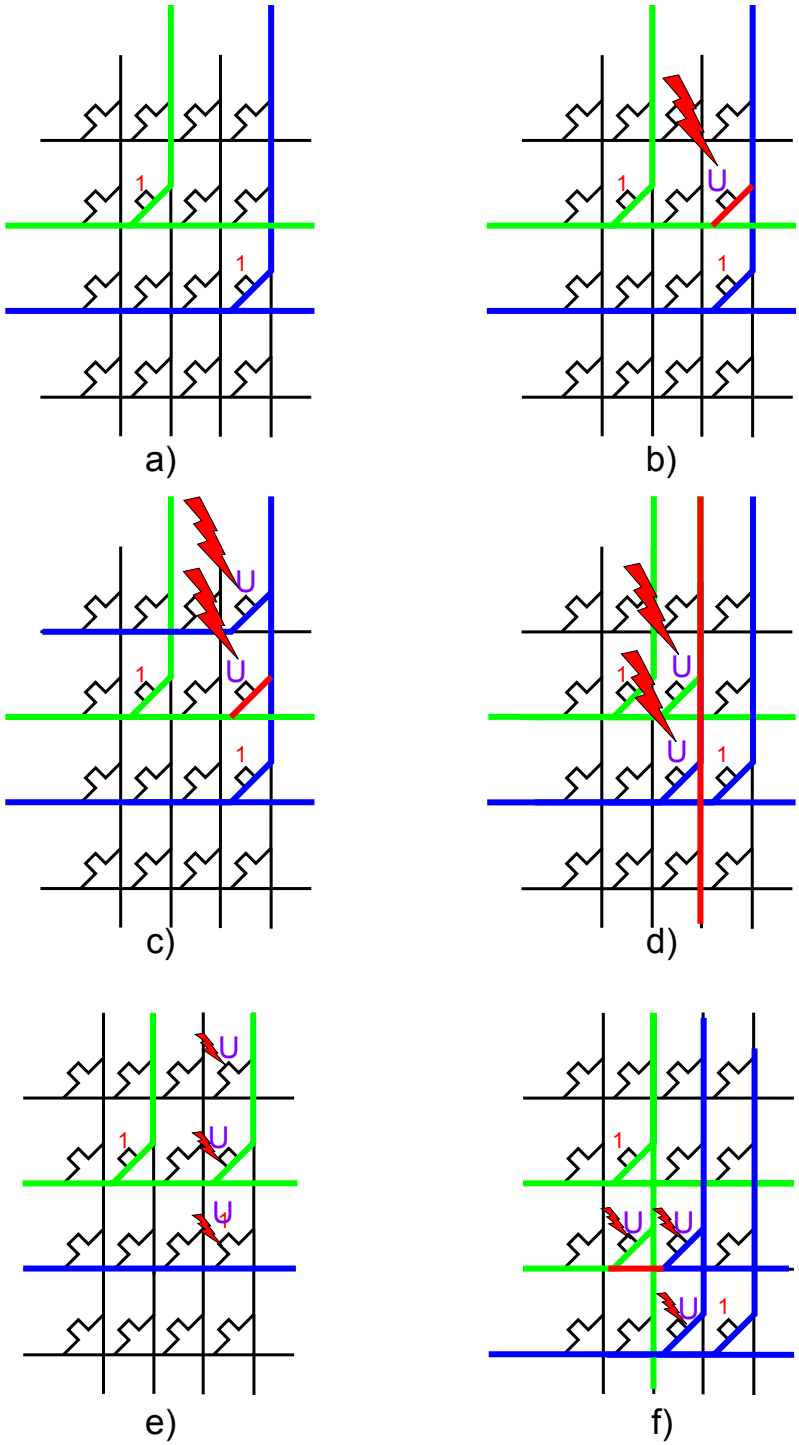


Figura 8.3: Interpretación física de los iBGUs

iBGUs, mientras que el peso de los iBGUs en la tasa de fallo resulta imposible de abordar mediante inyecciones simples. Sin embargo, la tasa de fallos obtenida mediante emulación de eventos múltiples es inferior a la obtenida de la emulación de SBUs. La interpretación de estos resultados se va a llevar a cabo a lo largo de la próxima sección.

8.5 Emplazamiento localizado

A la hora de modelar los MCUs como SBUs independientes, se comete la imprecisión de no considerar los errores denominados como iBGUs, por lo que cabría esperar una tasa de error más baja que la obtenida mediante emulación de MCUs. Sin embargo, a la vista de los resultados obtenidos, no es así. Al fenómeno que explica este hecho se le ha llamado "emplazamiento localizado".

Los diseños implementados en FPGAs rara vez ocupan la totalidad de los recursos lógicos del dispositivo. En la mayoría de los casos, los diseños se concentran en un área determinada. La herramienta de emplazamiento y rutado trata de ubicar las distintas funciones lógicas en recursos próximos entre sí. De esta forma, se consigue reducir el número de matrices de conmutación y de conexión que cruza cada señal, disminuyendo la congestión y mejorando la respuesta en frecuencia. Aún así, existen casos en los que no es conveniente agrupar demasiado los recursos, ya que se podrían llegar a saturar los recursos de rutado en zonas concretas.

El hecho de que el emplazamiento de los diseños esté localizado de esta forma tiene ciertas implicaciones desde el punto de vista estadístico. Para ilustrar esto se va a utilizar un ejemplo simplificado. Supongamos un diseño concreto, que toma el 53 % de los recursos del dispositivo, mientras que la otra mitad de la FPGA queda completamente vacía. Además, se supone una probabilidad de error constante del 14 % en la zona ocupada. Por lo tanto, la tasa de fallo global del dispositivo es del 7.42 % (0.14*0.53). Estos datos han sido escogidos porque coinciden exactamente con los valores obtenidos en el apartado anterior. El diseño implementado tiene una ocupación del 53 % en LUTs, y la tasa de fallo de un SBU es del 7.4 %.

En el caso de n SBUs independientes con una probabilidad de error p, la probabilidad de que no exista error es $(1-p)^n$, por lo que la probabilidad de error para n eventos SBU es su complementaria $1-(1-p)^n$. En el ejemplo descrito anteriormente la probabilidad de error es del 7.4 % (0.074), por lo que la tasa de fallos FR (Failure Rate) es de $1-(1-0,074)^n$. Sin embargo, cuando se considera un único evento múltiple, la probabilidad de error es 0 el 47 % de las veces, (el evento afecta a la zona no ocupada) y es $1-(1-0,14)^n$ para el otro 53 % de las veces. Por lo tanto, la tasa de fallos es $0,53*(1-(1-0,14)^n)$. En la figura

8.4 se presentan las tasas de fallo calculadas de ambas maneras para los datos expuestos en este caso particular.

Se puede apreciar que la probabilidad de error es siempre inferior cuando se considera la emulación de un MCU. Cuando un bit concreto es identificado como no crítico, la probabilidad de que los bits adyacentes sean también no críticos es superior a la probabilidad de bit no crítico de un bit cualquiera. Por lo tanto, en el caso de que el primero de los bits modificados sea no crítico, la probabilidad de que los siguientes tampoco lo sean es mayor en el caso de emulación de MCUs, donde los errores se inyectan en localizaciones con proximidad física. Por otro lado, en el caso de la emulación de SBUs independientes, la criticalidad del primero de los bits no condiciona la criticalidad del resto.

Sin embargo, este modelo simplificado está relativamente alejado de la realidad. Si bien la probabilidad de error es cero (o muy cercana a cero) en las zonas no ocupadas, la probabilidad de error en las zonas ocupadas nunca se puede considerar constante. Esto sucede debido a que la concentración de recursos lógicos no es uniforme a lo largo del dispositivo. Y de igual manera, la probabilidad de error dentro de una región considerada como ocupada tampoco es constante, ya que en las frames relativas a recursos ocupados por el diseño, existen amplias zonas de bits que no tienen ningún bit crítico.

Debido a esto, el diseño se ha modelado como una región ocupada de tasa de error constante del 22,5 %, que toma el 33 % del dispositivo, mientras que el otro 66 % permanece vacío. Las probabilidades de error obtenidos mediante esta aproximación son más cercanos a los resultados de emulación presentados en la tabla 8.2, y se presentan en la tercera columna de la tabla 8.4.

La tabla 8.4 establece una comparativa entre los datos obtenidos mediante emulación en la sección anterior (columna 1), y las estimaciones probabilísticas de acuerdo con el modelo matemático presentado en esta sección (figura 8.4) en las columnas 2 y 3. La columna 2 se corresponde exactamente con los datos representados en la figura (ocupación 53 % con tasa de fallo 14 %), mientras que la tercera columna presenta los datos de la hipótesis expuesta en el último párrafo (ocupación 33 % con tasa de fallo 22,5 %). Los valores representados en la primera de las columnas son los promedios de la tasa de error de las diferentes formas de MCUs de 2 y 4 bits (los SBUs y los MCUs de 3 bits solo tienen una forma, por lo que no hay que hacer promedio).

A la vista de estos datos, es evidente que un modelo que considera una ocupación del 33 % con tasa de fallo del 22,5 % para este caso concreto, presenta unos resultados bastante acordes con los obtenidos mediante emulación. Estos resultados indican que parece más apropiado considerar la ocupación real del dispositivo

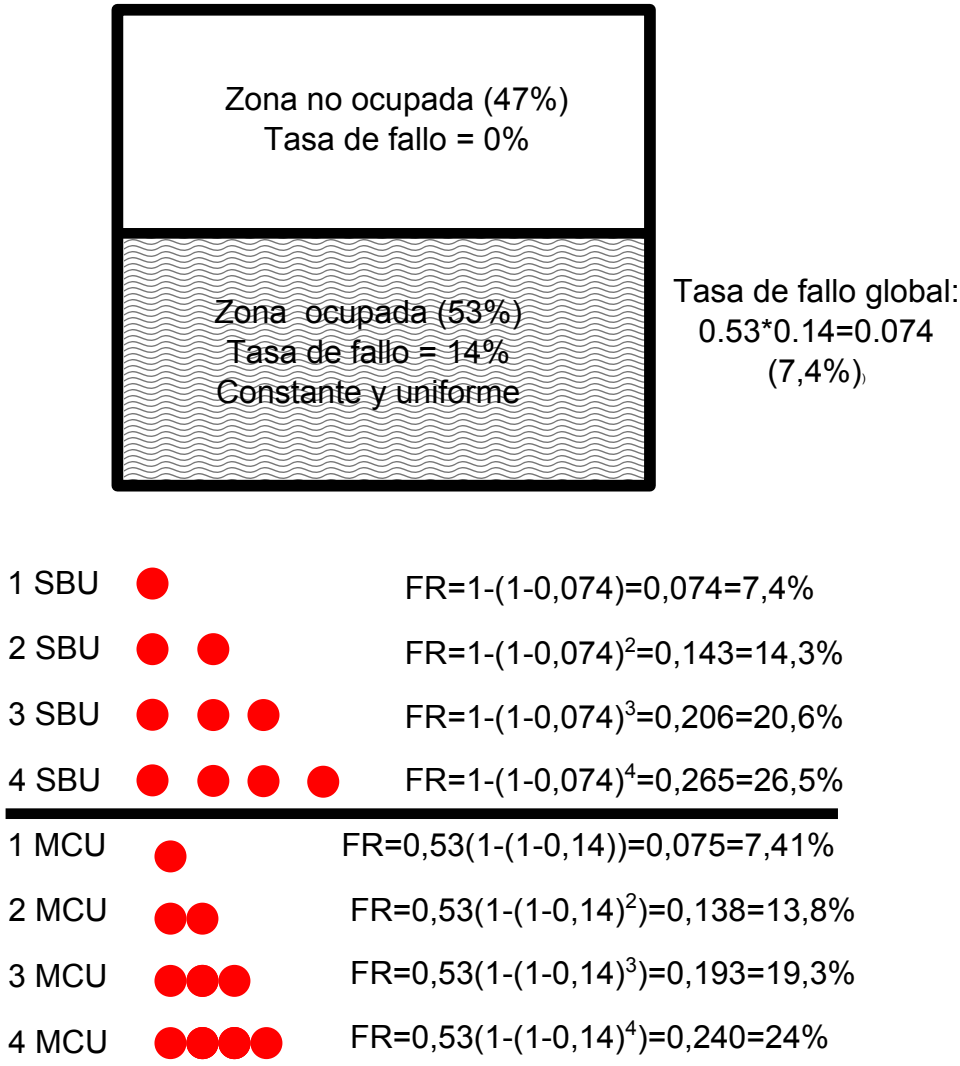


Figura 8.4: Esquema probabilístico de emplazamiento localizado

como un 33 %, en lugar del 53 % aportado por la herramienta software. Esto se debe a que regiones consideradas como ocupadas contienen amplias zonas con una cantidad muy baja de bits críticos, por lo que podrían considerarse como vacías.

Tabla 8.4: Efecto del emplazamiento localizado

	Datos emulación[1]	Ocupación 53 %	Ocupación 33 %
SBU	7.41 %	7.4 %	7.41 %
MCU 2 bit	13.21 %	13.8 %	13.16 %
MCU 3 bit	16.02 %	19.3 %	17.61 %
MCU 4 bit	21.34 %	24.0 %	21.07 %

[1] Promedio de la tasa de fallo de todas las formas de cada tamaño de MCU

Aún así, la estimación del valor real de ocupación no es un objetivo primordial. Además, resulta difícil llevar a cabo una estimación fiable, teniendo en cuenta que cada región ocupada tiene una densidad de recursos lógicos ocupados variable. A la vista de los datos de la tabla 8.4, se puede afirmar que el emplazamiento localizado de los recursos lógicos en regiones concretas tiene un impacto negativo sobre la tasa de fallo, que contrasta con el impacto positivo de los iBGUs, a la hora de realizar la comparativa entre los dos métodos de estudio de los MCUs analizados en el presente capítulo (SBUs independientes y emulación de MCUs).

El objetivo de esta sección es justificar la anomalía detectada en la anterior, que concluía que los fallos en eventos múltiples podían deberse a modificaciones de bits críticos y a modificaciones de iBGUs. Siendo estos segundos indetectabes mediante emulación de SBUs, el valor de la tasa de fallo obtenido mediante emulación de MCUs era inferior. La razón propuesta para explicar esto es la distribución no uniforme de los recursos a lo largo de la FPGA. Es decir, que el emplazamiento del diseño está localizado en unas regiones concretas, habiendo otras con una densidad más baja, o incluso vacías.

8.6 Conclusiones

En este capítulo, se ha propuesto la metodología para abarcar los MCUs desde la emulación de SEUs mediante el bitstream. Se han planteado una serie de ecuaciones, cuya aplicación permite la obtención de la tasa de fallos del dispositivo, haciendo uso de los datos obtenidos mediante emulación. Además de la emulación, se requiere la probabilidad de aparición de cada tipo de forma de MCU. Estas probabilidades han de ser obtenidas mediante campañas de radiación real. Para este capítulo estos datos han sido tomados del artículo [16].

Se ha realizado una prueba en un entorno de emulación real, implementando un diseño concreto. Se han obtenido los valores para α_{SBU} y α_{sMCU}. Estos valores combinados con las probabilidades de aparición de cada forma de MCU (P_{sMCU}) dan el factor de vulnerabilidad del dispositivo por medio de la ecuación 8.10. De esta forma, se conoce la tasa de fallo de un SEU, teniendo en cuenta que dicho SEU puede suponer un SBU o un MCU.

Se ha llevado a cabo una comparación entre esta forma de considerar los MCUs y la forma que tienen de contabilizar los eventos múltiples el resto de métodos de emulación observados en la literatura, que consiste en considerar un MCU como un conjunto de SBUs independientes. Los experimentos de emulación han mostrado resultados dispares para ambos casos. Se han propuesto dos mecanismos de fallo para explicar dicha disparidad. Por un lado, los iBGUs, que hacen que inyecciones múltiples sobre bits no críticos puedan provocar fallos en el sistema. Y por otro, las consecuencias del emplazamiento localizado, que supone una alteración probabilística, fruto de concentrar los diseños en áreas específicas. Se ha comprobado que este segundo efecto es predominante sobre el primero, y por lo tanto, la tasa de fallo obtenida mediante inyecciones múltiples es inferior a la obtenida por la consideración de SBUs independientes.

El efecto de los MCUs es más notable para LETs elevados, donde la probabilidad de eventos múltiples es mayor. Las MCU son una preocupación creciente debido a la reducción continua del tamaño de los dispositivos electrónicos. Como resultado, la metodología descrita en este documento podría ser útil en el futuro para incluir MCUs en el procedimiento de la estimación de la tasa de fallos, ya que aumenta el promedio de bits afectados por evento.

El estudio desarrollado en el presente capítulo permite justificar que el valor de tasa de fallo obtenido mediante SBUs independientes va a acotar por arriba la tasa real de fallo en presencia de MCUs, lo que supone el establecimiento de una cota superior para dicha tasa. Por lo tanto, si los valores de tasa de fallo obtenidos mediante SBU independientes son adecuados según las normativas de confiabilidad y seguridad, los valores reales en presencia de MCU no van a superar dicha cota. Aún así, en caso de que en el futuro la relevancia de los MCUs aumente debido a la reducción de voltajes de alimentación y tamaño de transistores, el procedimiento propuesto en este capítulo permite obtener datos de tasa de fallo más precisos.

Capítulo 9

Conclusiones, aportaciones y trabajo futuro

En este capítulo se resumen las principales conclusiones de la tesis. Posteriormente se enumeran las aportaciones más relevantes que ésta ha producido, así como las publicaciones científicas derivadas. Finalmente, se presentan algunas posibles líneas de trabajo futuro.

9.1 Conclusiones

9.1.1 Estado del arte, capítulos 1-5

En primer lugar se han estudiado los fundamentos básicos de las FPGAs, y se han analizado los conceptos principales de confiabilidad y seguridad. De esta forma, ha sido posible la interrelación de ambos conceptos, particularizando dichos conceptos generales de safety a los diseños en FPGA. De igual manera, se han estudiado los diferentes mecanismos de los que disponen los desarrolladores de FPGA para su mitigación.

Las FPGAs y otros dispositivos basados en lógica programable han evolucionado de manera continua desde su aparición a mediados de los 80. Teniendo en cuenta las últimas tendencias de la electrónica en general, y de este campo en particular, la llegada de nuevos dispositivos más potentes y con más funcionalidades parece

segura. Entre éstos destacan los SoCs que combinan FPGA y sistema microprocesador, que ofrecen múltiples posibilidades de diseño debido a la capacidad de integrar hardware a medida y software de altas prestaciones en un solo chip.

La principal causa de errores en FPGA son los SEE, que suceden cuando una partícula de radiación impacta en un semiconductor, generando corrientes parásitas no deseadas. Cuando estas corrientes inyectan o extraen carga de biestables que conforman celdas de memoria hasta el punto de invertir su valor lógico, se produce el efecto conocido como SEU. Este efecto concretamente tiene implicaciones muy particulares en el caso de las FPGAs, ya que, cuando afectan a bits de la memoria de configuración, pueden producir modificaciones críticas en el hardware implementado. Sin embargo, todos los bits de dicha memoria no son críticos, ya que pueden hacer referencia a recursos no utilizados por el diseño o sobrescritos inmediatamente. El porcentaje de bits críticos de la CRAM va a ser un parámetro clave a la hora de determinar la tasa de fallos.

Uno de los métodos más relevantes para la caracterización de la tasa de fallo debida al SEU en la memoria de configuración de la FPGA es la emulación. Consiste en configurar el dispositivo con un bitstream intencionadamente corrompido, para que así puedan almacenarse valores erróneos en la CRAM y poder estudiar las consecuencias.

9.1.2 Emulación de SEUs en FPGA, capítulo 5

Se ha realizado un análisis pormenorizado de las diferentes alternativas propuestas en la literatura para la emulación de SEUs en la memoria de configuración de una FPGA. Se ha discutido la aplicabilidad de cada uno de estos métodos para la estimación de la tasa de fallo en diseños industriales reales basados en FPGA, y se han detectado las siguientes carencias:

Existen dos vías para configurar la FPGA con un bitstream corrompido, la interfaz de configuración interna mediante RPD (Reconfiguración Parcial Dinámica) y la interfaz de configuración externa. El primero de éstos plantea problemas de intrusividad, ya que los errores pueden afectar a la lógica del control de la interfaz, llegando a bloquear el sistema. La segunda de las opciones puede requerir modificaciones a nivel de PCB o de hardware para el acceso a los pines relativos a la interfaz de configuración paralelo.

Una vez inyectado el error, ha de comprobarse si éste provoca un fallo a la salida del sistema, lo que se conoce como proceso de verificación. Los esquemas de verificación analizados son de dos tipos. El primero consiste en llevar a cabo una verificación funcional, válida únicamente para una aplicación concreta, algo que

exige un rediseño completo en el caso de cambiar de aplicación. Además, el tipo de aplicación testeada en estos sistemas es simple, por lo general aplicaciones matemáticas para uso exclusivamente académico. Este tipo de test se complica cuando hay que testear sistemas complejos que pueden llegar a tener múltiples funciones. La otra alternativa es la duplicación del módulo bajo test, algo que puede requerir modificaciones importantes en el diseño a nivel de hardware.

Después de haber verificado el correcto funcionamiento del sistema, es necesario hacer volver a los elementos secuenciales del sistema (flip-flops, registros, memorias ...) al estado inicial. La problemática de la recuperación del estado no se aborda en la mayoría de los esquemas analizados. La mayoría de estos implementan aplicaciones sencillas, que recuperan el estado inicial mediante un simple ciclo de reset. Sin embargo, esto no puede considerarse como un procedimiento universal.

9.1.3 Método propuesto, capítulo 6

En la literatura no se han encontrado plataformas de emulación basadas en SoCs que combinen FPGA con sistema procesador. Y lo cierto es que este tipo de dispositivos tiene una gran ventaja respecto de las FPGAs convencionales, ya que tienen una interfaz de configuración externa a la FPGA, pero accesible desde el sistema procesador. Debido a esto, es posible conseguir un alto throughput de configuración sin introducir modificaciones a nivel hardware. Este alto throughput permite realizar las emulaciones de SEUs mediante reconfiguración completa en un tiempo razonable, lo que soluciona el problema de la recuperación del estado. Además, al ser la lógica de configuración externa a la FPGA, no puede ser afectada por los SEUs emulados.

Se ha implementado el subsistema de verificación mediante un esquema BIST, y se ha podido demostrar la universalidad del entorno de test. Mientras la mayoría de métodos estudiados en la literatura se centran en la caracterización de una única aplicación, el presente método ha sido adaptado al testeo de sistemas diversos sin introducir grandes modificaciones. El test de una aplicación matemática sencilla ha podido ser adaptado a un sistema industrial real sin grandes complicaciones, mas allá de modificar un bloque concreto perteneciente al sistema de verificación.

9.1.4 Dependencia con la implementación, capítulo 7

Se ha testeado la cantidad de bits críticos de un circuito concreto al que se le han ido introduciendo modificaciones leves. A la vista de los resultados de estos

tests, se ha concluido que dichas modificaciones pueden provocar consecuencias importantes en la cantidad de bits críticos del sistema. Algunas consecuencias son esperables, como por ejemplo el incremento de la tasa de fallos con la frecuencia o la ocupación. Otras, en cambio, pueden llegar a ser imprevisibles, como por ejemplo el impacto del software/algoritmo de síntesis-implementación. Este hecho refuerza la necesidad de no aplicar ningún tipo de modificación después de la validación, algo que se contempla en el esquema de validación en V.

En este punto también se ha podido comprobar que una mayor ocupación a nivel de recursos lógicos (LUTs, flip-flops ...) no implica un aumento de la cantidad de bits críticos. Esto sucede debido a que la mayoría de los bits de configuración se corresponden con recursos de rutado. Por lo que va a ser el nivel de congestión en estos recursos lo que va a determinar principalmente la tasa de fallo global del sistema.

ciertas directivas de síntesis o emplazamiento y rutado tienen una repercusión directa, que puede ser importante sobre la tasa de fallo. El hecho de incluír un pblock como directiva de implementación a fin de localizar el diseño en una zona concreta del dispositivo puede provocar incrementos de hasta el 50 % en la cantidad de bits críticos del sistema. Sin embargo, cuando los pblocks se aplican a configuraciones TMR, se consigue reducir la tasa de fallo al disminuir el número de errores entre dominios. Las directivas de síntesis estudiadas (MAX_FANOUT y LUT_COMBINING) también pueden tener algún impacto sobre la tasa de fallo.

9.1.5 Emulación de eventos múltiples, capítulo 8

Los eventos múltiples o MCUs se producen cuando una partícula de gran energía incide contra el semiconductor, provocando errores en más de un bit de la memoria de configuración del dispositivo. Estas modificaciones han de producirse necesariamente en bits físicamente próximos entre sí. Efectos como la disminución del tamaño de los transistores o el descenso de las tensiones de alimentación pueden favorecer la aparición de este tipo de fenómenos en tecnologías futuras.

Es por esto que se plantea la necesidad de tener en cuenta los MCUs a la hora de llevar a cabo el proceso de emulación de SEUs. Se ha observado que los emuladores de SEUs analizados en la literatura no tienen en cuenta los eventos múltiples a la hora de llevar a cabo la estimación de la tasa de fallo general del sistema.

Se ha realizado un estudio comparando la emulación de eventos múltiples con el resultado de considerar los MCUs como múltiples SBUs independientes. Para ello, se ha propuesto un modelo matemático que describe las diferencias entre la emulación de MCUs y la consideración de éstos como conjuntos eventos simples.

Se ha llevado a cabo la emulación de eventos múltiples y se ha observado que la tasa de fallos medida por esta segunda opción es siempre superior a la primera.

Para explicar esto se han propuesto dos efectos, emplazamiento localizado e iBGUs. El primero de ellos es una consecuencia de la concentración de recursos lógicos en zonas muy concretas de la FPGA, y provoca una disminución del número de bits críticos medido mediante emulación múltiple. Por otro lado, los iBGUs consisten en la existencia de grupos de bits que en ningún caso son críticos por separado, pero que al inyectarse un error múltiple sobre ellos se produce un fallo en el sistema. Se concluye que el impacto del emplazamiento localizado es mayor que el de los iBGUs.

9.2 Principales aportaciones

Diferenciación de etapas en el proceso de emulación: Se ha descrito el proceso de emulación de SEU como la sucesión de 3 etapas claramente diferenciadas, que son inyección de errores, verificación y recuperación del estado inicial. Los métodos de emulación de SEUs en FPGAs estudiados en la literatura se centran principalmente en explicar el proceso de inyección y en la discusión de los resultados obtenidos. Para ello, se implementa un sistema fácil de verificar y con una posibilidad inmediata de recuperación del estado (reset). La diferenciación propuesta en este trabajo permite diseccionar con mayor precisión los diferentes métodos de emulación existentes. Se han valorado las fortalezas y debilidades de cada una de éstos métodos en cada una de las 3 etapas y se han extraído conclusiones al respecto.

Método de emulación de SEUs basado en SoC: Se ha propuesto un método de emulación de SEUs basado en un SoC que combina FPGA con sistema procesador. Este tipo de dispositivos permite su configuración desde una interfaz intermedia entre el procesador y la FPGA. Al ser una interfaz interna es posible obtener una alta velocidad de inyección utilizando bitstreams completos, consiguiendo un mecanismo de recuperación del estado universal y eficiente en tiempo sin introducir ninguna modificación de hardware. De igual manera, al ser una interfaz externa a la FPGA, la lógica que controla dicha interfaz no puede quedar afectada por los SEUs emulados, evitando bloqueos del sistema. Al utilizar bitstreams completos se evita la necesidad de utilizar el reset para la recuperación del estado, ya que la tendencia de diseño en FPGAs es la de restringir el uso de esta señal a los casos en los que es estrictamente necesaria.

Verificación universal, esquema BIST: Se propone un esquema de verificación universal basado en BIST. De esta forma, es posible adaptar el mismo al

entorno de verificación sin añadir grandes modificaciones para el testeo de diferentes sistemas. Además de esto, los diseños industriales reales suelen ser muy complejos, incorporando funcionalidades múltiples. Por lo tanto, no resulta sencillo plantear un esquema de verificación funcional para este tipo de sistemas, algo que queda resuelto mediante la aplicación de un esquema de verificación universal. El esquema BIST propuesto se implementa internamente en la FPGA, por lo que no hay que realizar modificaciones a nivel de PCB. El tamaño del circuito propuesto es bastante reducido, por lo que no presenta una gran intrusividad en la medida final.

Dependencia con las etapas de diseño: Se ha implementado un circuito concreto, sobre el que se han ido haciendo pequeñas modificaciones. Aquí se ha comprobado que las modificaciones en cualquiera de estas fases puede tener efectos impredecibles sobre la tasa de fallo del sistema. Se le ha dado especial relevancia a la impredecibilidad de la etapa de implementación, por las implicaciones que ella conlleva. De aquí se deduce que los métodos de estimación basados en estudiar los datos de ocupación o de síntesis son menos precisos que la emulación, al no considerar la etapa de implementación. Por otro lado, se pone en valor la necesidad de cumplir con el modelo de verificación en V, ya que los métodos basados en testear la tasa de fallo en plataformas concretas para después re-implementar el diseño en el PCB de trabajo puedan resultar imprecisos.

Eventos múltiples: Se ha propuesto una metodología para abarcar los MCUs desde la emulación de SEUs mediante el bitstream. Las ecuaciones planteadas permiten la obtención de la tasa de fallos del dispositivo, haciendo uso de los datos obtenidos mediante emulación. Se ha podido comprobar que la tasa de fallo obtenida siguiendo este procedimiento es inferior a la que se obtendría en el caso de considerar los MCU como SBUs múltiples e independientes. En el caso concreto que se ha evaluado, las diferencias de la tasa de fallos medida para ambas aproximaciones es pequeña. Sin embargo, se trata de una aportación novedosa que puede tener mayor relevancia en el futuro a medida que avanza la tecnología.

9.3 Publicaciones científicas

9.3.1 Revistas

- Igor Villalta, Unai Bidarte, Julen Gómez-Cornejo, Jaime Jiménez, Jesus Lázaro; SEU Emulation in Industrial SoCs combining Microprocessor and FPGA; Reliability Engineering & System Safety; JCR 2017 4.139 (Q1) Volumen 170, Febrero 2018, Págs 53-63, https://doi.org/10.1016/j.ress.2017.09.028

En este artículo se presentan las tres primeras aportaciones descritas en la sección anterior. El objetivo principal es el de presentar el método de emulación de SEUs basado en SoC Zynq (capítulo 6), aunque también se presenta la discusión sobre las diferentes fases del proceso de emulación de SEUs en FPGA para los emuladores de SEU existentes en la literatura (capítulo 5).

- Igor Villalta, Unai Bidarte, Julen Gómez-Cornejo, Jesús Lázaro, Armando Astarloa; Estimating the SEU Failure Rate of Designs implemented in FPGAs in Presence of MCUs; Microelectronics Reliability; JCR 2017 = 1.236 (Q3); Volumen 78, Noviembre 2017, Págs 85-92; https://doi.org/10.1016/j.microrel.2017.08.003

  Artículo que discute lo comentado en el capítulo relativo a la emulación de MCUs (Capítulo 8).

9.3.2 Congresos

- Igor Villalta, Unai Bidarte, Uli Kretzschmar, Armando Astarloa, Jesús Lázaro; Fault Injection System for SEU Emulation in Zynq SoCs ; Conference on Field Programmable Logic and Applications (FPL2014) ; Munich, 2-4 Septiembre 2014; 10.1109/DCIS.2014.7035579

  Trabajo muy preliminar acerca del método propuesto. En este caso se utilizaban bitstreams de 1 frame y la verificación no se llevaba a cabo mediante un esquema BIST.

- Igor Villata, Unai Bidarte, Uli Kretzschmar, Gorka Santos, Asier Matallana; Functional Verification for SEU Emulation in FPGA Designs ; Congreso: Jornada de Computación Reconfigurable y Aplicaciones (JCRA 2014) ; Valladolid, 17-19 Septiembre 2014 ;

  Trabajo que expone la necesidad de un sistema de verificación universal para emuladores de SEUs en FPGAs.

- Igor Villalta, Unai Bidarte, Gorka Santos, Asier Matallana, Jaime Jiménez; Fault Injection System for SEU Emulation in Zynq SoCs; Conference on Design of Circuits and Integrated Systems (DCIS 2014); Madrid, 26-28 Noviembre; doi: 10.1109/DCIS.2014.7035579

  Comunicación que discute la conveniencia de utilizar bitstreams completos respecto a bitstreams de 1 frame.

- Igor Villalta, Unai Bidarte, Julen Gómez-Cornejo, Jesús Lázaro, Carlos Cuadrado; Dependability in FPGAs, a Review; Conference on Design of Circuits and Integrated Systems (DCIS 2015); Lisboa, 25-27 Noviembre doi: 10.1109/DCIS.2015.7388570

  Revisión de los diferentes métodos para la obtención de sistemas confiables basados en FPGA (capítulos 2 y 4).

- Igor Villalta, Unai Bidarte, Julen Gomez-Cornejo, Jaime Jiménez, Carlos Cuadrado; Effect of Different Design Stages on the SEU Failure Rate of FPGA systems; Conference on Design of Circuits and Integrated Systems (DCIS 2016); Granada, 23-25 Noviembre; doi: 10.1109/DCIS.2016.7845269

  Trabajo que discute el efecto de las diferentes etapas de diseño en la tasa de fallo (capítulo 7).

9.4 Trabajo futuro

El mercado de las FPGAs está en constante ebullición. La aparición de nuevas familias de dispositivos basados en lógica programable cada dos o tres años con mayores densidades de integración y funcionalidades añadidas, ofrece a los desarrolladores de FPGA múltiples posibilidades para aumentar las prestaciones de sus diseños, que pueden incluso servir para alcanzar nuevos mercados. Por lo tanto, una línea de investigación en el futuro próximo consiste en aprovechar dichos avances para la obtención de diseños con un mayor grado de confiabilidad, de manera que se pueda justificar el uso de FPGAs comerciales en aplicaciones de safety, lo que se conoce como COTS. Como claro ejemplo de esto, se destacan los SoCs con procesadores Quad-Core, cuyos núcleos podrían emplearse para implementar arquitecturas de tipo TMR para el software.

Otra de las implicaciones de la reducción del tamaño de los transistores es la disminución de las tensiones de alimentación, lo que provoca que las celdas de memoria acumulen menos carga. Debido a esto, disminuyen los umbrales de las celdas de memoria, incrementando la probabilidad de que dicho umbral se rebase, provocándose un error. Los fabricantes suelen prevenir este riesgo introduciendo mejoras a nivel de silicio para mitigar estos efectos en la medida de lo posible. Aún así, las mediciones de las secciones de cruce de las celdas de memoria han de continuar produciéndose, por lo que los tests de radiación en FPGAs son parte importante del futuro de esta tecnología. Dichas mediciones son datos necesarios para la estimación de la tasa de fallo mediante emulación de SEU en la CRAM de la FPGA.

En lo referente a la emulación de SEUs en la memoria de configuración de FPGAs, una posible línea de trabajo futuro consiste en avanzar en el conocimiento del bitstream. Un mejor conocimiento de este fichero permitiría hacer las inyecciones de errores únicamente en bits que puedan afectar al funcionamiento del circuito implementado, mejorando la eficiencia de la inyección de errores. El conocimiento del bitstream también permitiría incrementar las garantías de que la emulación de un MCU de una forma concreta está siendo llevada a cabo en bits con proximidad física real. El procedimiento para avanzar en este sentido es el de generar bitstreams de circuitos pequeños en el que se introducen cambios mínimos para identificar qué bits cambian para cada bitstream generado.

Otra línea de trabajo se basa en aplicar este método en SoCs de otras familias, ya sean de Xilinx o de otros fabricantes. En principio, el método presentado es universal, ya que el único requisito es que el SoC permita configurar la FPGA desde el sistema procesador a través de una interfaz hard. Cada familia de dispositivos puede tener características que ayuden a incrementar la eficiencia del sistema, así como imprevistos que pueden dificultar su adaptación. Por lo tanto, la posibilidad de adaptar el método para FPGAs de diferentes familias y fabricantes de una manera simple permitiría justificar su universalidad.

La aplicación del presente método a FPGAs convencionales (que no sean SoC con PS) es otro posible campo de trabajo. En principio, el subsistema de verificación aquí propuesto no requeriría ninguna modificación. Las dificultades provienen de efectuar la inyección y la recuperación del estado. En principio, lo ideal sería utilizar directamente la interfaz de configuración paralelo desde otro chip en el mismo PCB, para poder realizar la reconfiguración completa en un tiempo razonable. De todas formas, en función de la aplicación podrían sacrificarse ciertos aspectos. En el caso de que el retorno al estado inicial sea sencillo, una posibilidad sería realizar las inyecciones mediante bitstreams parciales desde el JTAG.

En el capítulo 2 se ha mencionado que todos los modos de fallo no tienen las mismas consecuencias, y que éstos se pueden clasificar en tres categorías: leves, severos o catastróficos. Por lo tanto, una posible línea de investigación consistiría en la definición de un entorno de verificación que pueda clasificar el tipo de fallo producido según su severidad.

Por último, para poder justificar la tasa de fallo del sistema completo sería interesante avanzar en el establecimiento de una metodología concreta para la caracterización de la tasa de fallo del sistema procesador.

Apéndice A

Lista de parámetros

A continuación se va a aportar una breve descripción acerca de los parámetros que aparecen en las diferentes fórmulas a lo largo del documento, especialmente en los capítulos 3 y 8. La idea de este anexo es la de servir como apoyo para el lector cuando se encuentre con una fórmula durante la lectura, y pueda así comprender mejor el significado de cada uno de los términos que componen dicha expresión.

- **FR (Failure rate), tasa de fallo:** Cantidad de fallos en el sistema o interrupciones de servicio por unidad de tiempo. Lo habitual es aportar este dato en FIT (Failures In Time), cantidad de fallos en 10^9 horas.
- $\boldsymbol{\tau}$**, tasa de eventos:** Cantidad de eventos potencialmente peligrosos que afectan a un sistema por unidad de tiempo. Se puede presentar en s^{-1} o en FIT. Por evento peligroso se entiende cualquier tipo de SEE (SEU, SEL, SET, SEFI...).
  - $\boldsymbol{\tau_{SEU}}$**, tasa de SEU (Single Event Upsets):** Cantidad de SEUs por unidad de tiempo. puede presentar en $s^{-}1$ o en FIT. El SEU es el evento que provoca errores en la memoria de configuración, que pueden ser errores simples (SBU) o errores múltiples (MCU).
  - $\boldsymbol{\tau_{error}}$**:** tasa de bits modificados en la memoria de configuración debido al SEU por unidad de tiempo, independientemente de si son modificados por eventos simples o múltiples. Se expresa en s^{-1} o FIT.
  - $\boldsymbol{\tau_{bit}}$**:** Cantidad de bits afectados por SEUs en 1 Mb de memoria de configuración por unidad de tiempo. Se obtiene al dividir τ_{error} por el tamaño de la CRAM en Mb. De esta forma, los tests realizados para un

dispositivo concreto puedan extrapolarse a dispositivos de diferentes tamaños de la misma familia de FPGAs. Éste es el parámetro que aporta Xilinx en sus reportes, habitualmente en FIT/Mb.

- $\boldsymbol{\sigma}$**, Sección de cruce:** Área que indica la probabilidad de que una partícula de radiación genere un evento potencialmente peligroso en un dispositivo semiconductor. Cuanto mayor sea esta área, mayor será la probabilidad de que suceda un evento. Esta área, que se presenta en m^2, es dependiente de la energía de las partículas incidentes, ya que a mayor energía mayor es la probabilidad de que la partícula provoque un evento. Este parámetro aparece a veces como $\sigma(E)$ ó $\sigma(LET)$, ya que la energía de los iones pesados se proporciona en función del LET.
  - $\boldsymbol{\sigma_{SEU}}$**:** Área que indica la probabilidad de que una partícula de radiación genere un SEU. pueden ser errores simples (SBU) o errores múltiples (MCU).
  - $\boldsymbol{\sigma_{error}}$**:** Área que indica la probabilidad de que un bit de la memoria del dispositivo resulte modificado, independientemente de si ha sido como causa de un SBU o un MCU.
  - $\boldsymbol{\sigma_{bit}}$**:** Área que indica la probabilidad de que una partícula de radiación provoque un error en un bit concreto. Se obtiene dividiendo σ_{error} por el tamaño de la memoria.
- $\boldsymbol{DVF}$**, Design Vulnerability Factor:** Probabilidad de que un evento SEU provoque un fallo en el sistema.
  - $\boldsymbol{DVF_{SBU}}$**:** DVF calculado considerando únicamente SBUs, probabilidad de que un SBU provoque un fallo en el sistema.
  - $\boldsymbol{DVF_{MCU}}$**:** DVF calculado considerando que los SEUs pueden significar tanto SBUs como MCUs.
- $\boldsymbol{P_{SBU}}$**:** Probabilidad de que un SEU sea de tipo SBU.
- $\boldsymbol{P_s}$**:** Probabilidad de que un SEU sea un MCU de una forma s concreta.
- $\boldsymbol{\alpha_s}$**:** Probabilidad de que un MCU de la forma s provoque un fallo en el sistema
- **r, Valor medio de errores por evento:** Siempre va a ser mayor que 1. Para energías bajas va a ser muy cercano a 1, ya que la mayoría de los SEUs van a ser SBUs. Mientras que para energías altas, donde las probabilidades de MCUs son mucho más representativa.

Bibliografía

[1] M. Straka, J. Kastil, Z. Kotasek y L. Miculka, "Fault tolerant system design and seu injection based testing," *Microprocessors and Microsystems*, vol. 37, nº 2, páginas 155 – 173, 2013.

[2] N. Battezzati, F. Margaglia, M. Violante, F. Decuzzi, D. M. Codinachs y B. Bancelin, "Application-oriented seu cross-section of a processor soft core for atmel rhbd fpgas," *IEEE Transactions on Nuclear Science*, vol. 58, nº 3, páginas 987–992, June 2011.

[3] J. L. Nunes, T. Pecserke, J. C. Cunha y M. Zenha-Rela, "Fired – fault injector for reconfigurable embedded devices," en *2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*, páginas 1–10, Nov 2015.

[4] L. Sterpone y M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 54, nº 4, páginas 965–970, Aug 2007.

[5] F. Ghaffari, F. Sahraoui, M. E. A. Benkhelifa, B. Granado, M. A. Kacou y O. Romain, "Fast sram-fpga fault injection platform based on dynamic partial reconfiguration," en *2014 26th International Conference on Microelectronics (ICM)*, páginas 144–147, Dec 2014.

[6] U. Kretzschmar, A. Astarloa, J. Jimenez, M. Garay y J. D. Ser, "Compact and fast fault injection system for robustness measurements on sram-based fpgas," *IEEE Transactions on Industrial Electronics*, vol. 61, nº 5, páginas 2493–2503, May 2014.

[7] J. M. Mogollon, H. Guzman-Miranda, J. Napoles, J. Barrientos y M. A. Aguirre, "Ftunshades2: A novel platform for early evaluation of robustness against see," en *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, páginas 169–174, Sept 2011.

[8] N. A. Harward, M. R. Gardiner, L. W. Hsiao y M. J. Wirthlin, "Estimating soft processor soft error sensitivity through fault injection," en *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, páginas 143–150, May 2015.

[9] G. A. Klutke, P. C. Kiessler y M. A. Wortman, "A critical look at the bathtub curve," *IEEE Transactions on Reliability*, vol. 52, nº 1, páginas 125–129, March 2003.

[10] M. S. Alvarez-Alvarado y D. Jayaweera, "Bathtub curve as a markovian process to describe the reliability of repairable components," *IET Generation, Transmission Distribution*, vol. 12, nº 21, páginas 5683–5689, 2018.

[11] C. Slayman, *JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors*, capítulo 3, páginas 55–76. Boston, MA: Springer US, 2011. https://doi.org/10.1007/978-1-4419-6993-4_3

[12] A. Akturk, R. Wilkins y J. McGarrity, "Terrestrial neutron induced failures in commercial sic power mosfets at 27c and 150c," en *2015 IEEE Radiation Effects Data Workshop (REDW)*, páginas 1–5, July 2015.

[13] L. Sterpone y L. Boragno, "Analysis of radiation-induced cross domain errors in tmr architectures on sram-based fpgas," en *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, páginas 174–179, July 2017.

[14] Y. Li, B. Nelson y M. Wirthlin, "Synchronization techniques for crossing multiple clock domains in fpga-based tmr circuits," *IEEE Transactions on Nuclear Science*, vol. 57, nº 6, páginas 3506–3514, Dec 2010.

[15] Xilinx, "Device reliability report first half 2016 ug116 (v10.5.2)," April 4 2014.

[16] M. Wirthlin, D. Lee, G. Swift y H. Quinn, "A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and secded-protected arrays," *IEEE Transactions on Nuclear Science*, vol. 61, nº 6, páginas 3080–3087, Dec 2014.

[17] G. Lemieux, E. Lee, M. Tom y A. Yu, "Directional and single-driver wires in fpga interconnect," en *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, páginas 41–48, Dec 2004.

[18] U. Farooq, Z. Marrakchi y H. Mehrez, *Tree-based Heterogeneous FPGA Architectures*. Springer, 2012.

[19] S. M. Trimberger, "Three ages of fpgas: A retrospective on the first thirty years of fpga technology," *Proceedings of the IEEE*, vol. 103, nº 3, páginas 318–331, March 2015.

[20] W. Carter *et al.*, "A user programmable reconfigurable gate array," en *IEEE 1986 Custom Integrated Circuits Conference*, páginas 233–235, 1986.

[21] J. Babb, R. Tessier, M. Dahl, S. Z. Hanono, D. M. Hoki y A. Agarwal, "Logic emulation with virtual wires," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, nº 6, páginas 609–626, Jun 1997.

[22] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. H. Touati y P. Boucard, "Programmable active memories: reconfigurable systems come of age," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, nº 1, páginas 56–69, March 1996.

[23] R. Rajsuman, "Design of reprogrammable fpla," *Electronics Letters*, vol. 25, nº 11, páginas 715–716, May 1989.

[24] Z. Bao, M. Bhushan, S. Ran y J. E. Lukens, "Fabrication of high quality, deep-submicron nb/alo/sub x//nb josephson junctions using chemical mechanical polishing," *IEEE Transactions on Applied Superconductivity*, vol. 5, nº 2, páginas 2731–2734, June 1995.

[25] B. Tseng, J. Rose y S. Brown, "Improving fpga routing architectures using architecture and cad interactions," en *Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers Processors*, páginas 99–104, Oct 1992.

[26] Xilinx, "Spartan-3 fpga family data sheet, ds099 june 27,," 2013.

[27] F. ming Xiao, D. sheng Li, G. ming Du, Y. kun Song, D. li Zhang y M. lun Gao, "Design of axi bus based mpsoc on fpga," en *2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication*, páginas 560–564, Aug 2009.

[28] A. Astarloa, U. Bidarte y A. Zuloaga, "A reconfigurable soc architecture for high volume and multichannel data transaction in industrial environments," en *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, vol. 3, páginas 2322–2327 vol.3, Nov 2002.

[29] Xilinx, "Zynq-7000 soc data sheet: Overview, ds190 (v1.11.1) july 2,," 2018.

[30] Xilinx, "Ultrafast design methodology guide for the vivado design suite, ug949 (v2014.1)," April 2 2014.

[31] Xilinx, "System generator for dsp, ug640 (v 14.3) october 16, 2012," 2012.

[32] Xilinx, "Vivado design suite user guide, high-level synthesis, ug902 (v2014.1) may 30, 2014," 2014.

[33] Xilinx, "Vivado design suite user guide,synthesis, ug901 (v2015.3) september 30, 2015," 2015.

[34] Xilinx, "Vivado design suite user guide, implementation, ug904 (v2017.3), october 27, 2017," 2017.

[35] Xilinx, "7 series dsp48e1 slice user guide, ug479 (v1.10) march 27, 2018."

[36] Xilinx, "7 series fpgas clocking resources user guide, ug472 (v1.13) march 1, 2017."

[37] Xilinx, "7 series fpgas memory resources, ug473 (v1.12) september 27, 2016."

[38] Xilinx, "Zynq-7000 all programmable soc technical reference manual, ug585 (v1.12.1) december 6, 2017."

[39] Xilinx, "Zynq ultrascale+ mpsoc product tables and product selection guide, xmp104 (v2.4)," 2018.

[40] Intel, "Intel fpga product catalog, version18.1," 2018.

[41] Microsemi, "Smartfusion2 soc fpgas," 2017.

[42] Xilinx, "7 series fpgas configuration user guide, ug470 (v1.13) march 21, 2018."

[43] A. Avizienis, J. C. Laprie, B. Randell y C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, nº 1, páginas 11–33, Jan 2004.

[44] A. Avizienis y J. C. Laprie, "Dependable computing: From concepts to design diversity," *Proceedings of the IEEE*, vol. 74, nº 5, páginas 629–638, May 1986.

[45] I. E. C. (IEC), "Iec60300-1:2014 dependaibility management," 2014.

[46] T. Yuan, W. Kuo y S. J. Bae, "Detection of spatial defect patterns generated in semiconductor fabrication processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, nº 3, páginas 392–403, Aug 2011.

[47] C. Gao, W. Guo y Y. Sun, "Research on diagnostics and prognostics of fpga-based system," en *Proceedings of the IEEE 2012 Prognostics and System*

Health Management Conference (PHM-2012 Beijing), páginas 1–5, May 2012.

[48] E. Delic, M. Schreiber, A. Hayek y J. Borcsok, "Validation of a sil3 middleware for safety-related system-on-chips," en *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, páginas 85–90, May 2013.

[49] H. J. Vishnukumar, B. Butting, C. Muller y E. Sax, "Machine learning and deep neural network, artificial intelligence core for lab and real-world test and validation for adas and autonomous vehicles: Ai for efficient and quality test and validation," en *2017 Intelligent Systems Conference (IntelliSys)*, páginas 714–721, Sept 2017.

[50] D. Alexandrescu, L. Sterpone y C. Lopez-Ongil, "Fault injection and fault tolerance methodologies for assessing device robustness and mitigating against ionizing radiation," en *2014 19th IEEE European Test Symposium (ETS)*, páginas 1–6, May 2014.

[51] D. D. Ward, "Misra standards for automotive software," en *IEE Conference on Automotive Electronics*, páginas 5–18, March 2006.

[52] I. E. C. (IEC), *Functional safety and IEC61508 A basic Guide*, 2004.

[53] P. McNelles, Z. C. Zeng, G. Renganathan, G. Lamarre, Y. Akl y L. Lu, "A comparison of fault trees and the dynamic flowgraph methodology for the analysis of fpga-based safety systems part 1: Reactor trip logic loop reliability analysis," *Reliability Engineering & System Safety*, vol. 153, páginas 135 – 150, 2016. http://www.sciencedirect.com/science/article/pii/S0951832016300424

[54] R. Bono, R. Alexander, A. Dorman, Y.-J. Kim y J. Reisdorf, "Analyzing reliability - a simple yet rigorous approach," *IEEE Transactions on Industry Applications*, vol. 40, nº 4, páginas 950–957, July 2004.

[55] H. Aliee y H. R. Zarandi, "A fast and accurate fault tree analysis based on stochastic logic implemented on field-programmable gate arrays," *IEEE Transactions on Reliability*, vol. 62, nº 1, páginas 13–22, March 2013.

[56] A. Javanainen *et al.*, "Linear energy transfer of heavy ions in silicon," *IEEE Transactions on Nuclear Science*, vol. 54, nº 4, páginas 1158–1162, Aug 2007.

[57] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, nº 3, páginas 305–316, Sept 2005.

[58] T. Vanat, F. Krizek, J. Ferencei y H. Kubatova, "Comparing proton and neutron induced seu cross section in fpga," en *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, páginas 1–4, April 2016.

[59] T. R. Oldham y F. B. McLean, "Total ionizing dose effects in mos oxides and devices," *IEEE Transactions on Nuclear Science*, vol. 50, nº 3, páginas 483–499, June 2003.

[60] Altera, "Introduction to single-event upsets, wp-01206-1.0,," September 2013.

[61] A. Johnston, *Reliability and radiation effects in compound semiconductors*, W. S. P. C. P. Lt, Ed., 01 2010.

[62] J. L. Barth, C. S. Dyer y E. G. Stassinopoulos, "Space, atmospheric, and terrestrial radiation environments," *IEEE Transactions on Nuclear Science*, vol. 50, nº 3, páginas 466–482, June 2003.

[63] D. Heynderickx, B. Quaghebeur, J. Wera, E. J. Daly y H. D. R. Evans, "New radiation environment and effects models in esa's space environment information system (spenvis)," en *Proceedings of the 7th European Conference on Radiation and Its Effects on Components and Systems, 2003. RADECS 2003.*, páginas 643–646, Sept 2003.

[64] L. van Harten, R. Jordans y H. Pourshaghaghi, "Necessity of fault tolerance techniques in xilinx kintex 7 fpga devices for space missions: A case study," en *2017 Euromicro Conference on Digital System Design (DSD)*, páginas 299–306, Aug 2017.

[65] Souaad, Manzar, R. B. A. Rahim, S. F. Sabri y N. F. Hasbullah, "Radiation characteristics and seu rates in neqo environment using spenvis," en *2016 International Conference on Computer and Communication Engineering (ICCCE)*, páginas 454–458, July 2016.

[66] T. Uemura y M. Hashimoto, "Investigation of single event upset and total ionizing dose in feram for medical electronic tag," en *2015 IEEE International Reliability Physics Symposium*, páginas SE.1.1–SE.1.5, April 2015.

[67] P. Goldhagen *et al.*, "Measurement of the energy spectrum of cosmic-ray induced neutrons aboard an er-2 high-altitude airplane," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 476, nº 1, páginas 42 – 51, 2002, int. Workshop on Neutron

Field Spectrometry in Science, Technolog y and Radiation Protection. http://www.sciencedirect.com/science/article/pii/S0168900201013869

[68] S. Rezgui, P. Louris y R. Sharmin, "See characterization of the new rtax-dsp (rtax-d) antifuse-based fpga," *IEEE Transactions on Nuclear Science*, vol. 57, nº 6, páginas 3537–3546, Dec 2010.

[69] C. Urbina-Ortega, G. Furano, G. Magistrati, K. Marinis, A. Menicucci y D. Merodio-Codinachs, "Flash-based fpgas in space, design guidelines and trade-off for critical applications," en *2013 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, páginas 1–8, Sept 2013.

[70] X. Qin *et al.*, "Development of a high resolution tdc for implementation in flash-based and anti-fuse fpgas for aerospace application," *IEEE Transactions on Nuclear Science*, vol. 60, nº 5, páginas 3550–3556, Oct 2013.

[71] Xilinx, "Radiation-hardened, space-grade virtex-5qv fpga data sheet: Dc and ac switching characteristics (v1.3.1)," January 16 2015.

[72] R. Glein *et al.*, "Reliability of space-grade vs. cots sram-based fpga in n-modular redundancy," en *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, páginas 1–8, June 2015.

[73] Y. P. Fang y A. S. Oates, "Characterization of single bit and multiple cell soft error events in planar and finfet srams," *IEEE Transactions on Device and Materials Reliability*, vol. 16, nº 2, páginas 132–137, June 2016.

[74] R. Rajaei, "Radiation-hardened design of nonvolatile mram-based fpga," *IEEE Transactions on Magnetics*, vol. 52, nº 10, páginas 1–10, Oct 2016.

[75] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, nº 6, páginas 2742–2750, Dec 1996.

[76] P. Maillard, M. Hart, J. Barton, P. Jain y J. Karp, "Neutron, 64 mev proton, thermal neutron and alpha single-event upset characterization of xilinx 20nm ultrascale kintex fpga," en *2015 IEEE Radiation Effects Data Workshop (REDW)*, páginas 1–5, July 2015.

[77] D. S. Lee *et al.*, "Single-event characterization of the 20 nm xilinx kintex ultrascale field-programmable gate array under heavy ion irradiation," en *2015 IEEE Radiation Effects Data Workshop (REDW)*, páginas 1–6, July 2015.

[78] D. S. Lee, M. Wirthlin, G. Swift y A. C. Le, "Single-event characterization of the 28 nm xilinx kintex-7 field-programmable gate array under heavy

ion irradiation," en *2014 IEEE Radiation Effects Data Workshop (REDW)*, páginas 1–5, July 2014.

[79] L. D. Edmonds y F. Irom, "Extension of a proton seu cross section model to include 14 mev neutrons," *IEEE Transactions on Nuclear Science*, vol. 55, nº 1, páginas 649–655, Feb 2008.

[80] L. D. Edmonds, "Proton seu cross sections derived from heavy-ion test data," *IEEE Transactions on Nuclear Science*, vol. 47, nº 5, páginas 1713–1728, Oct 2000.

[81] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael y P. Alfke, "The rosetta experiment: atmospheric soft error rate testing in differing technology fpgas," *IEEE Transactions on Device and Materials Reliability*, vol. 5, nº 3, páginas 317–328, Sept 2005.

[82] D. G. Mahmoud *et al.*, "Fault secure fpga-based tmr voter," en *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, páginas 1–4, June 2018.

[83] Xilinx, "Xilinx tmrtool user guide version 13.2 (v3.1.2)," June 23 2017.

[84] M. Cannon, A. Keller y M. Wirthlin, "Improving the effectiveness of tmr designs on fpgas with seu-aware incremental placement," en *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, páginas 141–148, April 2018.

[85] H. Wei, W. Yueke, X. Kefei y D. Wei, "Single event effect vulnerability analysis and on-orbit error rate prediction," en *2016 IEEE International Conference on Signal and Image Processing (ICSIP)*, páginas 471–477, Aug 2016.

[86] X. She y N. Li, "Reducing critical configuration bits via partial tmr for seu mitigation in fpgas," *IEEE Transactions on Nuclear Science*, vol. 64, nº 10, páginas 2626–2632, Oct 2017.

[87] A. J. Sanchez-Clemente, L. Entrena y M. Garcia-Valderas, "Partial tmr in fpgas using approximate logic circuits," *IEEE Transactions on Nuclear Science*, vol. 63, nº 4, páginas 2233–2240, Aug 2016.

[88] B. Pratt, M. Caffrey, P. Graham, K. Morgan y M. Wirthlin, "Improving fpga design robustness with partial tmr," en *2006 IEEE International Reliability Physics Symposium Proceedings*, páginas 226–232, March 2006.

[89] A. Sari y M. Psarakis, "Scrubbing-aware placement for reliable fpga sys-

tems," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, nº 99, páginas 1–1, 2017.

[90] F. Siegle, T. Vladimirova, J. Ilstad y O. Emam, "Mitigation of radiation effects in sram-based fpgas for space applications," *ACM Comput. Surv.*, vol. 47, nº 2, páginas 37:1–37:34, 2015. http://doi.acm.org/10.1145/2671181

[91] A. Ahmed, "New fpga blind scrubbing technique," en *2016 IEEE Aerospace Conference*, páginas 1–9, March 2016.

[92] K. A. Hoque, O. A. Mohamed, Y. Savaria y C. Thibeault, "Probabilistic model checking based dal analysis to optimize a combined tmr-blind-scrubbing mitigation technique for fpga-based aerospace applications," en *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*, páginas 175–184, Oct 2014.

[93] A. M. Keller, T. A. Whiting, K. B. Sawyer y M. J. Wirthlin, "Dynamic seu sensitivity of designs on two 28-nm sram-based fpga architectures," *IEEE Transactions on Nuclear Science*, vol. 65, nº 1, páginas 280–287, Jan 2018.

[94] P. S. Ostler *et al.*, "Sram fpga reliability analysis for harsh radiation environments," *IEEE Transactions on Nuclear Science*, vol. 56, nº 6, páginas 3519–3526, Dec 2009.

[95] X. Iturbe, M. Azkarate, I. Martinez, J. Perez y A. Astarloa, "A novel seu, mbu and she handling strategy for xilinx virtex-4 fpgas," en *2009 International Conference on Field Programmable Logic and Applications*, páginas 569–573, Aug 2009.

[96] J. Tonfat, F. L. Kastensmidt, P. Rech, R. Reis y H. M. Quinn, "Analyzing the effectiveness of a frame-level redundancy scrubbing technique for sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 62, nº 6, páginas 3080–3087, Dec 2015.

[97] F. Brosser, E. Milh, V. Geijer y P. Larsson-Edefors, "Assessing scrubbing techniques for xilinx sram-based fpgas in space applications," en *2014 International Conference on Field-Programmable Technology (FPT)*, páginas 296–299, Dec 2014.

[98] H. Asadi y M. B. Tahoori, "Analytical techniques for soft error rate modeling and mitigation of fpga-based designs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, nº 12, páginas 1320–1331, Dec 2007.

[99] B. Schmidt, D. Ziener y J. Teich, "Minimizing scrubbing effort through

automatic netlist partitioning and floorplanning," en *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, páginas 299–304, May 2014.

[100] A. Sari y M. Psarakis, "Scrubbing-based seu mitigation approach for systems-on-programmable-chips," en *2011 International Conference on Field-Programmable Technology*, páginas 1–8, Dec 2011.

[101] G. L. Nazar, L. P. Santos y L. Carro, "Accelerated fpga repair through shifted scrubbing," en *2013 23rd International Conference on Field programmable Logic and Applications*, páginas 1–6, Sept 2013.

[102] G. L. Nazar, L. P. Santos y L. Carro, "Fine-grained fast field-programmable gate array scrubbing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, nº 5, páginas 893–904, May 2015.

[103] J. Y. Lee *et al.*, "Heterogeneous configuration memory scrubbing for soft error mitigation in fpgas," en *2012 International Conference on Field-Programmable Technology*, páginas 23–28, Dec 2012.

[104] A. Stoddard, A. Gruwell, P. Zabriskie y M. J. Wirthlin, "A hybrid approach to fpga configuration scrubbing," *IEEE Transactions on Nuclear Science*, vol. 64, nº 1, páginas 497–503, Jan 2017.

[105] V. D. Agrawal, C. R. Kime y K. K. Saluja, "A tutorial on built-in self-test. i. principles," *IEEE Design Test of Computers*, vol. 10, nº 1, páginas 73–82, Mar 1993.

[106] M. Rozkovec, J. Jenicek y O. Novak, "An evaluation of the application dependent fpga test method," en *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, páginas 22–25, April 2012.

[107] M. Abramovici, C. E. Stroud y J. M. Emmert, "Online bist and bist-based diagnosis of fpga logic blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, nº 12, páginas 1284–1294, Dec 2004.

[108] M. Abramovici y C. E. Stroud, "Bist-based test and diagnosis of fpga logic blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, nº 1, páginas 159–172, Feb 2001.

[109] M. Tahoori, "Application-dependent testing of fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, nº 9, páginas 1024–1033, Sept 2006.

[110] N. R. Shnidman, W. H. Mangione-Smith y M. Potkonjak, "On-line fault

detection for bus-based field programmable gate arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, nº 4, páginas 656–666, Dec 1998.

[111] A. Agarwal, G. Bhatia y S. Chakraverty, "State model for scheduling built-in self-test and scrubbing in fpga to maximize the system availability in space applications," en *India International Conference on Power Electronics 2010 (IICPE2010)*, páginas 1–7, Jan 2011.

[112] G. I. Alkady, H. H. Amer y R. M. Daoud, "Remotely configurable fault-tolerant fpga-based pacemaker," en *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, páginas 19–24, Dec 2017.

[113] T. S. Nidhin, A. Bhattacharyya, R. P. Behera, T. Jayanthi y K. Velusamy, "Dependable system design with soft error mitigation techniques in sram based fpgas," en *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, páginas 1–6, April 2017.

[114] F. G. de Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro y R. Reis, "Designing fault-tolerant techniques for sram-based fpgas," *IEEE Design Test of Computers*, vol. 21, nº 6, páginas 552–562, Nov 2004.

[115] J. Gomez-Cornejo, A. Zuloaga, U. Kretzschmar, U. Bidarte y J. Jimenez, "Fast context reloading lockstep approach for seus mitigation in a fpga soft core processor," en *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, páginas 2261–2266, Nov 2013.

[116] A. Hanafi, M. Karim y A. E. Hammami, "Dual-lockstep microblaze-based embedded system for error detection and recovery with reconfiguration technique," en *2015 Third World Conference on Complex Systems (WCCS)*, páginas 1–6, Nov 2015.

[117] I. M. Safarulla y K. Manilal, "Design of soft error tolerance technique for fpga based soft core processors," en *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, páginas 1036–1040, May 2014.

[118] J. Chen, X. Liang y Z. Chen, "Online algorithm-based fault tolerance for cholesky decomposition on heterogeneous systems with gpus," en *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, páginas 993–1002, May 2016.

[119] A. Scholl, C. Braun, M. A. Kochte y H. J. Wunderlich, "Efficient algorithm-based fault tolerance for sparse matrix operations," en *2016 46th Annual*

IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), páginas 251–262, June 2016.

[120] W. Long, L. Yue, Q. Yanling, X. Tengfei y W. Minhao, "A method of testability analysis and design based on fmea extension," en *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, páginas 361–367, Oct 2017.

[121] D. Ma, Z. Zhou, Y. Jiang y W. Ding, "Constructing bayesian network by integrating fmea with fta," en *2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, páginas 696–700, Sept 2014.

[122] X. Han y J. Zhang, "A combined analysis method of fmea and fta for improving the safety analysis quality of safety-critical software," en *2013 IEEE International Conference on Granular Computing (GrC)*, páginas 353–356, Dec 2013.

[123] M. M. Karimi, A. Anzagira, W. Taylor, J. Nelson y A. Osareh, "Component failure analysis (cfa): A new method for implemented fpga design failure analysis," en *SoutheastCon 2018*, páginas 1–8, April 2018.

[124] H. Wang, B. Hu, H. Zou y Y. Zhou, "Circuit modeling and simulation of cmos circuits latchup induced by microwave pulse injection," en *2017 Progress in Electromagnetics Research Symposium - Fall (PIERS - FALL)*, páginas 2988–2992, Nov 2017.

[125] A. Samaras *et al.*, "Experimental characterization and simulation of electron-induced seu in 45-nm cmos technology," *IEEE Transactions on Nuclear Science*, vol. 61, nº 6, páginas 3055–3060, Dec 2014.

[126] Xilinx, "Soft error mitigation using priorized essential bits xapp538 (v1.0)," April 4 2012.

[127] K. Chapman, "Virtex-5 seu critical bit information extending the capability of the virtex-5 seu controller, xilinx documentation seu lounge,," February 2010.

[128] A. Ullah, P. Reviriego y J. A. Maestro, "An efficient methodology for on-chip seu injection in flip-flops for xilinx fpgas," *IEEE Transactions on Nuclear Science*, vol. 65, nº 4, páginas 989–996, April 2018.

[129] U. Kretzschmar, A. Astarloa, J. LaÅºaro, J. JimeÅ„ez y A. Zuloaga, "An automatic experimental set-up for robustness analysis of designs implemented on sram fpgas," en *International Symposium on System on Chip (SoC)*, páginas 96–101, Oct 2011.

[130] Xilinx, "Logicore ip soft error mitigation controller v4.0, pg036," June 19 2013.

[131] Xilinx, "Isolation design flow xilinx 7 series fpgas or zynq-7000 ap socs (ise tools) xapp1086 (v1.3.1)," February 5, 2015.

[132] Xilinx, "Seu strategies for virtex-5 devices, xapp864 (v2.0)," April 1 2010.

[133] Xilinx, "Vivado design suite user guide, power analysis and optimization, ug907," April 6, 2016.

[134] A. Lesea, W. Koszek, G. Steiner, G. Switf y D. White, "Soft error study of arm soc at 28 nanometers," en *Proceedings of the IEE Workshop on Silicon Errors in Logic - System Effects*, 2014.

[135] S. on Chip engineering, "User guide for smartzynq family."

[136] I. 62439-3:2016, "Industrial communication networks - high availability automation networks - part 3: Parallel redundancy protocol (prp) and high-availability seamless redundancy (hsr)," 2016.

[137] I. 62439-3:2016, "Industrial communication networks - high availability automation networks - part 3: Parallel redundancy protocol (prp) and high-availability seamless redundancy (hsr)," 2016.

[138] L. Sterpone y M. Violante, "A new algorithm for the analysis of the mcus sensitiveness of tmr architectures in sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 55, nº 4, páginas 2019–2027, Aug 2008.

[139] H. Quinn, P. Graham, J. Krone, M. Caffrey y S. Rezgui, "Radiation-induced multi-bit upsets in sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 52, nº 6, páginas 2455–2461, Dec 2005.

[140] H. Quinn, K. Morgan, P. Graham, J. Krone y M. Caffrey, "Static proton and heavy ion testing of the xilinx virtex-5 device," en *2007 IEEE Radiation Effects Data Workshop*, vol. 0, páginas 177–184, July 2007.

[141] A. Manuzzato, S. Gerardin, A. Paccagnella, L. Sterpone y M. Violante, "On the static cross section of sram-based fpgas," en *2008 IEEE Radiation Effects Data Workshop*, páginas 94–97, July 2008.

[142] B. L. Bhuva *et al.*, "Multi-cell soft errors at advanced technology nodes," *IEEE Transactions on Nuclear Science*, vol. 62, nº 6, páginas 2585–2591, Dec 2015.