UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de
Ingeniería y Sistemas de Telecomunicación

PROYECTO FIN DE GRADO

ELABORACIÓN DE UNA METODOLOGÍA DE ANÁLISIS DE SEEs (Single Event Effects) EN SoCs PARA APLICACIONES AEROESPACIALES

INAN ILIK ILIK

Grado en Ingeniería Electrónica de Comunicaciones
Junio 2024

JUNIO
2024

ELABORACIÓN DE UNA METODOLOGÍA DE ANÁLISIS DE SEEs (Single Event Effects) EN SoCs PARA APLICACIONES AEROESPACIALES

INAN ILIK ILIK

PROYECTO FIN DE GRADO

TÍTULO: Elaboración de una Metodología de Análisis de SEEs (Single Event Effects) en SoCs para Aplicaciones Aeroespaciales

AUTOR/A: Inan Ilik Ilik

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

DIRECTOR/A: Luis Cárdenas González

TUTOR/A: María Pilar Ochoa Pérez

DEPARTAMENTO: ELECTRÓNICA FÍSICA, INGENIERÍA ELÉCTRICA Y FÍSICA APLICADA

VºBº TUTOR/A

Firmado por OCHOA PEREZ MARIA PILAR - ***5429** el día 11/06/2024 con un certificado emitido por AC FNMT Usuarios

Miembros del Tribunal Calificador:

PRESIDENTE/A: María Luisa López Ibáñez

TUTOR/A: María Pilar Ochoa Pérez

SECRETARIO/A: Laura Barrutia Poncela

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Resumen

El presente proyecto aborda el estudio de SEEs (Single Event Effects – Efectos de Evento Único) generados por la radiación ionizante en SoCs (Systems On Chip – Sistemas en Chip) para aplicaciones aeroespaciales, enfocándose en el análisis de fallos a nivel de equipamiento.

Primero, se describen las características generales del dispositivo electrónico investigado: el SoC de la familia Zynq 7000 de Xilinx, que es el núcleo central de la placa de desarrollo Blackboard, fabricada por RealDigital. Componentes fundamentales del SoC incluyen la FPGA (Field Programmable Gate Array – Matriz de Compuertas Programable en Campo) de la familia Artix-7 y el Microprocesador ARM Cortex-A9.

Se expone la estructura interna de las FPGAs, considerando su naturaleza como dispositivos semiconductores de silicio, lo cual sienta las bases teóricas para comprender los efectos de la radiación atmosférica en estos dispositivos. Se presenta la composición interna de la FPGA investigada, destacando sus singularidades.

Una vez presentada la arquitectura de la FPGA, se exponen los diversos tipos de SEEs y particularmente aquellos que ejercen un impacto significativo sobre los dispositivos basados en silicio. Este análisis culmina con un enfoque detallado en los SEUs (Single Event Upset – Perturbación o Alteración de Evento Único o Singular), pertenecientes a la categoría de errores transitorios conocidos como “errores suaves”, los cuales, a pesar de su naturaleza disruptiva, no ocasionan daños permanentes en el dispositivo, pero sí en el diseño que se implementa dando lugar a errores funcionales.

Para la programación del SoC se utilizan las plataformas Vivado Design Suite y Vitis IDE proporcionadas por Xilinx. Vivado se emplea para el diseño hardware en la FPGA, y Vitis para el programa software en el Microprocesador.

Se desarrollan dos diseños diferentes para analizar los efectos mencionados. El primero es una metodología de análisis de SEUs utilizando el bloque IP (*Intellectual Property* – Propiedad Intelectual) llamado SEM (Soft Error Mitigation – Mitigación de “Errores Suaves”) IP de Xilinx, que encapsula código VHDL. Este bloque se usa para inyecciones controladas de errores, alterando uno o más bits, lo que replica los SEUs y permite estudiar su comportamiento y las respuestas del sistema.

El segundo diseño representa una elaboración más refinada y complementaria del primero que introduce una metodología de análisis de los SEUs en el que se crea un Inyector de Fallos que consiste una Puerta XOR (eXclusive OR – O Exclusivo). Esta configuración se encarga de seleccionar de forma precisa el bit a alterar dentro de un conjunto de muestras procesadas de una señal sinusoidal, utilizando un bloque de Transformada de Fourier Rápida (FFT – Fast Fourier Transform). Tras la incidencia de la alteración y la subsiguiente recuperación o restauración de la señal sinusoidal original mediante un proceso inverso de FFT, se pretende examinar el impacto final que causa dicho error por medio de una simulación en la plataforma

Vivado. En este contexto particular, aunque no se ha utilizado el dispositivo electrónico para la implementación física, es pertinente destacar que sería plenamente factible integrar este diseño en la FPGA, demostrando así su funcionalidad; no obstante, esta aplicación práctica excede el alcance del objeto de estudio de este proyecto.

Abstract

The present project addresses the study of SEEs (Single Event Effects) caused by ionizing radiation in SoCs (Systems On Chip) for aerospace applications, focusing on the analysis of equipment-level failures.

First, the general characteristics of the investigated electronic device are described: the Zynq 7000 family SoC from Xilinx, which is the central core of the Blackboard development board, manufactured by RealDigital. Fundamental components of the SoC include the Artix-7 family FPGA (Field Programmable Gate Array) and the ARM Cortex-A9 microprocessor.

The internal structure of the FPGAs is presented, considering their nature as silicon-based semiconductor devices, which provides the theoretical basis for understanding the effects of atmospheric radiation on these devices. The internal composition of the investigated FPGA is presented, highlighting its unique features.

Once the FPGA architecture is presented, the various types of SEEs are exposed, particularly those that have a significant impact on silicon-based devices. This analysis culminates in a detailed focus on SEUs (Single Event Upset), belonging to the category of transient errors known as "soft errors," which, despite their disruptive nature, do not cause permanent damage to the device but can result in functional errors in the implemented design.

For programming the SoC, the Vivado Design Suite and Vitis IDE platforms provided by Xilinx are used. Vivado is employed for hardware design in the FPGA, and Vitis for the software program in the microprocessor.

Two different designs are developed to analyze the mentioned effects. The first is a methodology for analyzing SEUs using the SEM (Soft Error Mitigation) IP block from Xilinx, which encapsulates VHDL code. This block is used for controlled error injections, altering one or more bits, which replicates SEUs and allows for studying their behavior and the system's responses.

The second design is a more refined and complementary version of the first, introducing a methodology for analyzing SEUs by creating a Fault Injector consisting of an XOR Gate (eXclusive OR). This configuration precisely selects the bit to alter within a set of samples processed from a sinusoidal signal using a Fast Fourier Transform (FFT) block. After the alteration and subsequent recovery or restoration of the original sinusoidal signal through an inverse FFT process, the final impact of the error is examined through simulation on the Vivado platform. In this context, although the electronic device has not been used for physical implementation, it is pertinent to highlight that integrating this design into the FPGA would be fully feasible, thus demonstrating its functionality; however, this practical application exceeds the scope of this project's study.

Índice de contenidos

Índice de figuras

Índice de tablas

Lista de acrónimos

ADC	Analog to Digital Converter
AMBA	Advanced Microcontroller Bus Architecture
ARM	Advanced RISC Machines
ASICs	Application Specific Integrated Circuits
ASCII	American Standard Code for Information Interchange
AXI	Advanced eXtensible Interface
BIT	Binary Digit
BRAM	Block Random Access Memory
BUFIO	Clock Buffer Input/Output
BUFG	Global Buffer Clock
BUFR	Clock Buffer Register
CIN	Carry Input
CLB	Configurable Logic Blocks
CMOS	Complementary Metal Oxide Semiconductor
COTS	Component Off-The-Shelf
CRC	Cyclic Redundancy Check
CRAM	Configuration Random Access Memory
CPLD	Complex Programmable Logic Devices
COUT	Carry Output
DDS	Direct Digital Synthesizer
DMA	Direct Memory Access
DUT	Device Under Test
DSP	Digital Signal Processing
EBD	Essential Bit Data
EBC	Essential Bit Configuration
ECC	Error Correction Code
FDC	Fin De Cuenta
FFT	Fast Fourier Transform
FIT	Failure In Time
FPGA	Field Programmable Gate Array
FPU	Float Point Unit
GPIO	General Purpose Input Output
GCR	Galactic Cosmic Rays
HDMI	High Definition Multimedia Interface
HDL	Hardware Description Language
HE	Hard Error
HPL	High Performance Low-power
HKMG	High-K Metal Gate
IC	Integrated Circuit
ICAP	Interconnect Configuration Access Port

IEL	Ionizing Energy Loss
IDE	Integrated Development Environment
I/O	Input/Output
iFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
JTAG	Joint Test Access Group
LANSCE	Los Alamos Neutron Science Center
LED	Light Emitting Diode
LET	Linear Energy Transfer
LFA	Linear Frame Address
LUT	Look-Up-Tables
MBU	Multiple Bit Upset
MEMS	Micro ElectroMechanical System
MinGW64	Minimalist GNU for Windows 64 bits
MGT	Multi-Gigabit-Transceiver
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
NMOS	N-type Metal Oxide Semiconductor
OTG	On-The-Go
PCAP	Processor Configuration Access Port
PMOD	Peripheral Module
PMOS	P-type Metal Oxide Semiconductor
PL	Programmable Logic
PLD	Programmable Logic Devices
PWM	Pulse Width Modulation
PDM	Pulse Density Modulation
PS	Processing System
PS_SRST	Processing System – System Reset
QSPI	Quad Serial Peripheral Interface
RAM	Random Access Memory
RC	RC Circuit
RISC	Reduced Instruction Set Computer
SD	Secure Digital
SDIO	Secure Digital Input Output
SDR	Single Data Rate
SED	Single Event Disruption
SEE	Single Event Effect
SE	Soft Error
SECDED	Single Error Correction Double Error Detection
SEL	Single Event LatchUp
SEU	Single Event Upset
SEB	Single Event BurnOut
SEFI	Single Event Functional Interrupt

SEGR	Single Event Gate Rupture
SET	Single Event Transient
SIMD	Single Instruction Multiple Data
SLR	Super Logic Region
SoC	System on Chip
SPI	Serial Peripheral Interface
SPLD	Simple Programmable Logic Devices
SSI	Stacked Silicon Interconnect
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy
TMAC	Tera Multiply-Accumulate
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Synchronous Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
WiFi	Wireless Fidelity
XOR	eXclusive OR
XSA	Xilinx Support Archive

1 Introducción

1.1 Marco y motivación del proyecto

El presente proyecto se ubica en el marco tecnológico de la electrónica de semiconductores, crucial para el desarrollo y funcionamiento de los SoCs (*System On Chip* – Sistema en Chip), que integran como componentes principales: una FPGA y; un microprocesador. Este estudio se centra en aplicaciones aeroespaciales, donde la exposición a la radiación ionizante plantea retos considerables en términos de fiabilidad y disponibilidad de estos sistemas críticos.

El núcleo del proyecto se centra en la elaboración de una metodología que permita establecer un marco de análisis de los efectos de SEEs que experimentan los dispositivos mencionados, haciendo un mayor énfasis en la FPGA. Al existir un amplio abanico de SEEs, se opta establecer una metodología que satisfaga, concretamente, el análisis de los SEUs (*Single Event Upset* – Perturbación de Evento Singular o Único) que, en la mayoría de los casos, es el efecto que exhiben los componentes internos del dispositivo semiconductor en forma de alteraciones o cambios de estado de un único bit, efecto denominado SBU (*Single Bit Upset*), o de múltiples bits, MBU (Multiple Bit Upset – Alteración de Múltiples Bits). Se tratan de errores transitorios, es decir, aquellos que no generan daño permanente en la circuitería del dispositivo, pero sí contribuyen a errores funcionales u operacionales cuando van embarcados en vehículos espaciales.

La primera de las metodologías elaboradas se lleva a cabo mediante una tecnología desarrollada por el fabricante de dispositivos semiconductores Xilinx llamada Controlador SEM IP (*Soft Error Mitigation Intellectual Property* – Propiedad Intelectual Mitigación de Errores “Suaves” o “Blandos”) que presenta en su catálogo de IPs presente en una de sus herramientas *software* de programación llamado Vivado, programa que se ha empleado para llevar a cabo el diseño del hardware mediante la interconexión de otros bloques IP que se implementarán en la FPGA, además de otro programa llamado Vitis, que permite realizar la programación del código fuente que se ejecutará en el microprocesador de arquitectura ARM Cortex-A9 que internamente se comunicará con la FPGA. Esta metodología consiste en realizar inyecciones de errores en direcciones de la Memoria de Configuración RAM (*Random Access Memory* – Memoria de Acceso Aleatorio) que alberga la información necesaria para el correcto funcionamiento del diseño con el fin de alterar bits críticos que comprometa la funcionalidad del dispositivo y se requiera reiniciarlo. De esta forma, se busca aquellos recursos o regiones de la FPGA que presenten una mayor susceptibilidad frente a las inyecciones de errores controladas sin la posibilidad de recuperar o reestablecer la información alterada. Esta manera de proceder es útil para restringir ciertas áreas de la FPGA o abordarlas mediante técnicas de mitigación, estudio que se escapa del alcance de este proyecto.

La segunda metodología, llevada a cabo mediante las mismas herramientas de Vivado, pero sin utilizar la tecnología SEM IP, sino que, se elabora un diseño propio mediante un Inyector de Errores basado en una Puerta XOR en combinación con otros recursos ya disponibles en la

plataforma de desarrollo Vivado. Esta metodología surge como una necesidad de complementar de manera visual el efecto que tendrían las inyecciones de errores y la posterior alteración de los bits que contienen la información de una señal sinusoidal que tras pasar por un proceso de Transformada Rápida de Fourier (FFT – *Fast Fourier Transform*) y ser recuperada mediante el proceso inverso, iFFT.

1.2 Objetivos técnicos y académicos

Este proyecto tiene como finalidad el diseño de una metodología de análisis para la evaluación de los SEEs, concretamente, SEUs en SoCs utilizados en aplicaciones aeroespaciales. Dada la naturaleza exigente de los ambientes aeroespaciales, principalmente por la exposición a radiación ionizante, los dispositivos deben ser diseñados teniendo en cuenta dicho factor. Se han elaborado dos metodologías distintas que se describen de forma introductoria a continuación:

- **El desarrollo de una metodología analítica robusta**: Crear un marco de análisis sistemático para estudiar los efectos de los SEEs, con un enfoque particular en los SEUs y sus subtipos, como el SBU y MBU. Esto implicará la configuración de experimentos controlados y la simulación de eventos singulares para evaluar la respuesta de los dispositivos a variaciones específicas que podrían experimentar bajo condiciones de radiación.
- **Aplicación de la tecnología software SEM IP (Soft Error Mitigation Intellectual Property – Propiedad Intelectual para Mitigación de Errores Transitorios o "Suaves") del fabricante Xilinx**: Se considera la implementación de la tecnología de propiedad intelectual SEM IP de Xilinx, específicamente diseñada para la mitigación de errores transitorios. Este enfoque técnico permite la inyección controlada de errores — particularmente la alteración de un bit— en una región aleatoria de la Memoria de Configuración del FPGA. Posteriormente, la misma tecnología se encarga de detectar y corregir este error.

  Es fundamental destacar que, cuando dicho error afecta regiones críticas del dispositivo, puede ocasionar la pérdida de información crucial relativa al circuito diseñado sobre el FPGA. Esto, a su vez, puede desencadenar la necesidad de una reinicialización completa del dispositivo, debido a la pérdida de datos esenciales sobre la circuitería interna, resultando en la interrupción del funcionamiento del software implementado sobre el hardware.

  Este análisis sobre la tecnología SEM IP subraya con claridad la necesidad imperiosa de establecer estrategias de mitigación de errores para garantizar la fiabilidad y continuidad operativa de sistemas críticos en ambientes adversos, como los característicos de las aplicaciones espaciales. La meticulosa implementación de estas tecnologías es esencial para prevenir fallos sistemáticos que podrían provocar consecuencias catastróficas en misiones aeroespaciales. Cabe destacar que, en el

contexto de este proyecto, no se despliega una estrategia activa de mitigación; más bien, se enfatiza la vulnerabilidad del sistema frente a los SEUs para ilustrar la urgencia de desarrollar tales estrategias.

- **Empleo de una puerta lógica XOR para simular un SEU**: Esta metodología consiste en la introducción controlada de errores mediante la manipulación de una posición aleatoria de la memoria con dicha puerta lógica. Esta técnica es esencial para recrear las condiciones provocadas por los SEUs, facilitando la investigación y evaluación de su impacto en el funcionamiento del circuito interno del SoC.

  Además, en el proyecto actual, la puerta XOR se implementa para incidir sobre un circuito de Transformada Rápida de Fourier (FFT) que opera activamente dentro de la FPGA. Un SEU que altera un bit específico en la memoria de configuración de este circuito FFT puede causar un efecto disruptivo significativo: se traduce en la introducción de un tono espurio dentro de la señal legítima que el circuito está procesando, en este caso, una onda sinusoidal. Esta alteración no sólo afecta la precisión de la FFT, sino que también puede comprometer la integridad de la señal procesada, resaltando así, de nuevo, la importancia crítica de robustas estrategias de mitigación y corrección de errores para mantener la funcionalidad adecuada del sistema en condiciones adversas.

Desde el punto de vista académico, los conocimientos y habilidades adquiridos son:

- **Destreza en diseño y simulación de circuitos electrónicos**: A través del uso intensivo de herramientas de diseño como Vivado y Vitis, se han adquirido competencias avanzadas en la creación, prueba y optimización de diseños de circuitos integrados, especialmente en lo que respecta a la mitigación de efectos adversos como los SEUs, haciendo uso de tecnologías más complejas como es el caso de los SoCs.
- **Capacidad para el análisis técnico y científico**: Se precisó de la lectura de numerosas documentaciones técnicas, publicaciones científicas y acceso a foros científicos para poder implementar un diseño adaptado a las necesidades específicas proyecto teniendo en cuenta la necesidad de establecer un marco teórico y práctico para darle un contexto.

1.3 Estructura del resto de la memoria

La estructura de la memoria a partir de la introducción se enfoca en detallar, en el capítulo 2, el marco tecnológico del que forma parte esta investigación exponiendo aspectos como los comienzos de la detección e investigación de los SEEs y los tipos de radiación que son causa de estos fenómenos, además de introducir una breve clasificación. Se presentan también técnicas de mitigación de errores ya existentes y un diagrama de causalidad que expone mediante un diagrama de estados las causas que conducen a un SEE.

A continuación, en el capítulo 3, se exponen las especificaciones y restricciones a partir de las cuales se desarrolla el diseño tales como crear una metodología propia de análisis de SEEs, la

caracterización de los diferentes tipos de SEEs que tienen lugar en los SoCs, haciendo énfasis en los SEUs, el dispositivo electrónico a emplear y las herramientas software para programarlo. Se detallan más adelante los objetivos que se buscan lograr, tales como: poder usar la tecnología SEM IP en el diseño de la aplicación específica; y crear un Inyector de Fallos propio mediante el empleo de una puerta XOR para simular un SEU que podrá ser observado el efecto resultante en la forma de una onda sinusoidal.

Se mencionan los materiales disponibles como son los dispositivos físicos y herramientas en línea usadas a lo largo de todo el proyecto.

La parte más extensa de este documento es el apartado del desarrollo de la solución propuesta en el capítulo 4 debido a que fue necesario exponer el marco teórico de los SEEs que sirviera de contexto y justificación del diseño práctico que se ha llevado a cabo. Se detallan todos los pasos de los que consta la implementación del diseño basado en el Controlador SEM IP. La necesidad de un entendimiento más profundo de los efectos de los SEEs llevó a la realización de otro diseño complementario que sirve para reforzar de manera visual la forma que adquieren las señales alteradas que, en este caso, se ha optado por emplear una onda sinusoidal ya que permite ver mejor dichas alteraciones cuando el circuito que procesa dicha señal experimenta un SEU.

Finalmente, se detallan los resultados obtenidos a partir de ambos diseños sometidos a experimentación justo a continuación del propio cuerpo de la descripción y otros apartados reservados para el presupuesto total requerido para llevar a cabo este proyecto (ver capítulo 5) y, posteriormente, en el capítulo 6, se describe el impacto que puede llegar a tener. Las conclusiones que se han deducido están expuestas en el capítulo 7 junto con una serie de recomendaciones para futuras investigaciones.

2 Marco tecnológico

Cualquier dispositivo electrónico, por más sofisticado que sea, es susceptible de experimentar fallos bien durante su fabricación, o bien, durante su vida útil, por lo que, ninguno está completamente exento de enfrentar problemas operativos comprometiendo su fiabilidad y disponibilidad en mayor o menor medida. La fiabilidad de un dispositivo electrónico se define como la capacidad para funcionar correctamente durante un período de tiempo específico y bajo ciertas condiciones, sin experimentar fallos o averías. Por otro lado, la disponibilidad se refiere a la medida en que un dispositivo electrónico está listo y accesible para su uso cuando se necesite. En particular, llama especialmente la atención el impacto que la radiación ionizante puede tener sobre la mayoría de los dispositivos semiconductores basados en silicio, como pueden ser microprocesadores, FPGAs (Field Programmable Gate Array), SoCs (System on Chip), circuitos integrados (IC), transistores, entre otros, cuando operan en entornos espaciales. Es importante resaltar que la influencia de la radiación no se limita exclusivamente a altitudes elevadas, sino que también se manifiesta al nivel del mar, aunque solamente las partículas más cargadas alcancen los dispositivos antes mencionados.

Hoy en día, un número importante de satélites adoptan las FPGAs en sus sistemas electrónicos como, por ejemplo, el satélite EUTELSAT QUANTUM en su Unidad de Acondicionamiento de Señales de Geolocalización basado en dos complejas FPGAs [1], que tienen como misión operar en entornos con radiación de alta energía donde, a su vez, la fiabilidad se vuelve un factor crítico.

El estudio de las FPGAs en contraposición a los chips ad-hoc, particularmente en ambientes de alta radiación, revela la superioridad de las primeras en términos de flexibilidad y coste, gracias a su capacidad de reconfiguración y producción en masa. Esto facilita adaptaciones rápidas y ahorros significativos en desarrollo frente a los chips ad-hoc, que demandan procesos de fabricación exclusivos y costosos. No obstante, esta misma naturaleza genérica de las FPGAs las hace menos aptas para soportar condiciones extremas como la radiación espacial o militar, donde los chips ad-hoc, gracias a sus diseños específicos y protecciones especializadas, presentan una ventaja.

El fenómeno de los SEEs se describió por primera vez por WallMark y Marcus en 1962 [2] y reportadas las anomalías de un satélite de comunicaciones en operación de la empresa Hughes Aircraft Company, que experimentaba la activación inesperada de sus circuitos digitales llamando la atención de Binder, Smith y Holdman en 1975 que investigaron las interacciones con los rayos cósmicos galácticos como un mecanismo para varios de estos eventos [3]. Estos reportes fueron ignorados hasta que en 1979 May y Wood informaron sobre perturbaciones causadas por partículas alfa en memorias RAM dinámicas [4], y Pickel y Blanford analizaron en 1984 perturbaciones en circuitos RAM en el espacio debido a los rayos cósmicos de iones pesados [5]. A medida que más problemas relacionados con SEEs en vehículos espaciales eran reportados, este fenómeno fue reconocido como una seria amenaza a las operaciones de los sistemas electrónicos.

Debido a esta susceptibilidad inherente de los dispositivos de silicio a SEEs, el impacto resultante puede manifestarse de forma transitoria o perdurable, dando lugar a daños temporales o permanentes. En la mayoría de los casos, estos eventos no dañan al dispositivo de forma permanente. Los SEEs que generan daños temporales resultan en un error transitorio o error "suave" (SE – Soft Error). En cambio, aquellos que causan un daño permanente en el dispositivo se les denomina error permanente o error "duro" (HE – *Hard Error*).

Existen diferentes tipos de SEEs que se describirán más adelante, pero es de interés para este proyecto la Perturbación de Evento Singular SEU, es decir, aquel evento que causa un cambio temporal en el estado de uno o más bits en la memoria o en los biestables (FFs) del dispositivo empleado.

Los dos tipos de radiación que fueron la causa principal de SEU en dispositivos semiconductores son la radiación de partículas alfa y la radiación de rayos cósmicos. Los neutrones atmosféricos de los rayos cósmicos son, aún en la actualidad, los que lideran los problemas del SEU.

Los Alamos National Laboratory publicó una revista en 2012 llamada La Amenaza del Neutrón Invisible (The Invisible Neutron Threat) [6] donde se expone cómo un neutrón producido por los rayos cósmicos impacta contra un microchip ubicado en el controlador de vuelo de un avión Starlifter C-141B militar, causándole una repentina inclinación hacia la derecha perdiendo el control sobre él. Numerosos hechos similares han ocurrido durante la historia de la industria aeroespacial. Las compañías líderes en el sector propusieron soluciones como, por ejemplo, la redundancia de módulo triple (TMR – Triple Modular Redundancy), técnica con la que se duplica o triplica cada componente crítico del dispositivo; la verificación de redundancia cíclica (CRC – Cyclic Redundancy Check), técnica empleada para verificar la integridad de los datos almacenados en memoria o transmitidos entre diferentes componentes del sistema b ajo estudio; o la técnica que permite detectar y corregir un único error en un conjunto de datos, y detectar la presencia de hasta dos errores en el mismo conjunto de datos (SECDED – Single Error Correction Double Error Detection), entre otras, que no son objeto de estudio de este proyecto.

Existen una variedad de posibles efectos de eventos singulares (SEEs). Estos son importantes ya que pueden causar malfuncionamientos en dispositivos microelectrónicos que operan en el ambiente de radiación ionizante espacial. Los efectos principales son alteraciones o perturbaciones, transitorios y bloqueo, pero también es necesario tener en cuenta otros. Los efectos básicos se muestran en la tabla 1.

Efecto	Característica	Descripción
SEU	Upset – Perturbación	Cambio temporal de la memoria o bit de control.
SET	Transient – Transitorio	El dispositivo se bloquea en estado de alta corriente.
SEL	LatchUp – Bloqueo	El dispositivo se bloquea en estado de alta corriente.
SEB	BurnOut – Quemado	El dispositivo consume alta corriente y se quema.
SEGR	Gate Rupture – Rotura de Compuerta	Destrucción de Compuerta en MOSFETs de potencia.
SEFI	Functional Interrupt – Interrupción Funcional	Ruta de control corrompida por alteración.
MBU	MultiBit Upset – Perturbación de Múltiples bits	Varios bits se alteran por el mismo evento.

Tabla 1. Tipos básicos de SEEs

En la figura 1 se muestra una ruta de causalidad para los efectos de la radiación cósmica en dispositivos electrónicos y cómo los sistemas responden a estos eventos.

El diagrama ilustra el proceso por el cual la radiación cósmica afecta a los semiconductores y cómo esto puede causar distintos tipos de errores en los dispositivos electrónicos. Inicia con la interacción de partículas radiactivas que causan ionización y desplazamientos en la red cristalina del semiconductor. Esta actividad puede llevar a la acumulación de cargas que desencadenan errores transitorios (soft errors) o permanentes (hard errors) en los componentes electrónicos. Los errores suaves generalmente se pueden corregir con medidas como la redundancia o la reconfiguración, mientras que los errores duros pueden requerir pasos más drásticos como el reemplazo de componentes dañados. El diagrama destaca la importancia de diseñar sistemas electrónicos con estrategias de respuesta eficaces para mitigar el impacto de la radiación en entornos propensos a ella, como el espacio.

En el capítulo de la Descripción de la solución propuesta se expondrá detalladamente la técnica o metodología adoptada para analizar los efectos como respuesta a los SEUs en un SoC, con el objetivo de comprender su impacto en la fiabilidad y el rendimiento del sistema.

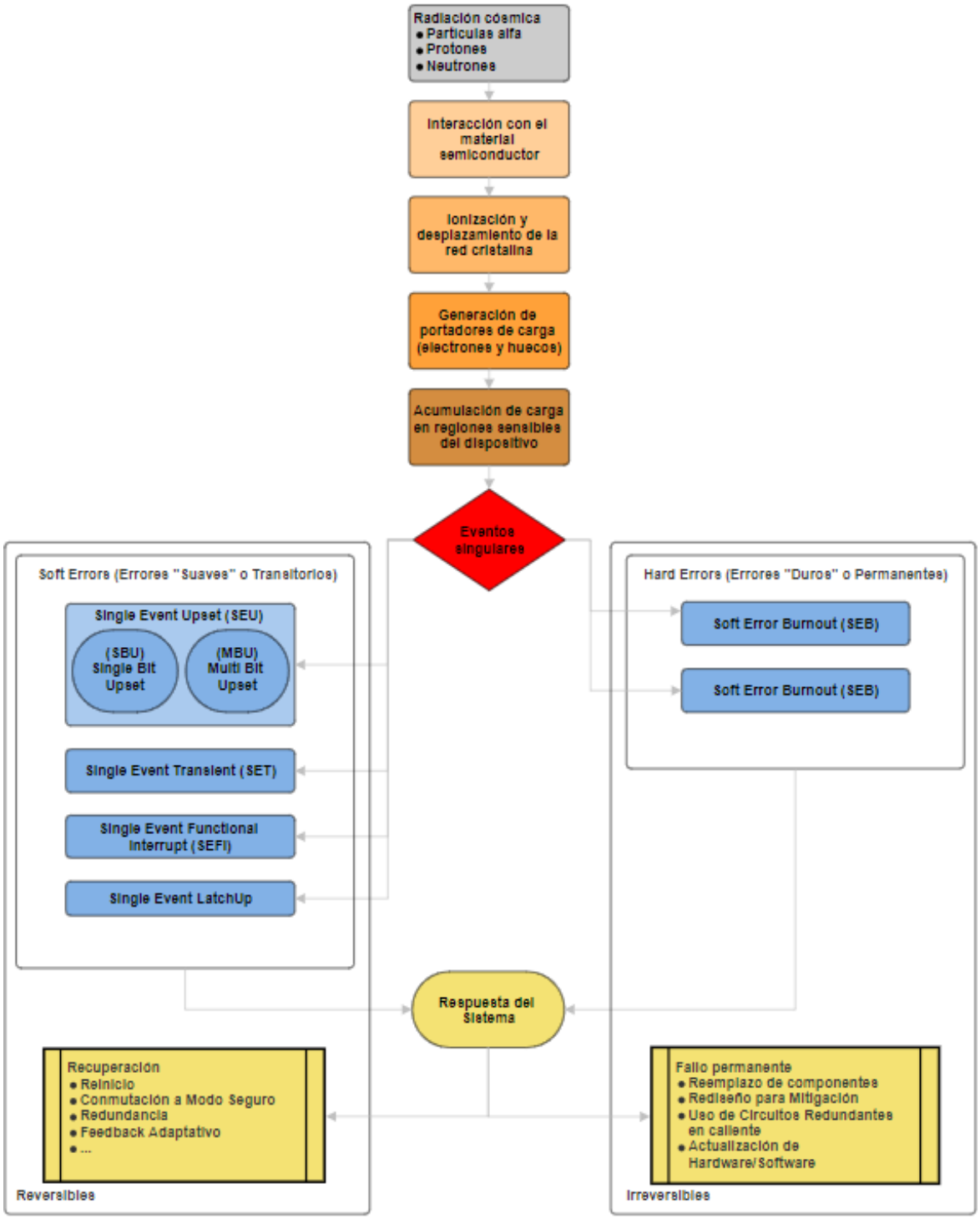


Figura 1. Diagrama de ruta de causalidad para SEEs en dispositivos semiconductores

3 Especificaciones y restricciones de diseño

Es primordial mencionar que esta investigación se llevará a cabo bajo condiciones simuladas y no involucrará experimentación directa en entornos de alta radiación como el espacio. Esta limitación, aunque reduce la autenticidad del entorno de prueba, es necesaria para una manipulación segura y práctica de los componentes electrónicos en estudio. Además, esta aproximación permite una mayor flexibilidad en la modificación y repetición de los experimentos, elementos cruciales para un estudio académico riguroso. El diseño estará regido bajo las siguientes especificaciones y requisitos:

a) **Desarrollar una metodología para el análisis de SEEs en sistemas empotrados**: La metodología propuesta para la caracterización y análisis de SEEs en sistemas embebidos, especialmente en dispositivos como el SoC de la familia Zynq 7000 de Xilinx será un protocolo que seguir (campaña de inyecciones de errores).
b) **Caracterizar de forma teórica las variantes de SEE en SoCs como SEU, SET y SEFI.**
c) **Llevar a cabo la inyección de fallos o errores mediante un Entorno de Desarrollo Integrado (IDE) como, por ejemplo, Vivado Design Suite y Vitis IDE.**
d) **Realizar un análisis comparativo entre dos escenarios, siendo estos, cuando el SoC no está sometido a ningún SEE y tras exponerlo a dicho efecto:** Este análisis comparativo se logra verificar visualmente en las formas de onda de la segunda metodología en el que se implementa una FFT.
e) **Cuantificar y graficar, si fuera necesario, los resultados obtenidos de ambos escenarios:** En el propio cuerpo del desarrollo de la solución se expondrá de manera detallada los resultados obtenidos. Se cuantificará en caso necesario los resultados de las inyecciones de errores para resaltar cierto detalle relevante, pero se hace constar que no se busca realizar un análisis de principio a fin de la magnitud de todos los errores detectados, sino que más bien, se aspira a la elaboración de una metodología que sea válida para emprender una campaña completa de inyecciones de errores en todas las regiones de la FPGA o sobre ciertas regiones específicas.
f) **La metodología de análisis debe alinearse con el fin de evaluar la fiabilidad**: Si la fiabilidad se definió como la capacidad de un dispositivo para funcionar correctamente durante un período de tiempo específico y bajo ciertas condiciones, sin experimentar fallos o averías, entonces, la metodología de análisis a desarrollar debe ser útil para poder evaluar la fiabilidad del dispositivo. Esta evaluación se debe poder llevar a cabo con la metodología implementada que permitirá identificar aquellas regiones de la FPGA más propensa a experimentar SEEs.

3.1 Materiales disponibles

Para organizar adecuadamente la información sobre los recursos disponibles para el proyecto, es conveniente distinguir claramente entre los materiales de hardware, software y las fuentes de información y soporte a través de Internet. A continuación, se detallan estas tres categorías:

3.1.1 Materiales Hardware

Se listan los materiales físicos disponibles para la ejecución del proyecto:

- **Blackboard de RealDigital**: Se usa la placa de prototipado Blackboard del fabricante RealDigital que integra el SoC Zynq-7000 del fabricante Xilinx. Esta plataforma educativa integra un SoC Zynq-7000 de Xilinx, diseñado específicamente para aplicaciones educativas y experimentales en el campo del diseño de sistemas embebidos. La Blackboard proporciona una excelente base para el aprendizaje práctico y el desarrollo de prototipos, combinando la potencia de un procesador ARM con la flexibilidad de la lógica programable de una FPGA.
- **PC**: Una computadora personal se utiliza como la interfaz principal para el diseño, la simulación, y la programación del SoC a través de los IDEs (Integrated Development Environment – Entorno de Desarrollo Integrado) proporcionados por Xilinx.

3.1.2 Materiales Software

Se listan los materiales no físicos tales como los programas informáticos que suministra el fabricante del propio SoC que es Xilinx.

- **Vivado Design Suite versión 2022.2**: Software (Programa Informático) de Xilinx que proporciona un entorno integrado para el diseño y simulación de FPGAs. Vivado es esencial para la síntesis, implementación y análisis de diseños digitales, facilitando la gestión de proyectos y la integración de módulos de IP.
- **Vitis Unified Software Platform versión 2022.2**: Esta plataforma permite la integración de aplicaciones de alto nivel y modelos de inteligencia artificial directamente en hardware, facilitando la aceleración de aplicaciones mediante FPGAs y otros dispositivos programables. Vitis es crucial para maximizar el rendimiento del hardware disponible sin requerir un profundo conocimiento en programación de bajo nivel.

3.1.3 Recursos de Internet

Internet es una herramienta indispensable para acceder a una vasta comunidad de diseñadores y desarrolladores. Foros y plataformas en línea, donde los usuarios comparten sus problemas, soluciones y técnicas, son recursos invaluables para resolver inquietudes específicas y obtener conocimientos prácticos.

4 Descripción de la solución propuesta

Este proyecto surge debido a la creciente utilización de dispositivos electrónicos digitales en entornos críticos como el espacio y por la importancia de comprender y mitigar los efectos de la radiación en los SoCs utilizados en aplicaciones aeroespaciales, como satélites o aviones comerciales. Esto se debe a que estos dispositivos desempeñan funciones críticas en sistemas de comunicación y navegación, donde la fiabilidad es un factor clave para el éxito de la misión. Además, en los últimos años se está generalizando el uso de electrónica de consumo en sistemas aeroespaciales (componentes COTS, Component Off-The-Self), en lugar de utilizar componentes específicamente diseñados y testeados para operar en capas altas de la atmósfera o en el espacio. Esto supone asumir algún riesgo adicional en cuanto a la fiabilidad de los sistemas; pero introduce reducciones de costes notables y posibilita la utilización de tecnologías más avanzadas.

Tras haber presentado los objetivos del proyecto en la sección 2.1, se procederá a detallar los puntos clave que servirán de base teórica para justificar la metodología de análisis adoptada que será puramente práctica.

4.1 Metodología de análisis de SEE

Se ha elaborado una metodología de análisis de los SEEs, en concreto, de los SEUs que dan lugar a soft errors para un caso de aplicación que se describirá más adelante, empleando un SoC, concretamente, el SoC XC7Z007S de la familia ZYNQ-7000 del fabricante Xilinx, que es el componente central alrededor del cual se ha construido la plataforma de diseño de sistemas digitales llamada Blackboard del fabricante RealDigital. Este SoC está constituido por un microprocesador con arquitectura ARM Cortex-A9 y una FPGA equivalente al modelo Artix de la serie 7, entre otros componentes. Con este análisis, se evaluará la fiabilidad y el desempeño de dicho dispositivo cuando esté sometido a SEUs, pero dicho sometimiento se llevará a cabo mediante herramientas software que el propio fabricante del SoC suministra a sus usuarios. Estas herramientas software, debidamente complementadas con la metodología de análisis que se ha desarrollado en este proyecto, permitirán simular los efectos producidos en condiciones reales de radiación ionizante emulando las condiciones que podrían producirse en un laboratorio especializado o en el espacio durante la vida útil del sistema.

Antes de adentrarse en la descripción de la metodología, es preciso introducir nociones teóricas sobre el comportamiento físico de los dispositivos electrónicos ante la radiación ionizante. La comprensión de estos principios fundamentales proporcionará un marco contextual para el análisis posterior.

Se comenzará por presentar el dispositivo electrónico a emplear, con especial énfasis a la FPGA y su estructura interna para, posteriormente, clasificar los distintos tipos de los SEEs y los efectos que inducen estos sobre la FPGA; se expondrá la fiabilidad como una cualidad de los dispositivos semiconductores propensos a alteraciones; se describirán los entornos de desarrollo integrado (IDE – Integrated Development Environment); se presentará el código de

aplicación para depuración y el circuito hardware sometido a fallos inducidos controlados; y finalmente, se detallará la metodología diseñada para el análisis de los SEEs.

4.2 Blackboard & Zynq 7000

La integración de dispositivos electrónicos en sistemas de cómputo ha sido una empresa continua y evolutiva en la historia de la tecnología. En este contexto, la placa de prototipado Blackboard emerge como una manifestación contemporánea de esta progresión. Esta placa, basada en el System On Chip (SoC) Xilinx XC7007S ZYNQ, de la familia ZYNQ-7000, está diseñada especialmente para propósitos educativos con aplicaciones en el campo de la ingeniería eléctrica y de computación.

La Blackboard se erige como un pilar en la intersección de la lógica programable y la computación de propósito general. El SoC que integra se compone por un procesador ARM Cortex-A9 y una FPGA de la familia Artix, concretamente, Artix-7. La figura 2 ilustra la configuración física de la placa en cuestión, ofreciendo una visión tangible de su construcción y disposición de componentes, que sirve como fundamento para las exploraciones teóricas y prácticas. En la Tabla 2 se enumeran los circuitos y dispositivos.

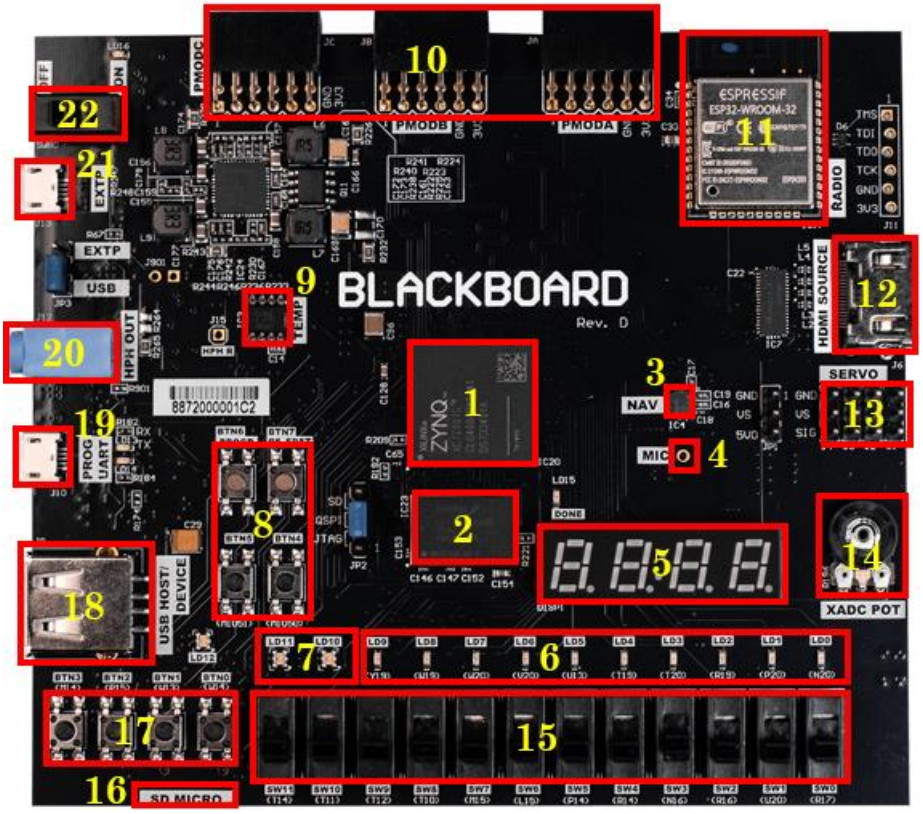


Figura 2. BlackBoard de RealDigital

1	SoC Zynq-7000 (ARM(Advanced RISC Machines) Cortex-A9 & FPGA Artix-7)
2	Memoria Principal DDR3 de 512 MBytes
3	Sensor de navegación de 9 ejes
4	Micrófono MEMS (Sistema Microelectromecánico)
5	Displays de 7 Segmentos de 4 Dígitos
6	10 LEDs Rojos
7	2 LEDs RGB (Los 3 colores primarios)
8	Botones (2 conectados al Sistema de Procesamiento y 2 de Reinicio)
9	Sensor de Temperatura
10	3 PMOD (Peripheral MODule) Expansion Ports
11	Radio de Bluetooth/Wifi
12	Fuente HDMI
13	4 conectores para Servomotores
14	Potenciómetro de 10kOhm (conectado al ADC (Analog-Digital Converter))
15	12 interruptores de deslizamiento conectados al FPGA
16	Conector de tarjeta Micro SD
17	4 botones conectados a la FPGA
18	Puerto USB Host/Dispositivo con OTG
19	Puerto USB de Depuración/Programación
20	Salida de Audio (PDM o PWM)
21	Conector de Alimentación Externa
22	Interruptor de Alimentación

Tabla 2. Dispositivos y Circuitos de la placa Blackboard

Esta plataforma se concibe como una entidad de diseño digital fundamentada en FPGA, también como una entidad basada en microcontrolador de arquitectura ARM, o como una amalgama de ambos, facilitando el alojamiento de diseños software y hardware a medida.

Este dispositivo envuelve al SoC Zynq con un elenco de dispositivos periféricos, comprendiendo una vasta matriz de memoria, dispositivos básicos de entrada/salida y elementos más avanzados como un acelerómetro y un puerto HDMI. Ciertos dispositivos periféricos se manejan directamente desde el sistema procesador, siempre accesibles para el ARM, mientras que otros, vinculados a la FPGA, requieren una configuración previa con circuitos de IP (Intellectual Property – Propiedad Intelectual) personalizados.

La figura 3 es una representación gráfica extraída de la página web del fabricante RealDigital que, a su vez, fue extraída y remodelada del manual de referencia técnico del ZYNQ-7000 de la firma Xilinx [10]. En ella, se despliega con meticulosidad el sistema de procesamiento (PS – Processing System) de la entidad computacional SoC, que exhibe un sistema de núcleo múltiple con capacidades avanzadas como la unidad SIMD (Single Instruction, Multiple Data – Procesamiento de Instrucción Única en Múltiples Datos) de NEON™, que mejora la paralelilazación de tareas: una única operación puede ejecutarse al mismo tiempo para varios datos; la FPU (Float Point Unit – Unidad de Punto Flotante), también de NEON™, es un

componente hardware especializado en realizar operaciones aritméticas con números de punto flotante, esencial para cálculos matemáticos complejos. Se destaca el arreglo jerárquico de interconexiones AMBA (Advanced Microcontroller Bus Architecture – Arquitectura Avanzada de Bus de Mictrocontroladores) que facilita la comunicación entre los diversos subsistemas, incluidos el controlador de caché L2, la memoria integrada, y los dispositivos de temporización y DMA (Direct Memory Access – Acceso Directo a la Memoria). La lógica programable, accesible a través de puertos AXI (Advanced eXtensible Interface) generales y de alto rendimiento, que son interfaces de comunicación diseñadas por ARM dentro de la arquitectura AMBA, que se presenta como el núcleo de flexibilidad del sistema, complementado por transceptores seriales y diversas I/Os (Inputs/Outputs – Entradas/Salidas) de múltiples estándares. Este esquema detallado enfatiza la potencialidad de la máquina para el manejo y la asignación de recursos computacionales en aplicaciones de procesamiento complejas.

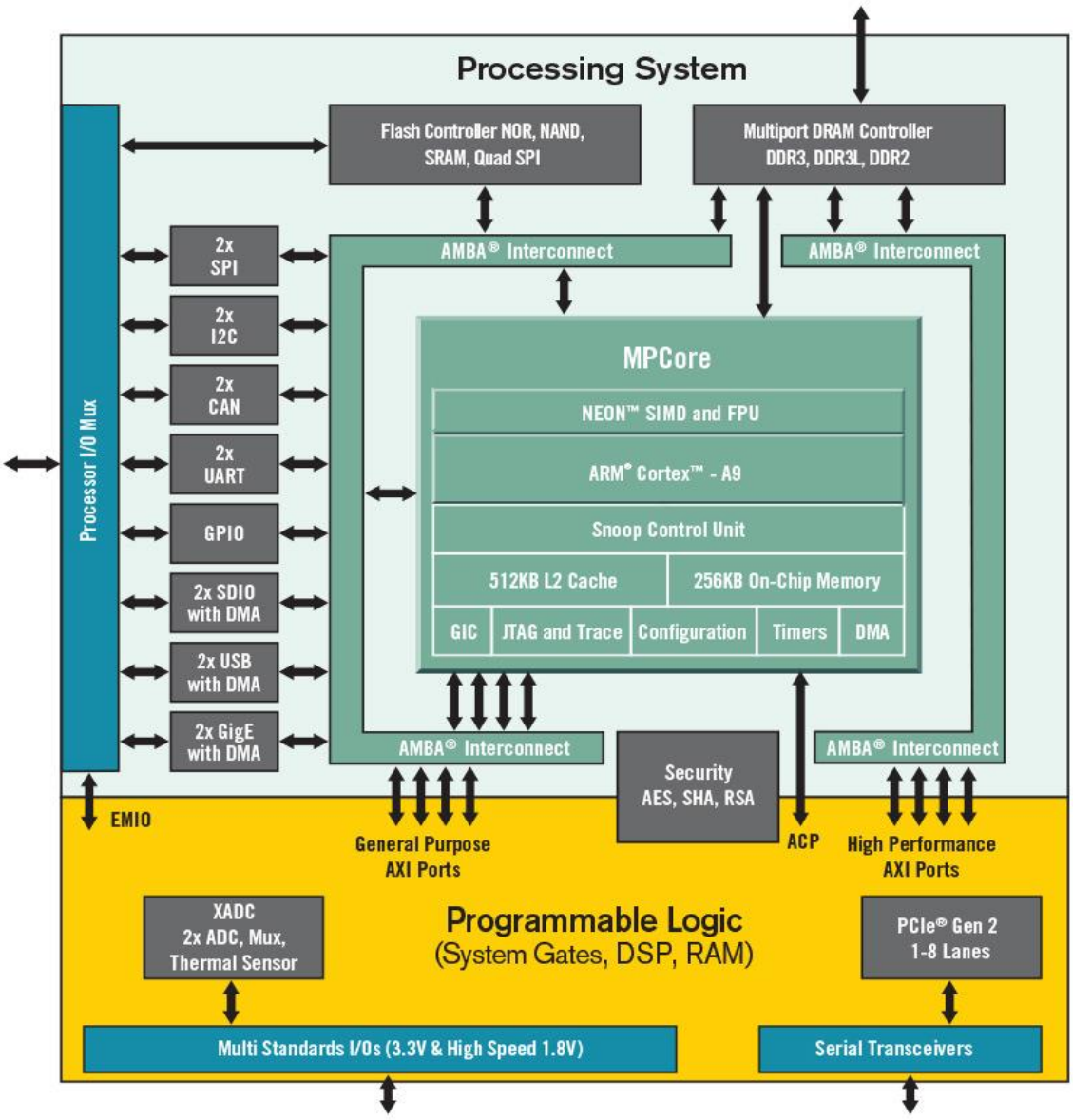


Figura 3. Diagrama de Bloques del SoC de Xilinx Zynq 7000

En contraposición al sistema de procesamiento, pero complementario a este, reside la PL (Programmable Logic – Lógica Programable), equivalente a lo que de aquí en adelante se llamará FPGA, debido a que, es esencialmente una matriz de puertas lógicas, DSP (Digital Signal Processing – Procesado Digital de la Señal) y RAM configurable, que puede ser adaptada para ejecutar operaciones especializadas que el procesador general puede no realizar de manera eficiente.

Cuando se trata realizar aplicaciones de sistemas digitales, la FPGA (PL) y el Microprocesador (PS) se usan conjuntamente donde interaccionan Software y Hardware personalizados, pero cuando la aplicación lo requiera, ambas partes pueden ignorarse entre sí, haciendo únicamente uso de la FPGA o del Microprocesador sin ninguna participación o interferencia entre ambos. En este proyecto, la configuración ha sido del empleo de ambos sistemas en interacción para un caso de aplicación y del uso de la FPGA sola para otro caso de aplicación.

Sirva la figura 4 de ilustración para conocer cómo la PS y la PL se interconectan con el resto de los circuitos y dispositivos presentes en la placa de prototipado Blackboard.

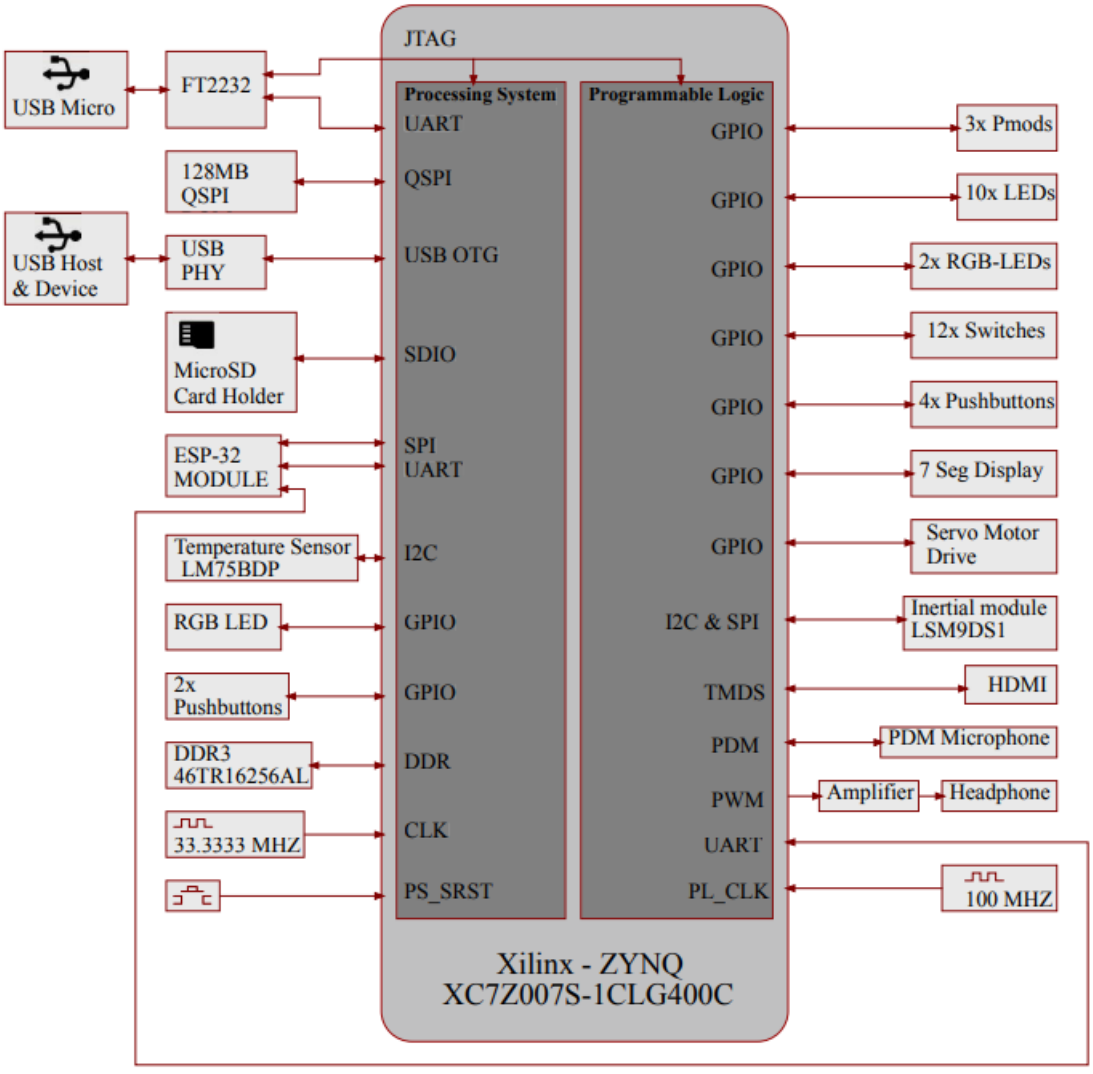


Figura 4. Diagrama de Bloques del Conexionado Global de la Blackboard

El color gris claro indica el área del SoC entero que, a su vez, se divide en dos mitades claramente diferenciadas, por ser de un color gris más oscuro, siendo la mitad izquierda el Sistema de Procesamiento (PS o Microprocesador) y la mitad derecha, está reservada para la Lógica Programable (PL o FPGA).

El sistema de procesamiento incluye interfaces UART (Universal Asyncronous Receiver-Transmitter – Receptor-Transmisor Asíncrono Universal) y QSPI (Quad Serial Peripheral Interface – Interfaz Periférica Serial Cuádruple) para la comunicación y almacenamiento en serio rápido, respectivamente. La presencia de una interfaz USB OTG (On-The-Go – En Movimiento) permite que la placa funcione como host o dispositivo USB (de ahí, "En Movimiento"), ampliando su capacidad de interacción con otros dispositivos USB. También cuenta con interfaces SDIO (Secure Digital Input Output – Entrada Salida Digital Segura) y SPI (Serial Peripheral Interface – Interfaz Periférica Serial) UART para la expansión y comunicación con módulos de memoria y periféricos. El bus I2C (Inter-Integrated Circuit – Circuito Integrado Interconectado) es vital para la comunicación con dispositivos que requieren menos pines y una configuración más sencilla. GPIOs (General Purpose Input Output – Entrada Salida de Propósito General) se utilizan para la entrada y salida de señales digitales genéricas.

En el lado de la PL, se encuentran una matriz de conexiones para GPIOs adicionales, permitiendo una amplia gama de entrada y salidas configurables para propósitos específicos del usuario. Hay conexiones para módulos PMODs (Peripherals Module – Módulo de Periféricos), LEDs (Light Emitting Diode – Diodo Emisor de Luz), interruptores, botones, un Display (pantalla) de 7 segmentos, entre otros. También se observan conexiones para un módulo de servo motor y un módulo inercial LSM9DS1 o acelerómetro, útil para aplicaciones que requieren detección de movimiento o posicionamiento.

Otros componentes periféricos son el módulo ESP-32 para capacidades de conexión inalámbrica, un sensor de temperatura LM75BDP, un amplificador, una interfaz para auriculares y un módulo de memoria DDR3, que proporciona al sistema una cantidad significativa de memoria para tareas de procesamiento complejas. Se puede apreciar también cómo, tanto la "Processing System" como la "Programmable Logic", poseen cada una sus propios dominios de reloj siendo estos CLK de 33.3333 MHz y PL_CLK de 100 MHz, respectivamente. Adicionalmente, la PS tiene conectada un botón PS_SRST (Processing System – System Reset) que permite reiniciar el sistema.

Las FPGAs y los SEEs son el núcleo de este proyecto, por tanto, merecen un apartado propio exponiendo su naturaleza y uno adicional de cómo los SEUs (uno de los tipos de SEEs) alteran la estructura interna del dispositivo mencionado.

4.3 FPGAs

Una FPGA es un dispositivo lógico o chip prefabricado, se constituye sobre la premisa de una rejilla de puertas lógicas, análoga a la configuración presente en matrices de puertas convencionales, con la diferencia fundamental radicando en que la programación de esta

matriz no se efectúa durante la etapa de manufactura, sino en el campo de aplicación mismo, de ahí el término field-programmable (programable en campo de aplicación). Esta singularidad otorga una versatilidad sin par, permitiendo al usuario definir y reconfigurar la funcionalidad de la FPGA en concordancia con las necesidades específicas de la tarea en cuestión.

Con el paso de las décadas, dos ramas del diseño de circuitos integrados (ICs – Integrated Circuits) empezaron a tomar forma: por un lado, los dispositivos lógicos programables (PLDs – Programmable Logic Devices), que permiten a los diseñadores implementar y modificar funciones lógicas después de la fabricación del circuito; por otro lado, los circuitos integrados de aplicación específica (ASICs – Application Specific Integrated Circuits), que están optimizados para tareas específicas y son diseñados y fabricados para realizar una función determinada desde el principio.

En la categoría de PLDs, encontramos los simples (SPLDs – Simple Programmable Logic Devices) y los complejos (CPLDs – Complex Programmable Logic Devices), dispositivos que ofrecen diferentes niveles de complejidad y flexibilidad. Los SPLDs son relativamente simples y útiles para funciones lógicas básicas, mientras que los CPLDs permiten más complejidad en su programación.

Por otro lado, los ASICs ofrecen rendimiento y eficiencia energética superiores para aplicaciones especializadas. Tienen varias subcategorías: las matrices de compuertas (gate arrays) son una forma temprana de ASICs que permiten cierta personalización; los ASICs estructurados (structured ASICs) ofrecen una solución intermedia entre la personalización y el costo; los ASICs de células estándares (standard cell ASICs) ofrecen un equilibrio entre personalización y producción en masa; y finalmente, los ASICs de personalización total (full custom ASICs), que son diseñados completamente a medida para una tarea específica, ofreciendo el máximo rendimiento y eficiencia.

Ahora, la “brecha” que se puede observar en la figura 5, representa la diferencia entre la flexibilidad y el tiempo de comercialización de los PLDs frente a la eficiencia y el rendimiento de los ASICs. Con el tiempo, la tecnología ha intentado cerrar esta brecha, ofreciendo soluciones que combinan lo mejor de ambos mundos, pero aún persiste una distinción clara entre la personalización y el rendimiento optimizado.

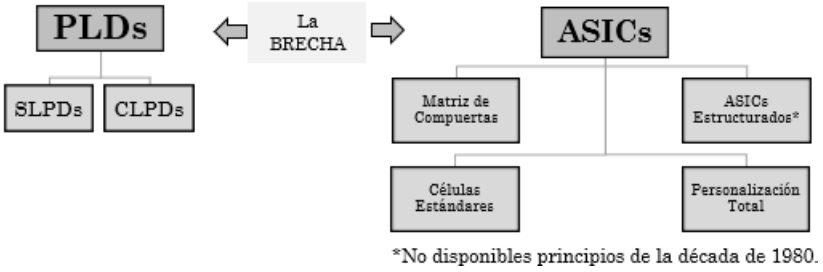


Figura 5. La brecha entre los PLDs y los ASICs

Para abordar esta brecha, Xilinx desarrolló una nueva clase de circuito integrado llamada matriz de puertas programables en campo, o FPGA, por sus siglas en inglés, que estuvo disponible en el mercado en 1984.

Las primeras FPGAs se basaron en la tecnología CMOS (Complementary Metal Oxide Semiconductor – Semiconductor Complementario de Óxido de Metal) y usaron celdas SRAM (Static Random Access Memory – Memoria de Acceso Aleatorio Estático) para fines de configuración. Aunque estos primeros dispositivos tenían comparativamente muy pocos transistores y eran relativamente simples (o el equivalente de ese tiempo) según los estándares actuales, muchos aspectos de su arquitectura subyacente todavía se emplean hasta el día de hoy. En la figura 6 se puede ver una abstracción de cómo se combinaban las memorias SRAM (en la izquierda) y los transistores MOS (en la derecha).

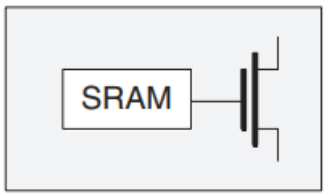


Figura 6. Célula Programable basada en Memoria SRAM

La arquitectura de una FPGA inicial incluía una matriz de elementos de configuración, tales como LUTs (Look-Up-Tables – Tablas de Búsqueda), flip-flops (circuitos biestables —aquellos que únicamente pueden tener dos estados estables, el '1' o el '0' —), multiplexores y bloques de memoria, dispuestos en una estructura bidimensional. Estos elementos son interconectados mediante una red de líneas de interconexión programables, que permiten establecer conexiones entre ellos según los requerimientos del diseño, junto con algunos otros elementos que son de poco interés para este proyecto.

Cada FPGA contenía un gran número de estos bloques de lógica programables. Mediante la programación adecuada de las celdas SRAM, cada bloque lógico en el dispositivo podría ser configurado para realizar una función diferente. Cada registro podría ser configurado para inicializarse conteniendo un lógico 0 o un lógico 1 y para actuar como un flip-flop.

La figura 7 muestra, a grandes rasgos, la estructura interna de una FPGA. El poder lógico de una FPGA reside en sus bloques lógicos configurables (CLBs – Configurable Logic Blocks). La figura 8 muestra los elementos clave que conforman un simple bloque lógico programable. El bloque lógico contiene una tabla de búsqueda (LUT – Look-Up Table) o memoria de acceso rápido que almacena una tabla de valores de salida previamente calculados en función de las entradas proporcionadas que implementa cualquier circuito lógico combinacional que, esencialmente, no deja de ser una combinación o red de interconexión de elementos como transistores y compuertas de la lógica booleana básicas (AND, OR, NOT – Y, O, NO) y estos, a su vez, se reducen también a transistores. La salida de la Look-Up Table puede ser seleccionado como la salida del bloque lógico o como entrada del flip-flop tipo D. Si se

selecciona para ser entrada del flip-flop, la salida de este será, a su vez, la salida del bloque lógico.

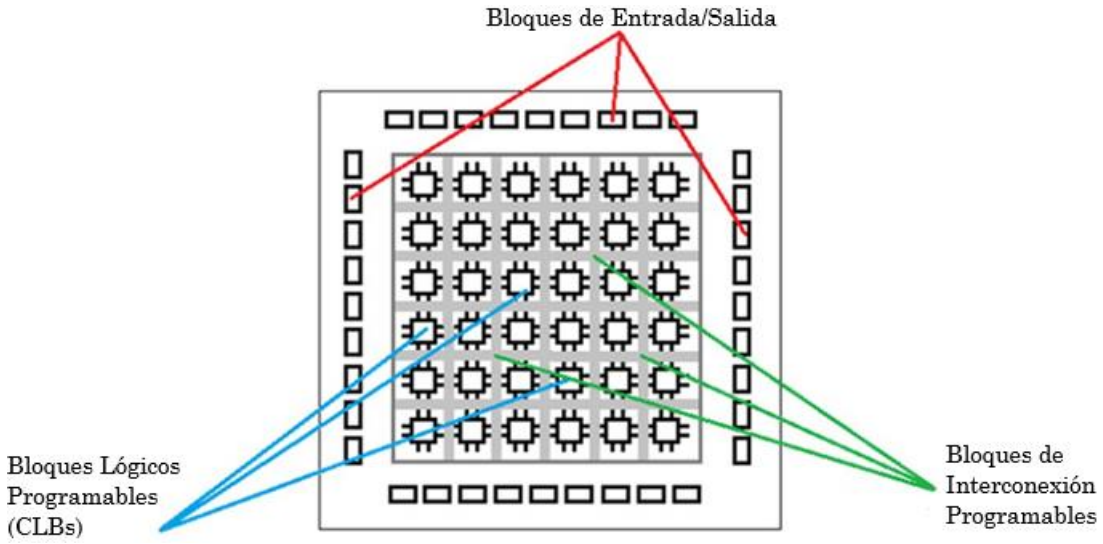


Figura 7. Estructura Conceptual de una FPGA

No se entra en explicaciones de configuración de la LUT, del flip-flop o del multiplexor, a conciencia, debido a que se sale del ámbito de estudio de este proyecto.

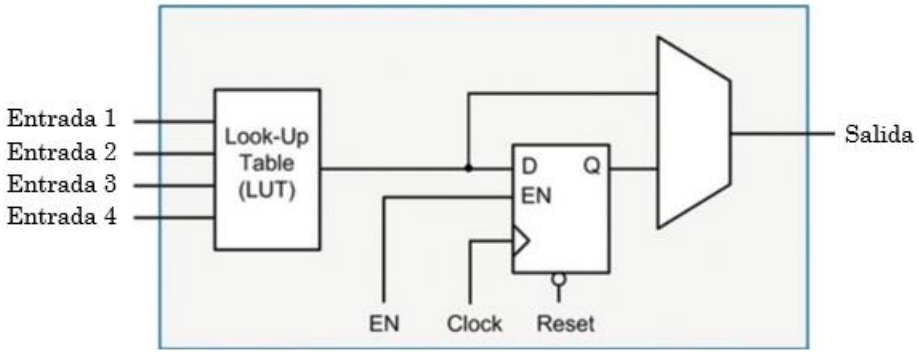


Figura 8. Bloque Lógico Programable (CLB o Elemento Lógico) de una FPGA

La base física de una FPGA está constituida por un sustrato semiconductor, comúnmente elaborado a partir de silicio monocristalino. La práctica totalidad de los dispositivos electrónicos actuales se halla concebida en base a materiales semiconductores, donde se encuentran integrados millones de transistores. Estos semiconductores, pertenecientes al grupo IV del sistema periódico, como el Silicio o el Germanio, o compuestos III-V, como el Arseniuro de Galio, se distinguen por su capacidad de conductividad, la cual se encuentra diferenciada tanto de la de los metales como de la de los aislantes. Tal conductividad, por su parte, está íntimamente vinculada a dos parámetros fundamentales: la temperatura y el nivel de impurificación.

A temperaturas cercanas al cero absoluto, un semiconductor en su estado puro, denominado intrínseco, exhibe un comportamiento aislante, incapaz de conducir corriente. No obstante, conforme se incrementa la temperatura por encima de cierto umbral crítico, dicho

semiconductor inicia su capacidad de conducción. La segunda propiedad, la del grado de impurificación, resulta igualmente destacable. La conductividad de un semiconductor intrínseco puede ser rigurosamente controlada mediante la adición de minúsculas cantidades de átomos de otros materiales, pertenecientes a los grupos III y V, transformándolo en un semiconductor extrínseco, de tipo P o de tipo N [12].

Las siguientes dos figuras (figura 9, figura 10) serán muy esclarecedoras para entender cómo está construido un circuito digital tanto a nivel de compuertas de la lógica booleana como a nivel de componentes electrónicos. La Figura 9 muestra mediante símbolos lógicos cómo sería un circuito digital, en este caso, un circuito XOR (eXclusive OR – 'O' Exclusivo) y a continuación, a nivel físico mediante la simbología electrónica, que se reducen a unos cuantos transistores que pueden estar construidos con tecnología MOSFET (*Metal Oxide Semiconductor Field Effect Transistor* – Transistor de Efecto de Campo de Óxido Metálico Semiconductor) o CMOS.

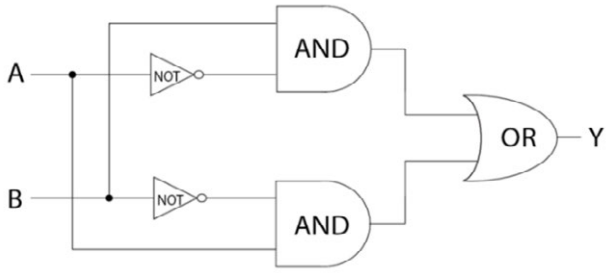


Figura 9. Puerta XOR (mediante símbolos lógicos)

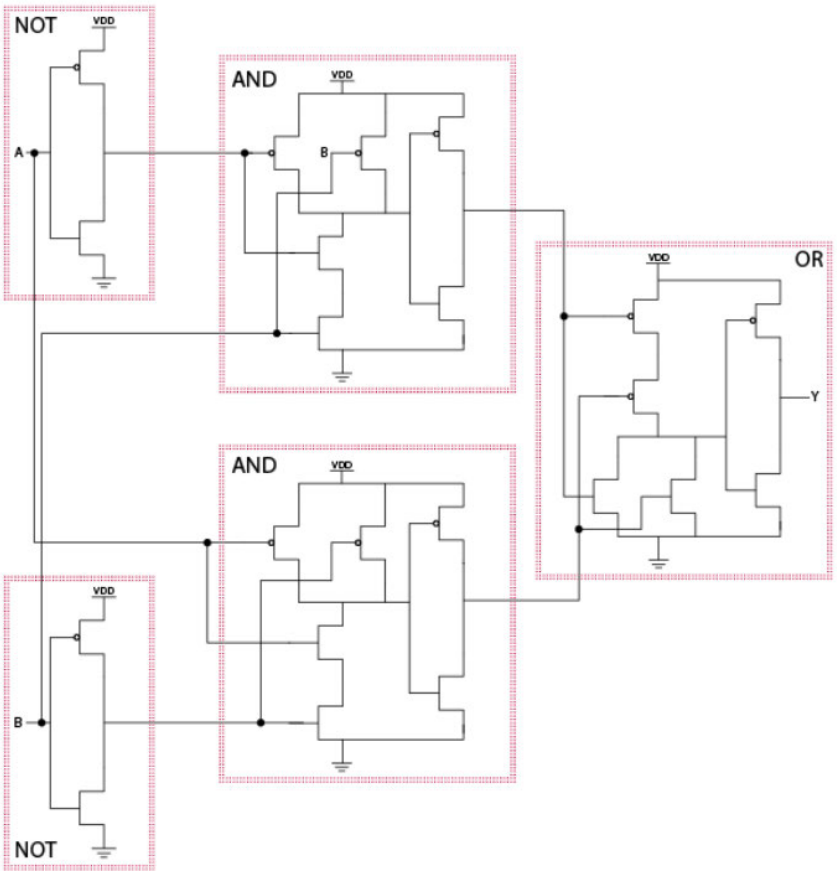


Figura 10. Puerta XOR (mediante símbolos electrónicos)

La figura 10, como se puede observar en ella, aboca al terreno de los transistores, que son el eje central en el estudio de los SEEs, puesto que, cuando ocurre un SEU en una FPGA y se ven comprometidos los LUTs, los flip-flops y los registros de configuración, es porque los transistores que componen estos elementos pueden verse afectados por la radiación ionizante, lo que puede provocar un cambio no deseado en el estado de estos componentes lógicos. Los transistores se caracterizan por su capacidad de controlar el flujo de corriente mediante sus dos posibles estados lógicos. Estos estados, esencialmente dos, se definen como el estado activo (ON o 1) y el estado inactivo (OFF o 0). En el primero, el transistor habilita el flujo de corriente, posibilitando así el tránsito de la señal a través de este. En el segundo, el transistor se encuentra interrumpido, inhibiendo la circulación de corriente. Estos estados, conocidos respectivamente como encendido y apagado, son de naturaleza esencial en la operatividad de los transistores como interruptores electrónicos, contribuyendo así a la realización de la lógica configurable en los dispositivos de FPGA.

La incorporación de un diseño de circuito dentro de una FPGA involucra un proceso metódico y sofisticado que se despliega desde la conceptualización hasta la implementación física del circuito en el hardware. Este proceso se articula mediante el uso de lenguajes de descripción de hardware (HDLs), como VHDL o Verilog, que permiten al diseñador especificar la estructura y el comportamiento del circuito de manera abstracta y precisa. A continuación, se desceibirán los pasos fundamentales que intervienen en la compilación e interpretación de estos lenguajes por la FPGA, culminando en la materialización del diseño del circuito:

- Escritura del Código HDL (Hardware Description Language – Lenguaje de Descripción de Hardware): El primer paso consiste en la redacción del diseño utilizando un lenguaje de descripción de hardware. Este código define la funcionalidad del circuito, describiendo los módulos, las interconexiones, y la lógica que se debe implementar. El diseño puede incluir definiciones para estructuras de datos, señales, y los algoritmos de control que dictan la operación del circuito.
- Simulación: Antes de implementar físicamente el diseño en la FPGA, se realiza una simulación del código HDL. Este proceso permite al diseñador validar la corrección del comportamiento del circuito, verificando que cumple con todas las especificaciones y requisitos funcionales sin errores lógicos.
- Síntesis: El código HDL es entonces procesado por un software de síntesis, el cual traduce las especificaciones de alto nivel del HDL en una red de puertas lógicas y conexiones, conocidas como netlist. La síntesis determina la manera en que las estructuras descritas en el HDL se mapearán a los bloques lógicos disponibles en la FPGA, como las LUTs y los flip-flops.
- Colocación y Enrutamiento: Una vez sintetizado el diseño, el paso siguiente es la colocación, donde se asignan los componentes lógicos del netlist a bloques físicos específicos en la FPGA. Posteriormente, se realiza el enrutamiento, que establece las conexiones físicas entre estos bloques utilizando las interconexiones disponibles en la FPGA.

- Generación de Bitstream: El resultado del proceso de colocación y enrutamiento es un archivo de configuración, comúnmente conocido como bitstream. Este archivo contiene toda la información necesaria para configurar la FPGA de manera que se emule el comportamiento del circuito diseñado. Se darán más detalles respecto a este paso en el apartado de la Descripción de la Solución Propuesta.
- Programación de la FPGA: Finalmente, el Bitstream es cargado en la FPGA a través de un interfaz de programación (Vivado Design Suite, en el caso del proyecto). Una vez cargado, este archivo configura la red de puertas lógicas y las conexiones internas de la FPGA, transformando el dispositivo en una implementación del hardware físico del diseño especificado inicialmente en el HDL.

Para el caso del proyecto se optó por emplear el lenguaje de descripción de hardware denominado VHDL (Very High Speed Integrated Circuits – Hardware Description Language – Lenguaje de Descripción de Hardware de Circuitos Integrados de Muy Alta Velocidad).

4.3.1 FPGA ARTIX de la serie 7

Las FPGAs, como cualquier otro dispositivo electrónico, no tienen un único fabricante. La que conforma el SoC empleado en este proyecto es la Artix-7, del fabricante Xilinx, que forma parte de la séptima generación de dispositivos de la compañía (de ahí, el 7).

Las FPGAs de la serie 7 se destacan por su alta densidad de lógica configurable, su velocidad de funcionamiento, su eficiencia energética y su capacidad para integrar diversos bloques de procesamiento especializado, como DSP y bloques de memoria RAM. Estas FPGAs se utilizan en una amplia gama de aplicaciones, como telecomunicaciones, aeroespacial, automotriz, médica, industrial y de consumo, donde se requiere flexibilidad, rendimiento y capacidad de procesamiento. La figura 11 es una representación abstracta de la arquitectura de la FPGA Artix-7.

La FPGA Artix-7 contiene CLBs; BRAMs (Block RAM – Bloque RAM); bloques I/O (Input/Output – Entrada/Salida); bloques CMT (Clock Management Tile – Módulo de Gestión de Reloj); bloques de lógica FIFO (First Input First Output – Primero en Entrar, Primero en Salir), un algoritmo de almacenamiento de datos; bloques BUFG (Global Clock Buffer – Buffer de Reloj Global), que forman la parte central del chip, siendo la espina de la red de reloj global; bloques DSP para procesado de señales digitales; BUFIO (Clock Buffer Input/Output – Buffer de Reloj de Entrada/Salida) que sirve para direccionar la red de reloj de entrada/salida en el banco en que se encuentra; BUFR (Clock Buffer Register – Registro de Buffers de Reloj) que distribuye las señales de reloj a diferentes regiones del chip; bloques MGT (Multi-Gigabit Transceiver – Transceptor Multi-Gigabit) diseñados para transmitir y recibir datos a velocidades muy altas.

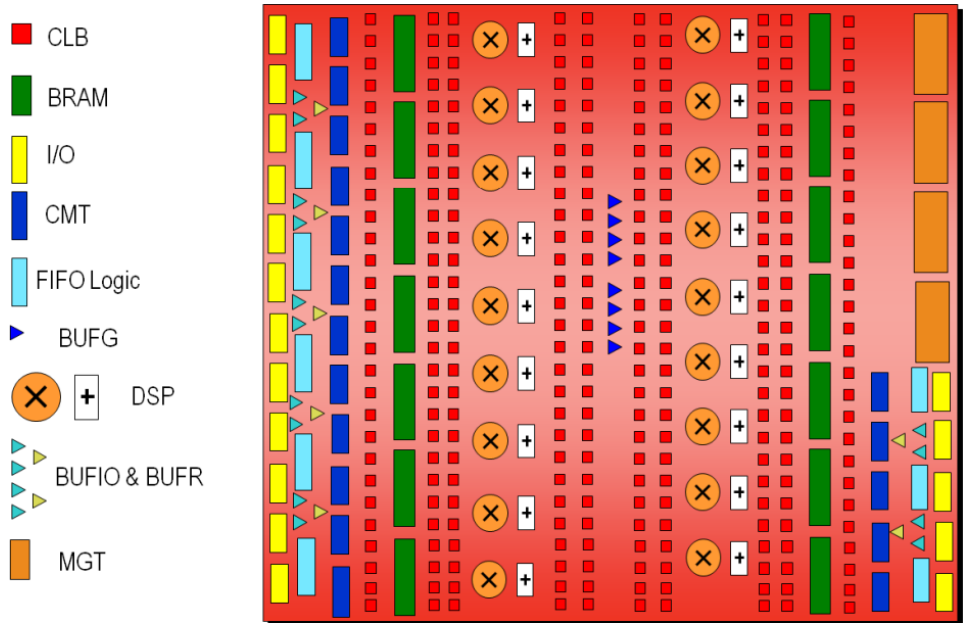


Figura 11. Arquitectura de la FPGA Artix-7

Todas las FPGAs de Xilinx contienen los mismos recursos básicos. Estos recursos se clasifican como:

- Recursos Lógicos:
    a. Slices (agrupados en bloques lógicos configurables (CLBs)): Estos slices son unidades básicas de lógica y almacenamiento que se utilizan para implementar la funcionalidad lógica configurable. Contienen lógica combinatoria y recursos de registro.
    b. Memoria: Bloques de memoria integrados en la FPGA que se usan para almacenar datos de forma temporal o permanente. Pueden ser configurados para funcionar como bloques RAM (BRAM) para almacenar datos durante la ejecución de un programa, o como bloques ROM (Read Only Memory) para almacenar datos permanentes, como tablas de búsqueda (LUTs) o valores constantes.
    c. Multiplicadores: Unidades especializadas diseñadas para realizar operaciones de multiplicación de manera eficiente. Pueden ser configurados para multiplicar números enteros o de punto flotante (decimal) con precisión variable. Son útiles para operaciones matemáticas intensivas, como procesamiento de señales digital (DSP), procesamiento de imágenes o criptografía.
- Recursos de Inteconexión:
    a. Interconexión Programable: Capacidad de una FPGA para configurar las rutas de conexión internas entre sus distintos componentes, como los bloques lógicos, los bloques de memoria y los bloques de entrada/salida (IOBs – Input/Output Blocks).
    b. IOBs: Bloques de entrada/salida: Puntos de conexión entre la FPGA y los dispositivos externos, como sensores, actuadores o circuitos integrados. Estos

bloques proporcionan la interfaz física y eléctrica necesaria para transferir datos dentro y fuera de la FPGA.

- Otros recursos:
    a. Buffers de reloj global (BUFG): Sirven para distribuir señales de reloj a través de la FPGA, asegurando una sincronización adecuada en todo el dispositivo. Estos buffers ayudan a minimizar la fluctuación del tiempo de llegada de la señal del reloj, lo que es crucial para el funcionamiento correcto de los circuitos sincrónicos.
    b. Lógica de escaneo de frontera: Se utiliza para facilitar pruebas de diagnóstico y depuración de la FPGA. Permite que los diseñadores accedan y prueben los puntos de conexión en los bordes de la FPGA, lo que simplifica la detección de fallos y la verificación del funcionamiento correcto del diseño.

En el caso de la FPGA Artix-7, los recursos primordiales para el diseño de un circuito digital son funciones combinatorias y flip-flops. Estos elementos se encuentran dentro de los bloques lógicos configurables (CLBs). Cada CLB contiene dos slices o fragmentos. Estos fragmentos están conectados a una matriz de interruptores (Switch Matrix) que facilita el enrutamiento hacia otros recursos de la FPGA. Además, una característica importante es que la cadena de acarreo (Carry Chain) se extiende verticalmente en una columna desde un fragmento hasta el que está arriba. Cuando se realiza una operación de suma o resta de números binarios, la cadena de acarreo permite que los bits de acarreo se propaguen de manera eficiente a través de múltiples elementos lógicos, sin tener que pasar a través de las rutas de interconexión. La figura 12 expone la representación conceptual de la descripción anterior, con CIN (Carry Input – Acarreo de Entrada), COUT (Carry Output – Acarreo de Salida) el acarreo de entra) y Fabric Routing o enrutamiento de tejido que es la circuitería y los algoritmos que dirigen el tráfico de datos.

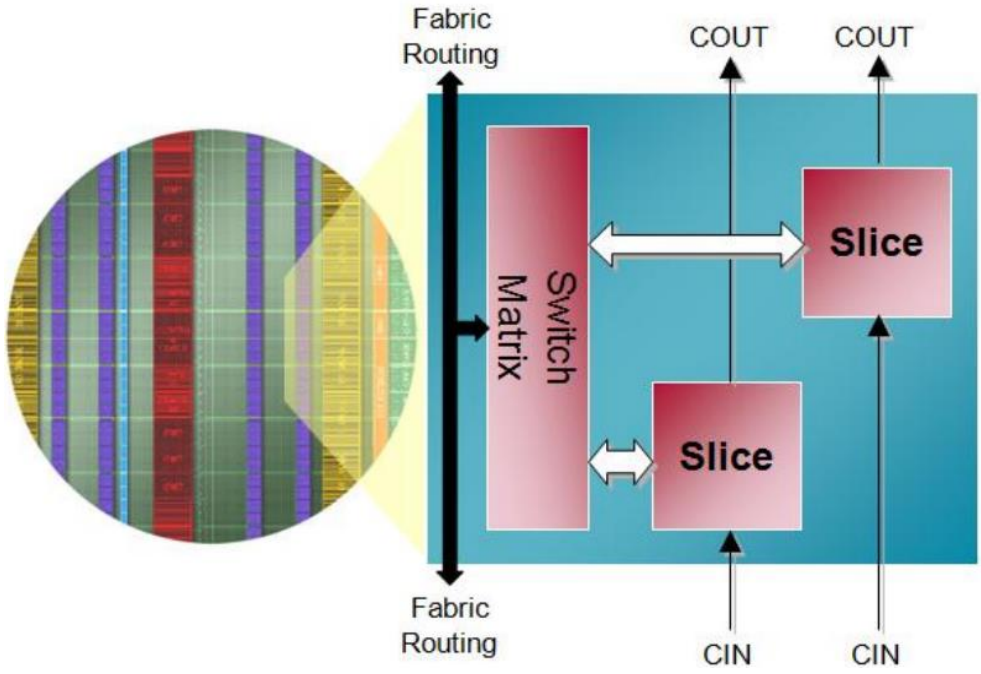


Figura 12. Bloque Lógico Programable (CLB) de la FPGA Artix-7

Una vez visto, a grandes rasgos, cómo es la estructura interna de las FPGAs y, concretamente, de la FPGA Artix-7, se puede realizar un estudio exponiendo los efectos que los SEUs tendrían en los FPGAs, más concretamente, en el semiconductor de que están construidos los transistores en un apartado dedicado a ello.

La FPGA Artix-7 de Xilinx está construida con tecnología CMOS avanzada, específicamente, un proceso de 28 nm de alto rendimiento y bajo consumo (HPL – High Performance Low-power) con puertas de metal de alta constante dieléctrica (HKMG – High-K Metal Gate). Esto permite a las FPGAs de la serie 7 ofrecer un aumento sin precedentes en el rendimiento del sistema, alcanzando hasta 2.9 Tb/s de ancho de banda de E/S, capacidad de 2 millones de bloques lógicos y 5.3 TMAC/s (Tera Multiply-Accumulate Operations per Second – Operaciones a nivel Tera (10^12 bits) de Multiplicar y Acumular por Segundo) DSP, todo mientras consume un 50% menos de energía en comparación con las generaciones anteriores de dispositivos [14].

Para profundizar en la comprensión de los efectos potenciales de los SEUs en las FPGAs, resulta imperativo explorar la tecnología CMOS en estas unidades de procesamiento reconfigurables.

4.4 Tecnología CMOS

La concepción de los transistores de efecto de campo de óxido metálico-semiconductor (MOSFET) es una idea que data de los años 30 por J. E. Lilienfeld, anterior incluso al transistor bipolar. No obstante, la tecnología MOS se volvió viable bastante después, específicamente en los 60, cuando se empezaron a producir las primeras generaciones de MOSFETs que eran solo de tipo N. No fue hasta mediados de la misma década que aparecieron MOSFETs de tipo P, lo cual marcó el inicio de una revolución en la industria de semiconductores.

Los MOSFETs se clasifican en dos tipos fundamentales, siendo, de tipo N y de tipo P, según el tipo de carga que portan en su canal de conducción. Los de tipo N utilizan electrones, que son portadores de carga negativa, como portadores de corriente principal. En contraste, los MOSFETs de tipo P hacen uso de huecos, que son las ausencias de electrones, y se comportan como portadores de carga positiva. La integración de ambos tipos (de ahí, “Complementario”) en la tecnología CMOS permite la construcción de circuitos eficientes que consumen energía solo durante el cambio de estado, es decir, en la transición de encendido a apagado y viceversa, lo que reduce significativamente el consumo de energía en comparación con otros tipos de tecnologías de semiconductores. La Figura 13 muestra los modelos de los transistores MOSFET de tipo N (NMOS – *N-type Metal Oxide Semiconductor*) y de tipo P (PMOS), mientras que en la figura 14 se puede ver la composición física real de estos dispositivos semiconductores.

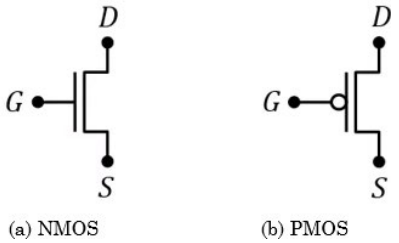


Figura 13. Modelos o símbolos de transistores MOS

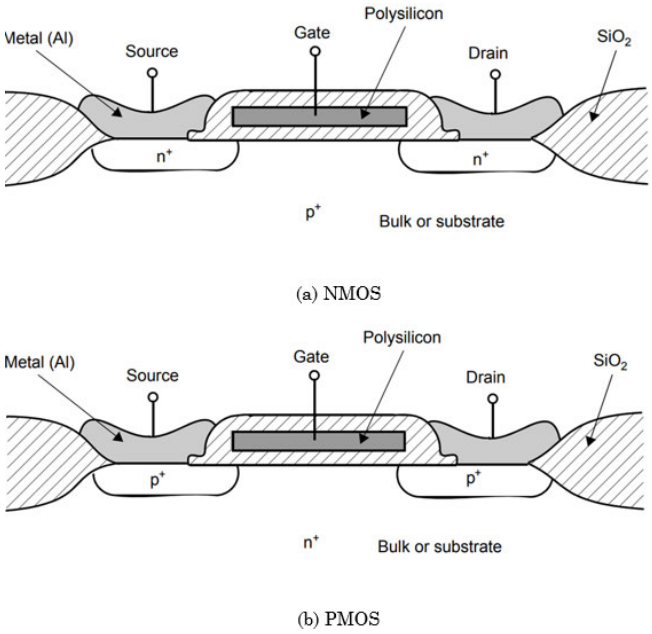


Figura 14. Estructura física de los transistores MOS [16]

Ambos transistores constan de tres terminales que son: G (Gate - Compuerta); D (Drain - Drenador); y S (Source - Fuente). El funcionamiento básico es que los NMOS permiten el flujo de cargas negativas cuando se aplica una tensión positiva en su compuerta (G), mientras que los PMOS (*P-type Metal Oxide Semiconductor*) permiten el flujo de cargas positivas cuando en su compuerta se aplica una tensión negativa. El drenador y la fuente, sombreados en gris, están fabricados con metal de aluminio. Las capas de aluminio se suelen depositar para formar las interconexiones que conectan las distintas regiones del transistor y proporcionan los contactos a través de los cuales el dispositivo se integra en un circuito mayor. Debajo de la compuerta (gate) de polisilicio (Polysilicon) hay una capa delgada de SiO_2 (Dióxido de Silicio) que actúa como un aislante entre la compuerta y el canal del semiconductor, que se formaría bajo la compuerta.

En la figura 15 se muestra la sección transversal de un CMOS en el que se integran los transistores NMOS y PMOS. Se puede observar que el CMOS está construido sobre un sustrato de tipo P, es decir, un semiconductor que ha sido dopado con impurezas para crear una región con una carga eléctrica positiva (más huecos que electrones) predominante, lo que daría lugar al NMOS visto en la figura 14(a). Al mismo tiempo, el PMOS está construido sobre un sustrato de tipo N, es decir, un semiconductor dopado en el que hay más electrones que huecos, visto

en la figura 14(b), sigue con la misma configuración, pero pasa a llamarse cavidad de tipo N en lugar de sustrato de tipo N.

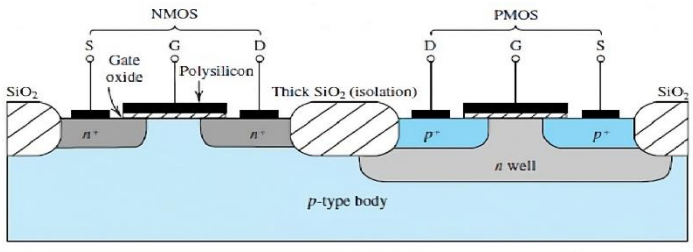


Figura 15. Sección transversal de un CMOS [17]

Existe una capa gruesa de SiO_2 (Thick Silicon Dioxide – Óxido de silicio grueso) que aísla eléctricamente ambos tipos de transistores previniendo el flujo de corriente entre ellos, crucial para la integridad del diseño del chip completo.

Tomando como ejemplo el transistor de tipo NMOS (el principio de funcionamiento del tipo PMOS es igual, salvo en los portadores de carga), los dos estados que tendría son de conducción (ON o 1) o de no conducción (OFF o 0). La figura 16 muestra estos dos estados dando detalles sobre su funcionamiento básico.

La compuerta (Gate) actúa como una de las placas de un capacitor, y la superficie del silicio, justo debajo del delgado aislante SiO_2, actúa como la otra placa. Si se aplica una tensión muy negativa en la puerta, como se muestra en la figura 16(a), se produce una atracción de carga positiva hacia la región que se denomina como de acumulación (Accumulation region). Dado que el sustrato fue originalmente dopado, esta polarización negativa en la compuerta tiene el efecto de simplemente aumentar la dopación del canal a p+, lo que resulta en lo que se llama un canal acumulado. Las regiones de fuente y drenaje n+ están separadas de la región del canal p+ por regiones de agotamiento (Depletion region), lo que resulta en el circuito equivalente de dos diodos en sentido opuesto. Por lo tanto, solo fluirá corriente de fuga incluso si una de las tensiones de la fuente o el drenaje se vuelve grande (a menos que la tensión de drenaje se vuelva tan grande como para provocar que el transistor se rompa). En el caso de aplicar un voltaje positivo en la compuerta, ocurre la situación opuesta, como se muestra en la figura 16(b).

Para pequeños voltajes positivos en la compuerta, los portadores de carga positiva en el canal bajo la compuerta son inicialmente repelidos y el canal cambia de un nivel de dopaje a una región de agotamiento. El voltaje compuerta-fuente (tensión umbral), para el cual la concentración de electrones bajo la compuerta es igual a la concentración de huecos en el sustrato p lejos de la compuerta, comúnmente se denomina voltaje umbral del transistor. Para voltajes compuerta-fuente mayores que esta tensión umbral, a medida que se aplica un voltaje de compuerta más positivo, esta atrae carga negativa de las regiones de la fuente y el drenaje, y el canal se convierte en una región ‘n’ con electrones móviles que conectan las regiones del drenaje y la fuente, por tanto, tiene lugar un flujo de corriente desde la fuente hacia el drenador.

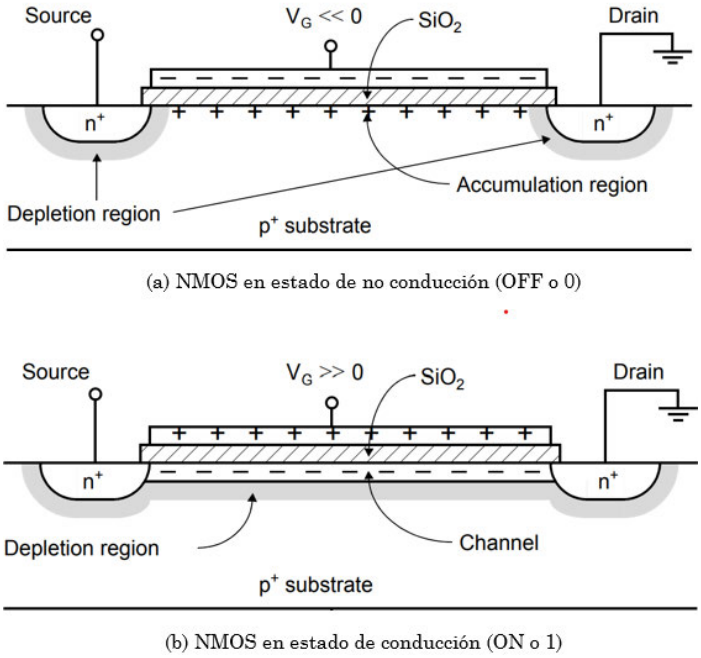


Figura 16. Estados de un transistor tipo N (NMOS)

El entendimiento previo del funcionamiento de los transistores MOS en los circuitos CMOS resulta fundamental al considerar la susceptibilidad a los SEUs. Estos eventos de radiación pueden impactar diversas regiones del transistor, incluyendo el canal y las regiones de fuente y drenaje. Este entendimiento proporciona el contexto necesario para considerar abordar la mitigación de los efectos de los SEUs. Esta consideración es particularmente crítica en aplicaciones donde la seguridad y la fiabilidad son prioritarias, aunque, se sale del ámbito de estudio de este proyecto.

Sin embargo, en el apartado de los SEEs (3.4) se verá cómo los estados del transistor (NMOS o PMOS) se alteran dando lugar a cambios en la operación del circuito.

4.5 SEEs

La historia de los SEEs se origina en una discusión teórica sobre su impacto en el avance del escalado o miniaturización de los Circuitos Integrados (IC - Integrated Circuit). Estos fenómenos surgen como resultado de la constante tendencia en el diseño de dispositivos electrónicos hacia una mayor densidad y tamaños de características más reducidos, un conocimiento asociado con la ley de Moore [7] y las reglas de Dennard [8]. Esto ocurre cuando una sola partícula ionizante penetra en nodos sensibles dentro de los ICs. Esta evolución permite un procesamiento más rápido de la información con una menor cantidad de carga eléctrica requerida. Conforme la carga involucrada disminuye, entra en una escala en la que la radiación cósmica o las partículas alfa pueden generar cantidades de carga correspondientes en el semiconductor. Esta carga puede parecer una señal legítima, alterando temporalmente el contenido de la memoria o los comandos en un flujo de instrucciones. El

problema se agrava debido a la naturaleza aleatoria y el omnipresente espectro de estas partículas de alta energía en el espacio.

En la incursión de una partícula nuclear energética en cualquier material semiconductor, pierde energía a través de la dispersión de Rutherford (interacciones coulómbicas) con la estructura reticular del semiconductor. Mediante interacciones predominantemente de Compton con los núcleos de la estructura cristalina, el frenado de la partícula al transferir energía a la red deja una estela de ionización de pares electrón-hueco libres; portadores de carga móviles que eran eléctricamente inexistentes antes del evento de radiación. Dentro de una estructura de circuito integrado, estos portadores en exceso pueden depositar carga en lugares inesperados y no deseados, a menudo provocando transitorios de voltaje en los nodos del circuito y transitorios de corriente a través de las uniones de los dispositivos [18].

Una interacción de evento singular o único es un efecto localizado y puede dar lugar a un transitorio aparentemente espontáneo dentro de una región del circuito. Si este transitorio afecta a un nodo que está almacenando información, puede provocar una alteración; es decir, la corrupción de la información a un estado irreconocible, ilegible o inestable. Esta alteración puede, a su vez, provocar un error en el circuito si este estado corrupto altera información legítima almacenada o propagada a través del circuito, es decir, una alteración se convierte en un error cuando está enclavada o es interpretada erróneamente como datos válidos por otros circuitos.

Los ICs digitales basados en la tecnología CMOS sufren de cuatro tipos diferentes de SEEs. Todos ellos se originan a partir de la carga inyectada por la incidencia de radiación de partículas, tales como [19]:

- Rayos α (Alpha - Alfa): Núcleos de helio procedentes de materiales radiactivos. Las partículas alfa son emitidas cuando el núcleo de un isótopo inestable decae a un estado de energía más bajo. Contienen energía cinética en el rango de 4 a 9 MeV^1. Hay muchos isótopos radiactivos, sin embargo, el uranio y el torio tienen la mayor actividad entre los materiales que se encuentran naturalmente. En el entorno terrestre, las principales fuentes de partículas alfa son las impurezas radiactivas, como los isótopos a base de plomo en las protuberancias de soldadura de la tecnología de chip flip, el oro utilizado para alambres de unión y el chapado de la tapa, el aluminio en los paquetes cerámicos, las aleaciones de bastidor de plomo y la metalización de interconexión.
- Rayos cósmicos galácticos (GCRs – Galactic Cosmic Rays): Son principalmente núcleos atómicos cargados que provienen del espacio, incluyendo iones pesados como los núcleos de hierro. Menos del 1% del flujo primario alcanza el nivel del suelo. En la atmósfera, estos rayos cósmicos generan partículas secundarias como muones, neutrones, protones y piones, que son las predominantes en el nivel del suelo. Debido a que los piones y muones tienen una vida útil corta y los protones y electrones son atenuados por la interacción coulombiana con la atmósfera, los neutrones son las fuentes de radiación cósmica más probables que causan efectos de eventos únicos SEU en semiconductores de submicrones profundos a altitud terrestre. El flujo de

neutrones depende de la altitud sobre el nivel del mar, la densidad del flujo de neutrones aumenta con la altitud.

- La tercera fuente significativa de partículas ionizantes en dispositivos electrónicos es la radiación secundaria inducida por la interacción de neutrones cósmicos y boro. Se trata de la radiación inducida por la interacción de neutrones cósmicos de baja energía con el isótopo boro-10 (10B, comúnmente utilizado como dopante de tipo p para la formación de uniones en los ICs).

De entre los cuatro tipos de SEEs que afectan a los transistores CMOS, tres se consideran errores transitorios o errores suaves (soft errors), es decir, que no dañan el dispositivo de forma permanente, siendo estos tipos: SEU, SET (Single Event Transient – Transitorio por Evento Singular o Único); y SEFI (Single Event Functional Interrupt - Event Functional Interrupt – Interrupción Funcional por Evento Singular o Único). El otro tipo de SEE, que deja daños permanentes en el dispositivo o en el circuito, es el SEL (Single Event LatchUp – Bloqueo por Evento Singular o Único). Este proyecto se enfoca en los SEUs que afectan a las SRAMs, debido a su construcción a partir de bloques de ICs digitales basados en CMOS vulnerables a tales incidencias.

4.5.1 SEU

La perturbación de evento único (SEU) inducida por neutrones terrestres en las memorias es una de las principales limitaciones para las aplicaciones espaciales y ha sido estudiada durante mucho tiempo [20-22]. Este efecto generalmente se estudia con neutrones de reactores o aceleradores. Los neutrones son partículas sin carga y no pueden inducir directamente la generación de pares electrón-hueco en el semiconductor [23,24]. Sin embargo, transfieren energía a la red y pueden producir productos de retroceso y espalación, que son altamente ionizantes y, por lo tanto, capaces de generar pares electrón-hueco en el semiconductor. Estos iones secundarios pueden generar grandes cantidades de portadores y causar SEU en dispositivos de memoria. En consecuencia, el mecanismo del efecto SEU a partir de las partículas secundarias inducidas por neutrones siempre ha sido un problema complejo y desafiante.

La radiación ionizante induce efectos no deseados sobre los dispositivos semiconductores como resultado de un incidente aislado o singular a lo que se denomina Efecto de Evento Único (SEE). Si bien, los SEEs son más comunes y mejor estudiados en dispositivos de silicio, no se limitan exclusivamente a este tipo de dispositivos.

No solo los drenajes del MOS en estado de no conducción (OFF o 0) son las regiones sensibles para acumular cargas y causar una alteración en una sola celda de memoria de una SRAM [25], sino también muchas otras regiones de la celda: los transistores de habilitación de lectura (RE) y de habilitación de escritura (WE), y, más sorprendente a primera vista, la unión de la cavidad tipo N y el sustrato tipo P que aísla los transistores MOS complementarios.

Las partículas secundarias inducidas por neutrones son altamente ionizantes y pueden generar pérdida de energía ionizante (IEL – Ionizing Energy Loss) cuando atraviesan la región

sensible. Para entender la magnitud del efecto electrónico de las reacciones de neutrones, es útil comprender la transferencia de energía linear (LET – Linear Energy Transfer) de las partículas secundarias inducidas por neutrones en silicio. El IEL está determinado por el LET y por la longitud de transporte (promedio de la distancia que una partícula recorre antes de que sus interacciones con el material lo ralenticen lo suficiente como para que su energía sea absorbida o disipada en el medio). No puede inducirse un SED (Single Event Disruption – Disrupción por Evento Único) cuando el LET de la partícula es menor que el umbral LET de la SRAM, que es el LET crítico para causar una alteración en una celda de memoria. La figura 17 muestra el LET en silicio para diez tipos de partículas secundarias derivadas de la interacción de neutrones con el material del modelo. El LET de cada partícula secundaria primero aumenta con su energía y luego comienza a disminuir en su LET máximo (excepto titanio y tungsteno). La partícula con un número atómico mayor induce un LET mayor en la región sensible. Por lo tanto, para reducir la sección transversal de SEU inducida por neutrones, se debe evitar el uso de materiales con un número atómico alto para eliminar la contribución de partículas secundarias con un LET alto.

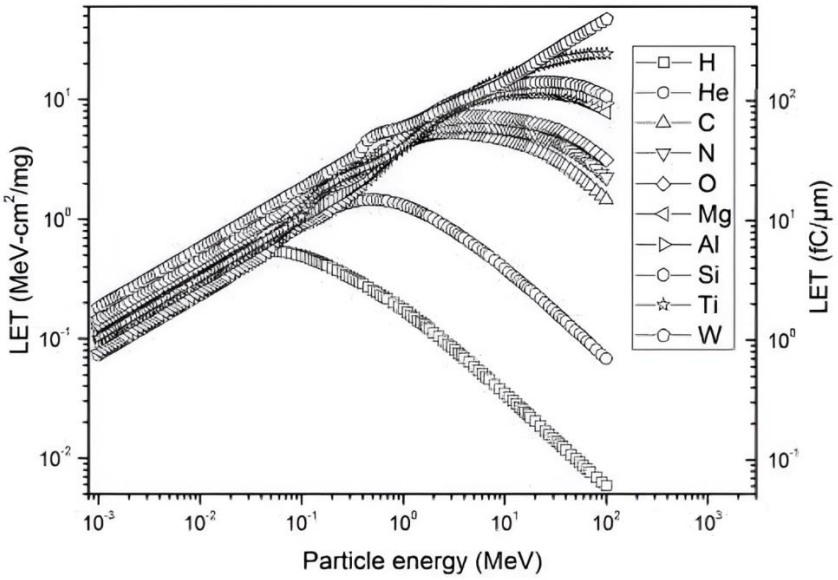


Figura 17. LET en función de la energía para partículas secundarias en silicio [26]

En términos más simples, LET versus energía proporciona información sobre cuánta energía transfieren las partículas secundarias al material de silicio a medida que se desplazan a través de él, lo que ayuda a evaluar su posible impacto en el rendimiento y la fiabilidad de los dispositivos electrónicos fabricados con materiales basados en silicio.

Una partícula puede inducir una Perturbación por Evento Único (SEU) cuando impacta en la región del canal de un transistor NMOS apagado o en la región de drenaje de un transistor PMOS apagado o de no conducción. La ionización puede inducir un pulso de corriente en una unión P-N. Conceptualmente, cuando la carga inyectada por el pulso de corriente en un nodo sensible excede la carga crítica (Q_crit), se genera un error transitorio en la unión afectada. Como consecuencia de este fenómeno, se originará la corrupción de los datos almacenados en la memoria SRAM como un simple bit flip (cambio del estado de un bit) o MBU que pueden

ser transitorios o permanentes. En el caso de una alteración transitoria, se estaría ante un SET (Single Event Transient – Transitorio por Evento Único). Por el contrario, si la alteración es permanente, es imprescindible reiniciar y reconfigurar el dispositivo.

4.5.2 SET (Single Event *Transient* – Transitorio por Evento Único)

Un SET se produce después de que una partícula ionizante energética ha llegado al silicio cerca de los nodos sensibles del dispositivo [26].

A lo largo del camino recorrido (Ion Track), indicado por la flecha, en la figura 18(a), la partícula produce una distribución radial densa de pares electrón-hueco. Si la huella de ionización resultante atraviesa la región de agotamiento (en color blanco), los portadores son recogidos rápidamente por el campo eléctrico, compensando así la carga almacenada en la unión. Fuera de la región de agotamiento, la distribución de carga no equilibrada induce una distorsión de potencial temporal en forma de embudo a lo largo de la trayectoria del evento, aumentando así, aún más, la recolección de carga por deriva, es decir, al movimiento de los portadores de carga en el material semiconductor debido a la presencia de un campo eléctrico generado por una diferencia de potencial (figura 18(b)). Una fase de recolección "inmediata" típicamente sigue durante decenas de picosegundos y, a medida que el embudo colapsa, la difusión domina entonces el proceso de recolección (figura 18(c)) hasta que todos los portadores en exceso han sido recogidos, recombinados o difundidos lejos del área de la unión (aproximadamente en nanosegundos).

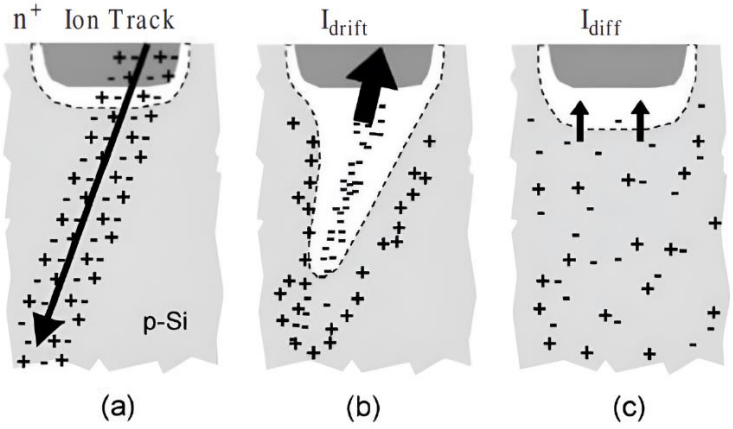


Figura 18. Representación de la acumulación de carga en la unión del silicio [27]

La carga transitoria recogida del evento de radiación produce un pulso de corriente en la unión, como se ilustra en la figura 19. Se puede observar en ella una región llamada Inicio de Evento (Onset of Event), punto donde la corriente comienza a aumentar debido al evento de ionización inicial hasta alcanzar la región de Acumulación de Carga Inmediata (Prompt Charge Colecction) donde la mayoría de los portadores de carga son rápidamente acumulados, seguido de la región de Acumulación de Carga por Difusión (Diffusion Charge Collection) donde la corriente disminuye a medida que los portadores de carga restantes son acumulados más lentamente.

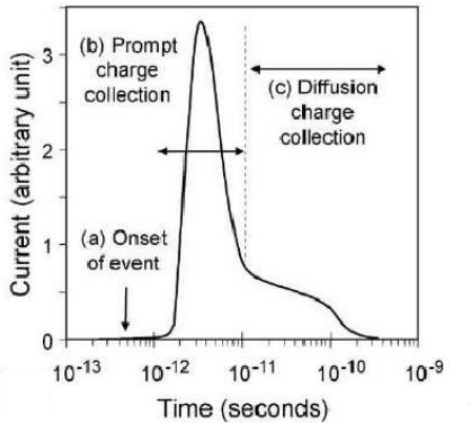


Figura 19. La corriente de unión inducida en función del tiempo

El transitorio de corriente típicamente dura 200 picosegundos, con la mayor parte de la acumulación de carga ocurriendo dentro de 2-3 micrones de la región de la unión para las tecnologías CMOS (de tamaños menores que la micra) modernas. Las constantes de tiempo dependen fuertemente del tipo de ion, su energía inicial y las propiedades de la tecnología específica. Si se recoge suficiente carga en un nodo, el estado de los datos puede cambiar. La carga recogida (Q_coll) es una función de la energía y trayectoria de la partícula ionizante, la estructura y de cómo se ha dopado el sustrato de silicio, y el campo eléctrico local.

En la figura 20 se muestra cómo un pulso de corriente inducida por un SEE se convierte en un pulso transitorio de voltaje. El pulso transitorio de voltaje inducido puede propagarse a través de varios niveles de puertas lógicas. Debido a que una partícula puede inducir un SEU cuando golpea ya sea la región de canal de un transistor NMOS apagado o la región de drenador de un transistor PMOS apagado, se considerará el golpe en el área de drenador de un PMOS apagado como un ejemplo particular, simple e ilustrativo.

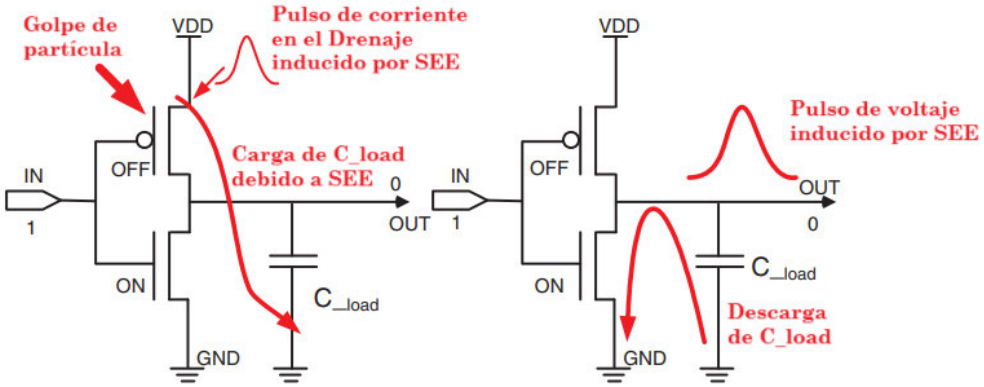


Figura 20. Vista esquemática de pulso de corriente inducido por SEE convertido en pulso de voltaje en un inversor CMOS.

La carga crítica depende de la carga total recogida en el nodo sensible, así como de la forma temporal del pulso de corriente y el voltaje de suministro del dispositivo. Así, se define un

parámetro llamado tiempo de conmutación (t_th), siendo la th (Threshold - Umbral), es decir, Tiempo hasta el Umbral o Tiempo de Realimentación como el intervalo después del impacto de la partícula en el cual el voltaje del nodo afectado supera el voltaje umbral. La carga en el capacitor de salida (C_load) es igual a (Q_crit) en ese momento. (Q_crit) se puede calcular integrando la corriente que fluye en el nodo sensible después del impacto [27]. La condición para que el SEE se propague es que el voltaje del nodo de salida siga la Ecuación 1.

$$\boldsymbol{V \geq \frac{Q_{crit}}{C} = \frac{1}{C} \int_0^{t_{th}} I_{drain}(t)\, dt} \quad (1)$$

La ecuación anterior describe la condición para que un SEU se propague en el circuito, es decir, la propagación del SEU depende de la capacidad del nodo sensible para alcanzar un voltaje lo suficientemente alto (mayor o igual al voltaje resultante de $\frac{Q_{crit}}{C}$) dentro de un periodo de tiempo dado t_{th}. Si tal condición se da, el pulso de corriente inducido por el impacto de una partícula energética ha sido suficiente para cambiar el estado lógico del circuito.

El proceso de descarga se puede modelar como un circuito sencillo RC. El ancho de pulso del pulso de voltaje que se genera depende del valor de la capacitancia y de la constante de tiempo τ=RC que describe cómo de rápido el circuito RC se descarga, siendo R la resistencia del nodo y C la capacitancia de este. Reescribiendo la corriente que se genera en el drenaje, dicha constante se justificaría en la forma (ver ecuación 2).

$$I_{drain}(t) = I_0 e^{-\frac{t}{\tau}} = I_0 e^{-\frac{t}{RC}} \quad (2)$$

La ecuación anterior denota que la corriente decae después del evento ionizante y esta forma exponencialmente decreciente se usa en la integral para calcular la carga total acumulada durante el tiempo de conmutación t_{th}, donde I_0 es la corriente inicial justo después del evento. En este sentido, el voltaje o tensión resultante se puede expresar como se muestra en la Ecuación (3).

$$v(t) = v_0 e^{-\frac{t}{\tau}} = v_0 e^{-\frac{t}{RC}} \quad (3)$$

Por tanto, la disminución del valor de RC potencia el proceso de descarga, lo que podría menguar la probabilidad de que una unidad de error inducido por radiación altere el estado lógico del circuito. Sin embargo, como contrapartida, podría exponer al circuito a la susceptibilidad ante señales de ruido transitorio, propiciando disparos falsos o cambios en el estado lógico. Esta situación promueve la aplicación de técnicas de mitigación de errores, como la redundancia y los diseños tolerantes a fallos, aunque estos aspectos no se abordarán aquí por exceder el alcance del presente proyecto.

4.5.3 SEL (*Single Event Latchup* – Bloqueo por Evento Único)

El fenómeno del Bloqueo por Evento Único (SEL – Single Event LatchUp) ha sido identificado en una variedad de dispositivos basados en tecnología CMOS durante los últimos 40 años [29-

31]. Este fenómeno es inherente a la tecnología CMOS y puede ser inducido por partículas de radiación como protones de alta energía, neutrones o iones pesados. Su efecto es que las cargas depositadas por el ion modifican el campo eléctrico y pueden desencadenar la estructura bipolar parásita (configuración no deseada dentro del transistor CMOS que puede comportarse como un transistor bipolar adicional no planificado), lo que conduce a la destrucción del dispositivo si no se corta la alimentación del dispositivo. Con la escala tecnológica, las capacitancias nodales reducidas y los tamaños de los dispositivos disminuyen la cantidad de cargas inducidas por radiación necesarias para modificar el campo eléctrico y crear un bloqueo [32]. Además, se sabe que el bloqueo es dependiente de la temperatura [30,33].

El efecto SEL representa un reto de fiabilidad para la industria aeroespacial y los fabricantes de semiconductores. Ambos están enfocados en la reducción de la sensibilidad al SEL basada en la disposición y las optimizaciones del proceso. El modelado del Bloqueo por Evento Único inducido por partículas de radiación sigue siendo un campo que se investiga [34].

4.5.4 SEFI (*Single Event Functional Failure* – Interrupción Funcional por Evento Único)

El fenómeno denominado Interrupción Funcional por Evento Único (SEFI – Single Event Functional Interruption) se menciona por primera vez en la edición de 1996 del Estándar EIA/JEDEC Standard [35]. SEFI es una anomalía que ocurre en microcircuitos debido al impacto de un solo ion, similar a las perturbaciones de evento único (SEUs) que afectan a los dispositivos de memoria, en el caso de este proyecto: SRAM. Aunque comparte similitudes con las SEUs, SEFI se presenta de manera distinta al provocar una interrupción temporal en el funcionamiento normal del dispositivo afectado. La duración de SEFI puede variar: en algunos casos persiste mientras la alimentación se mantenga, mientras que en otros tiene una duración finita pero significativa. Aunque SEFI ha sido observado durante mucho tiempo, solo recientemente ha recibido atención en publicaciones [36]. A diferencia del bloqueo por evento único (SEL) o la ruptura repentina, SEFI generalmente no provoca un consumo elevado de corriente.

4.6 Efecto de los SEUs en la FPGA Artix de la serie 7

Cuando se produce un SEU (error transitorio o error suave (*soft error*)), uno o más bits de memoria se corrompen. Los bits de memoria afectados pueden estar en la memoria de configuración del dispositivo (que determina el comportamiento del diseño), o pueden estar en elementos de memoria de diseño (que determinan el estado del diseño). Las siguientes cuatro categorías de memoria representan la mayoría de la memoria en un dispositivo [36]:

- Memoria de Configuración: Elementos de almacenamiento utilizados para configurar la función del diseño cargado en el dispositivo. Los elementos esenciales dentro del contexto del diseño son aquellos bits (esenciales) que mantienen una conexión directa con la circuitería subyacente. Si uno de estos elementos experimenta un cambio, este modificará la configuración del diseño, aunque no necesariamente afectará su

funcionalidad inherente. En ausencia de un conocimiento preciso sobre qué elementos son esenciales, el sistema se ve compelido a considerar que cualquier error suave detectado ha comprometido la integridad del diseño en su totalidad. El comportamiento de mitigación a nivel del sistema, en consecuencia, suele conllevar una interrupción o degradación del servicio hasta que se realice la reparación de la configuración de la FPGA y se reinicie o restablezca el diseño en cuestión.

- Memoria de Bloques: Constituye una reserva de almacenamiento de gran capacidad, destinada a preservar el estado operativo del diseño en cuestión. Tal como su nombre sugiere, los bits se organizan en conglomerados físicos, formando así bloques que se distribuyen por todo el dispositivo. En términos de volumen, esta memoria ocupa el segundo lugar en cuanto a cantidad de bits, revelando su importancia dentro del conjunto de elementos de almacenamiento de la FPGA.
- Memoria Distribuida: Se erige como una reserva de almacenamiento de capacidad intermedia, destinada también a preservar el estado operativo del diseño. Esta categoría de memoria se encuentra dispersa a lo largo y ancho del dispositivo, alojada en ciertos bloques lógicos configurables (CLBs). A pesar de su ubicuidad, su capacidad se sitúa en un punto intermedio en comparación con otras formas de almacenamiento. En términos de volumen, la Memoria Distribuida constituye el tercer bloque más significativo en términos de cantidad de bits dentro de la FPGA.
- Flip-Flops: La denominación flip-flop (circuitos biestables) se acuñó para describir un tipo de elemento de memoria que tenía la capacidad de cambiar entre dos estados distintos, como si "diera un vuelco" de un estado a otro. Son modestos en capacidad, pero cruciales en función, pues se encargan de conservar el estado del diseño. Esta forma de almacenamiento se encuentra omnipresente en cada uno de los bloques lógicos configurables (CLBs), extendiéndose a lo largo y ancho del dispositivo. Aunque su capacidad individual es limitada, en conjunto, los flip-flop conforman el cuarto bloque más relevante en cuanto a la cantidad de bits dentro de la FPGA.

Cuando se ensambla el diseño de un circuito en la FPGA, principalmente se resguarda en la memoria de configuración. Este repositorio es el almacén donde los bits de configuración, cuya explicación detallada será proporcionada posteriormente, son cargados. Estos bits, determinantes para el comportamiento del diseño en la FPGA, se almacenan en esta memoria para formatear la lógica y los recursos del dispositivo conforme a las especificaciones del diseño. La Memoria de Configuración se estructura como un conjunto de marcos, semejante a una memoria RAM estática (SRAM) de amplia extensión, como ya se delineó en una sección dedicada a las FPGAs.

4.7 Elaboración de la metodología de análisis de SEEs mediante el Controlador SEM IP

La elaboración de la metodología de análisis de los SEEs se llevará a cabo utilizando el software Vivado Design Suite y Vitis IDE (Integrated Development Environment – Entorno de Desarrollo Integrado), aprovechando la SEM IP (Soft Error Mitigation Intellectual Property – Propiedad Intelectual de Mitigación de Errores Suaves o Transitorios) proporcionada por el fabricante Xilinx. En el contexto del software Vivado, las IPs son componentes predefinidos que encapsulan funcionalidades específicas y complejas dentro de un diseño electrónico. Estas IPs son desarrolladas por fabricantes de chips como Xilinx y proporcionan una manera eficiente de integrar características complejas en un diseño sin la necesidad de implementarlas desde cero. Al utilizar IPs, se puede acelerar el proceso de desarrollo al aprovechar el trabajo previo y la experiencia de otros en la implementación de funciones específicas. La SEM IP, que más adelante se detallará su funcionamiento interno, es un controlador diseñado específicamente para mitigar, y no para prevenir, los efectos de errores suaves en dispositivos FPGA. En segundo lugar, pero no menos simple ni menos importante, se expone una técnica propia, que consiste en la inyección de un error (alteración de bit) en un punto sensible de un circuito que realiza la FFT de una señal sinusoidal que se le entrega como entrada, para posteriormente observar el estado de la transformada antes y después de la inyección del error. Los detalles se expondrán en los respectivos subapartados.

La elección de realizar una inyección de errores en un punto sensible de un circuito que ejecuta la Transformada Rápida de Fourier (FFT) de una señal sinusoidal es particularmente reveladora. La FFT es una herramienta esencial en el procesamiento de señales y su selección como punto de inyección de errores proporciona un caso de estudio valioso, dado que las señales procesadas por FFT son críticas en una amplia gama de aplicaciones, desde comunicaciones hasta radar y más allá. Observar el estado de la transformada antes y después de la inyección de errores permite no solo entender la robustez del diseño actual, sino también desarrollar métodos para aumentar la tolerancia a errores en operaciones críticas.

Sin embargo, es fundamental reconocer que los análisis a realizar a través de la SEM IP y la inyección de errores en circuitos FFT son solamente componentes de una estrategia más amplia. La decisión de no priorizar inmediatamente una estrategia holística reconoce la necesidad de conciliar los objetivos de investigación con los recursos a corto plazo, como es el tiempo que se debe dedicar al proyecto. Además, se necesitaría una comprensión más profunda de los SEEs, incluidos sus mecanismos de generación y las vías para su mitigación, que exige un enfoque multidisciplinario incorporando conocimientos de física de semiconductores, teoría de la fiabilidad, y, por supuesto, diseño y análisis de sistemas electrónicos.

Los análisis de los Efectos de Eventos Singulares (SEEs), todos realizados de forma virtual mediante el uso exclusivo de software, además del hardware físico que es la Blackboard, tienen por objetivo observar la reacción del System on Chip (SoC) frente a tales eventos y sus consiguientes efectos.

El controlador SEM IP, desarrollado por Xilinx, constituye un mecanismo avanzado para la detección, corrección y clasificación de errores transitorios (soft errors) en la memoria de configuración de una FPGA. Según se expone en su documentación técnica [36], este dispositivo no impide la ocurrencia de tales errores transitorios, sino que provee al diseñador de un método sistemático y eficiente para manejar estos incidentes, con el fin de mejorar la fiabilidad y la disponibilidad del sistema. Este enfoque permite enfrentar los desafíos inherentes a la integridad de los datos en entornos susceptibles a perturbaciones, reforzando la continuidad y el rendimiento operativo de los dispositivos críticos permitiendo la reducción del mantenimiento y los tiempos de inactividad. Debe constar que el controlador no opera sobre los posibles errores transitorios que pudieran generarse en la Memoria de Bloque, Memoria Distribuida o en los flip-flop. Estas regiones deben abordarse con medidas de precaución tales como redundancia o detección de error y códigos de corrección.

Como recordatorio de una mención previa, la SEM IP, como su nombre indica, es una Propiedad Intelectual (IP – Intellectual Property) de Xilinx diseñado para ser implementado específicamente en FPGAs y para tal acción, el fabricante provee a los usuarios de la plataforma de software de desarrollo Vivado Design Suite, el cual integra al controlador SEM que es, esencialmente, un módulo o bloque predefinido o, en términos más técnicos, se puede describir como un bloque de IP encapsulado, lo cual significa que bajo su interfaz y funcionalidades visibles se encuentra una implementación más compleja compuesta principalmente de código VHDL o Verilog.

Como se expondrá a continuación, la SEM IP no sólo desempeña una función en la detección y corrección de errores, sino que también permite al controlador inyectar errores en diversas localizaciones de la Memoria de Configuración, para posteriormente proceder a su detección y corrección. En la figura 21 se ilustran todas las señales de entrada y salida pertinentes al controlador SEM, indicando, en las áreas sombreadas, que dichas señales existen solamente con ciertas configuraciones.

A primera vista, el funcionamiento del controlador puede parecer desafiante debido a la complejidad y cantidad de puertos disponibles. No obstante, no es necesario utilizar todos estos puertos; se emplean únicamente aquellos que son esenciales para la aplicación específica. Este proyecto tiene como objetivo evaluar y analizar las respuestas del dispositivo SoC ante eventos singulares, particularmente cuando el controlador inyecta fallos de manera periódica en ubicaciones aleatorias de la Memoria de Configuración. Esta metodología simula el efecto de la radiación en distintas posiciones, permitiendo observar cómo las alteraciones inducidas pueden ser transitorias o, en ciertos casos, requerir un reinicio del dispositivo para restaurar su funcionamiento normal.

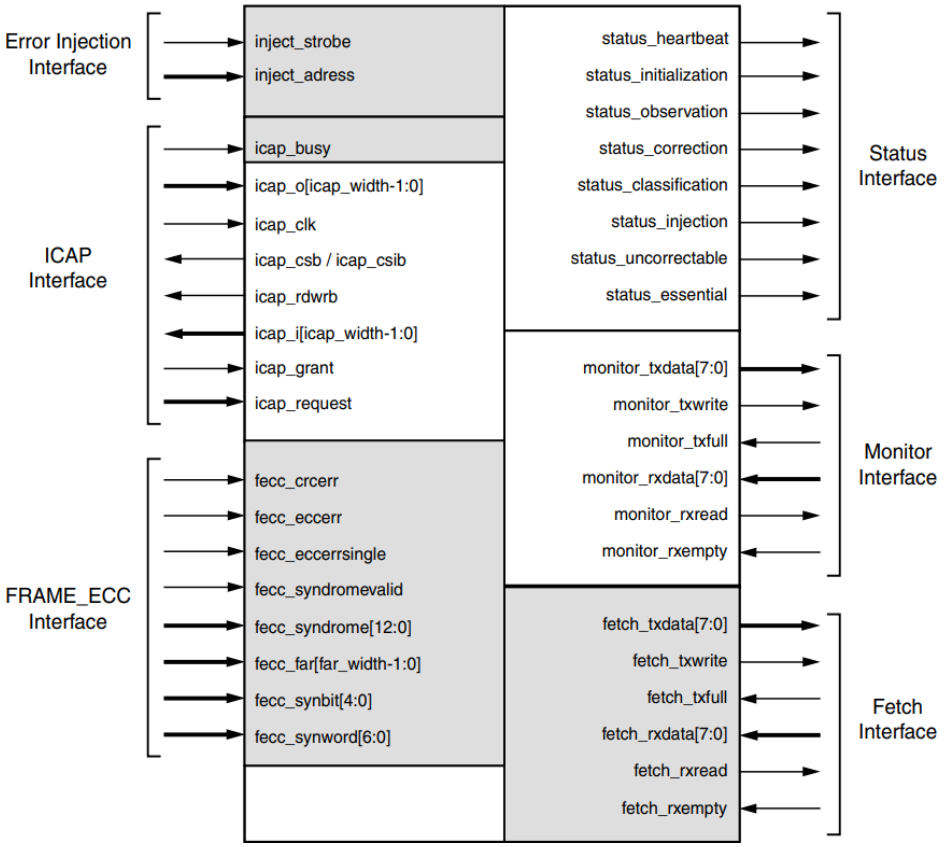


Figura 21. Representación de los puertos del Controlador SEM

El controlador está equipado con seis interfaces, de las cuales se utilizarán solamente tres en particular: la Interfaz de Inyección de Errores (Error Injection Interface), la Interfaz ICAP (*Interconnect Configuration Access Port* – Puerto de Acceso a la Configuración Interna), y la Interfaz de Estado (Status Interface). Las primeras dos constituyen interfaces de entrada al controlador, proporcionando los medios para introducir fallos y configuraciones. Por su parte, la última es una interfaz de salida, la cual está diseñada para monitorear y reportar el estado actual del controlador. Estas interfaces seleccionadas son cruciales para la administración y evaluación de la integridad del sistema en el contexto del proyecto.

Cualquier diseño de circuito digital debería incluir una señal de reset (reajuste o reinicio). La razón fundamental para incorporar una señal de reset en los circuitos digitales es asegurar que el sistema comience desde un estado conocido y definido. Esto es especialmente crítico en sistemas complejos que incluyen múltiples módulos de procesamiento o estados lógicos, como microprocesadores, microcontroladores, o sistemas en chip (SoCs). En el caso del controlador SEM, este no tiene entrada ni salida de reset. Se inicializa automáticamente con un reset síncrono interno.

Se procederá a seguir el ejemplo de diseño a nivel de sistema del Controlador SEM que se describe en su hoja técnica [36]. El ejemplo consta del Controlador y varios adaptadores (shims) que sirven para interconectar el controlador con otros dispositivos. Tal como se entrega, el ejemplo de diseño a nivel de sistema no es un diseño de referencia, sino una parte integral de la solución total y completamente verificada por Xilinx. Aunque los diseñadores

tienen la flexibilidad de modificar el ejemplo de diseño a nivel de sistema, el enfoque recomendado es utilizarlo tal como se entrega. Se muestra en el diagrama de bloques de la figura 22 los módulos de los que consta el controlador.

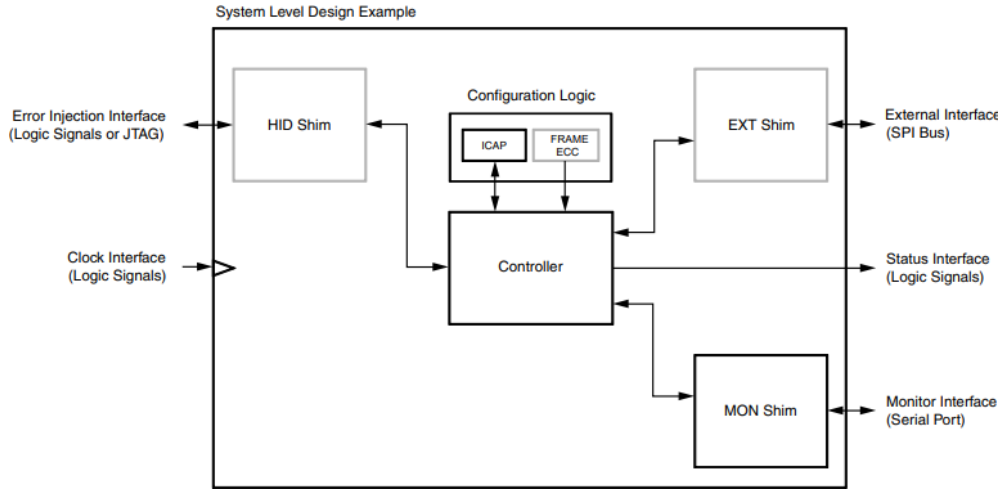


Figura 22. Diagrama de Bloques interno del Controlador SEM IP

Están recuadradas en rojo las únicas interfaces que son necesarias para el caso particular del proyecto. Entre todos los componentes de los que consta el controlador, son relevantes:

- Lógica de Configuración (Configuration Logic): Es el núcleo del sistema de configuración, que podría incluir un bloque de Interfaz de Configuración de Acceso Interno (ICAP) para la reconfiguración dinámica y una lógica FRAME ECC (*Error Correction Code*) para la corrección de errores. Este módulo es esencial para gestionar la configuración y el funcionamiento del FPGA, lo que incluye la capacidad de alterar dinámicamente la configuración para simular y estudiar los efectos de los SEUs.
- Controlador (Controller): Actúa como el cerebro del sistema, coordinando las operaciones y el flujo de datos entre los distintos shims (adaptadores) y módulos de lógica de configuración. Este controlador centraliza la gestión y el control del dispositivo, ejecutando los algoritmos necesarios para operar y monitorear el sistema.
- Interfaz de Inyección de Error (Error Injection Interface): A través de señales lógicas o el estándar JTAG (Joint Test Access Group – Grupo de Acción Conjunta para Pruebas), esta interfaz permite la inyección controlada de errores en el sistema.
- Interfaz de Reloj (Clock Interface): Esta interfaz proporciona las señales de reloj necesarias para sincronizar las operaciones del sistema. Una señal de reloj precisa y estable es crucial para el funcionamiento correcto y coherente de todos los componentes digitales, y especialmente para la sincronización de eventos en un entorno de pruebas controladas.
- Interfaz de Estado (Status Interface): Mediante señales lógicas, esta interfaz ofrece un canal para monitorear el estado del sistema. La información sobre el estado es vital para asegurar que el sistema está funcionando como se espera y para detectar cualquier irregularidad provocada por la inyección de errores.

Los puertos de entrada y salida que se usarán del controlador son representados en la figura 23.

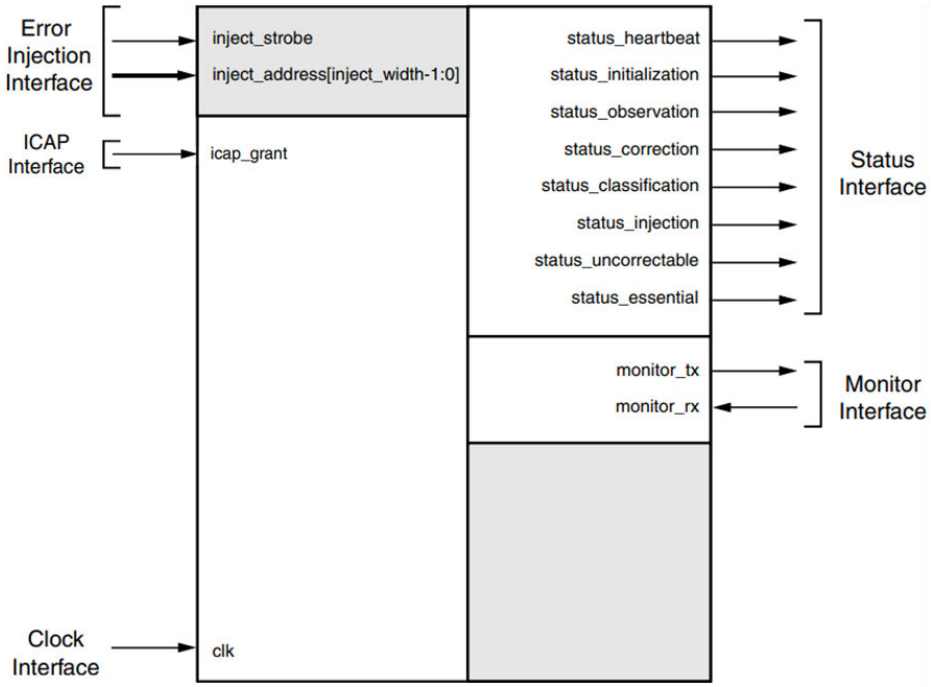


Figura 23. Representación definitiva de los puertos del Controlador SEM

Aunque la Interfaz de Monitor (Monitor Interface), no ejerce influencia sobre las operaciones que realizará el Controlador en el marco del proyecto actual, se mantiene como una interfaz residual, debido a la naturaleza inextricable de la misma dentro del conjunto de la tecnología prediseñada [36]. Su presencia no altera ni compromete la funcionalidad del dispositivo.

En la presente etapa, se ha establecido con claridad el empleo del Controlador SEM IP que se hará efectivo, y la herramienta de software provista para tal fin es la Vivado Design Suite de Xilinx. Subsiguientemente, una vez efectuada su implementación, se procederá a una exposición detallada tanto de las señales de entrada y salida, como de la mecánica interna de su funcionamiento.

4.7.1 Generación del Controlador SEM IP en Vivado

Para la creación de una instancia personalizada del Controlador SEM IP dentro del entorno de Vivado, es imperativo seguir una serie de pasos metodológicos que, aunque de naturaleza aparentemente sencilla, su inobservancia podría resultar en la falla de generación de la IP o su generación en un estado defectuoso. Los procedimientos que se deben adherir, para el caso particular del proyecto, son los siguientes:

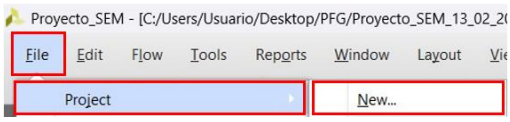
a. Inicialización del entorno de Vivado y creación de un nuevo proyecto que alojará la SEM IP. Este proyecto de Vivado será para la exclusiva generación del Controlador que se empaquetará como cualquier IP de las que dispone Vivado. En esta fase, se debe

aceptar todas las opciones por defecto salvo cuando solicite la placa sobre la que se va a trabajar (penúltima fase de la creación del proyecto), que en este proyecto es la Blackboard.

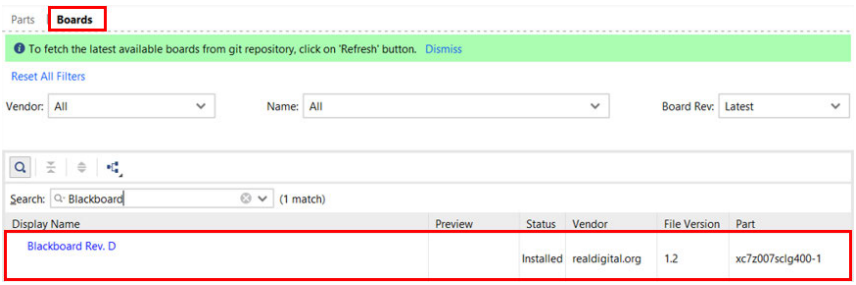
b. Una vez creado el proyecto nuevo, accedemos al IP Catalog (Catálogo de IPs) desde la ventana de Project Manager.

c. Se abrirá una nueva ventana que dará la posibilidad de buscar la IP de Soft Error Mitigation. Una vez localizada la IP, se accederá a sus propiedades (click derecho) para generar dicha IP de forma personalizada etiquetado como Customize IP.

A continuación, se muestran en la figura 24 varias capturas para facilitar la creación de la SEM IP.

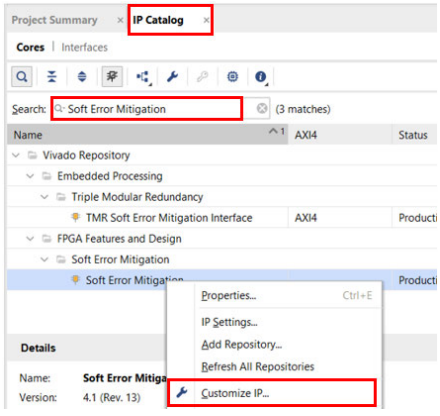


a.1) Creación de un nuevo Proyecto



a.2) Selección de la placa de desarrollo Blackboard

b) Selección de IP Catalog



c) Búsqueda y personalización de la IP

Figura 24. Acceso a la personalización del Controlador SEM IP

Una vez alcanzado este punto, se abrirá una nueva ventana en la que se puede configurar los parámetros internos del Controlador, como se muestra en la figura 25. Se puede observar cómo existen dos ventanas separadas, siendo la de la izquierda una representación en forma de bloque de las entradas y salidas al Controlador, mientras que en la ventana de la derecha se encuentran los parámetros de configuración. Bajo la pestaña de IP Solution Options, se selecciona la configuración que tendrá la IP. Para el contexto de este proyecto, se dejan seleccionadas las opciones de Enable Error Injection (Habilitar Inyección de Error) y Enable Error Correction (Habilitar Corrección de Error). En el apartado de Error Correction Method (Método de Corrección de Error) se deja el que viene por defecto, que es Repair (más adelante se dará la explicación del por qué). La frecuencia de reloj a la que funcionará el Controlador debe ser el mismo que ordene el SoC para que exista sincronización que, en este caso, se ha optado por una frecuencia de 100MHz de reloj para el conjunto del sistema. Terminada esta fase, se presiona OK y se abre una nueva ventana llamada Generate Output Products (Generación de Productos de Salida) como la mostrada en la figura 26.

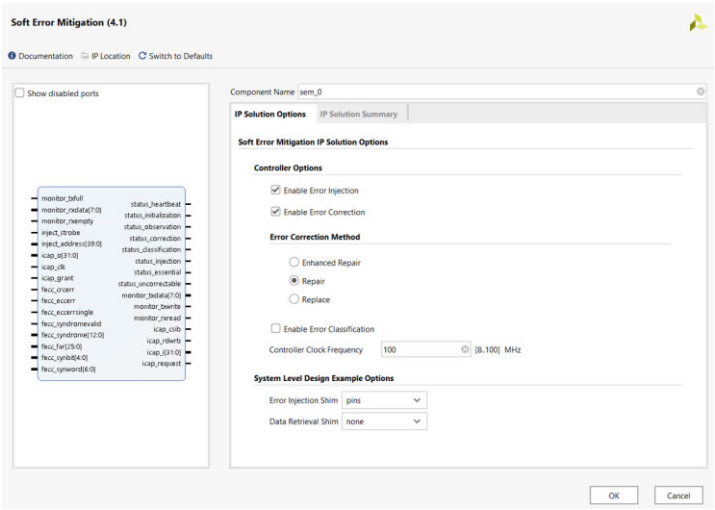


Figura 25. Configuración del Controlador SEM IP

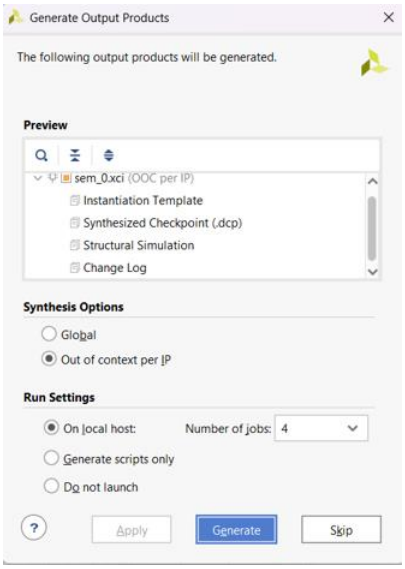


Figura 26. Generación de los Productos de Salida

La generación de los Productos de Salida incluye archivos de código fuente (en VHDL), plantillas de instanciación (proporcionan la estructura necesaria para integrar el IP con el resto del circuito), puntos de control de síntesis (archivos generados después de la síntesis del diseño para retomar el proceso de diseño desde el punto de la síntesis sin tener que repetir pasos anteriores) y archivos de simulación estructural (contienen una descripción a nivel de puertas lógicas y conexiones del diseño), entre otros.

En este punto, ya se tiene acceso al Controlador SEM que se acaba de crear como se muestra en la figura 27. Para ello, presionamos (click derecho) sobre el módulo que se acaba de crear bajo la pestaña de **Sources (Fuentes)** > **Design Sources** y tendrá el nombre que se le haya asignado cuando se configuró el Controlador, en este caso, "sem_0". De las opciones que se despliegan se elige **Open IP Example Design** para poder acceder al diseño que se generó del Controlador, abriéndose un nuevo proyecto de Vivado.

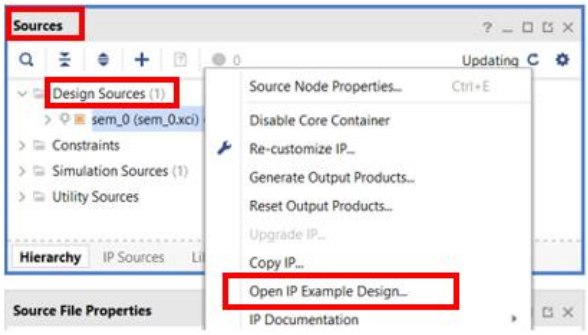


Figura 27. Acceso al Diseño de Ejemplo del Controlador SEM IP

Una vez abierto el proyecto de ejemplo que alberga el diseño del Controlador, se puede aplicar modificaciones a su código fuente, uno de los pasos más para la compleción del diseño de la IP, siguiendo, esta vez, una de las recomendaciones que el diseñador de la IP, Xilinx, presenta a los usuarios. Para ello hay que acceder al código fuente global, es decir, el código de mayor nivel de abstracción del diseño, debajo del cual subyacen los submódulos que conforman el proyecto entero. Lo que se va a modificar a continuación, afectará a la primera de entre varias de las actividades que realiza el controlador internamente, concretamente, se hace referencia al estado de Inicialización, que es el primer estado del Controlador cuando arranca el sistema. Existen un total de siete estados que se puede representar mediante autómatas de estado finito (FSM – Finite State Machine) facilitando el entendimiento de las transiciones que se efectúan entre los distintos estados, que más adelante se expondrán.

Antes de acceder al estado de Inicialización, durante el arranque del sistema, el Controlador SEM supervisa su entrada icap_grant para determinar si se le ha otorgado permiso para entrar al estado de Inicialización y comenzar a utilizar la ICAP. En la mayoría de las implementaciones, se recomienda que la señal icap_grant esté permanentemente activa (es decir, en estado alto o '1' lógico). Sin embargo, las implementaciones en dispositivos Zynq-7000 requieren un manejo especial de esta señal [37]. Prestando atención a las indicaciones y a la figura 28 podemos completar dicho manejo especial de forma satisfactoria.

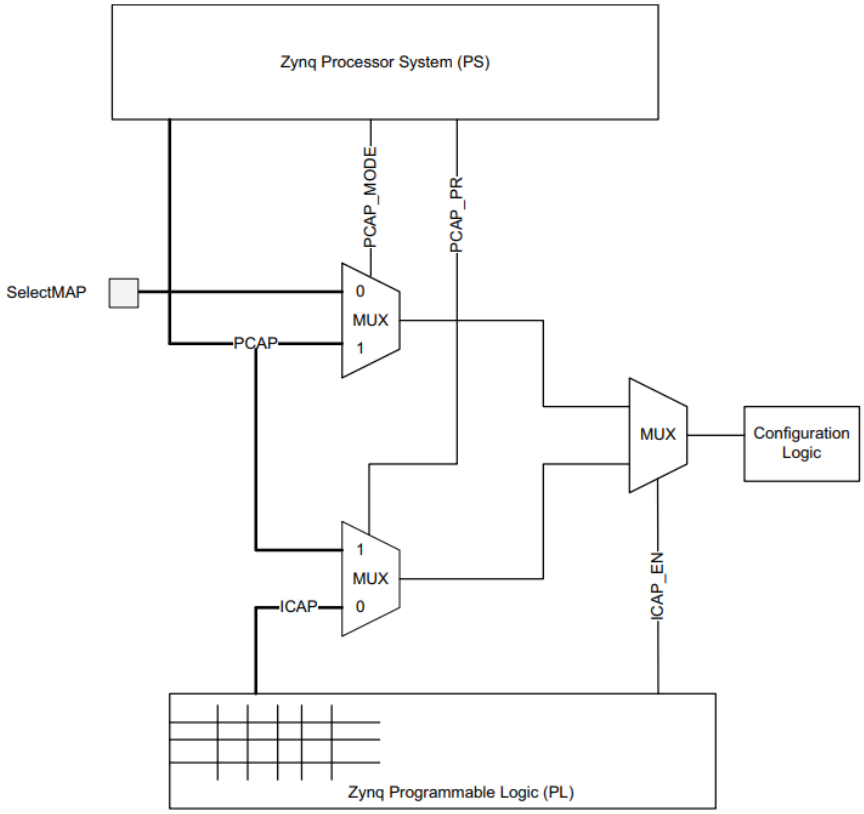


Figura 28. Acceso a la Lógica de Configuración en ZYNQ-7000

Se tiene, por un lado, en el bloque rectangular superior al Microprocesador (PS) y, por otro lado, al bloque rectangular inferior que hace referencia a la Lógica Programable (PL) o FPGA. En los dispositivos Zynq-7000, durante el arranque del Sistema de Procesamiento (PS), el acceso a la lógica de configuración se asigna a la PS a través del puerto de acceso de configuración del procesador (PCAP – *Processor Configuration Access Port*). Esto permite que el cargador de arranque de la PS descargue un bitstream (el que se genera a partir del diseño creado en Vivado) en la Lógica Programable (PL). Al finalizar el cargador de arranque de la PS, tanto la PS como el PCAP mantienen el control sobre la lógica de configuración, permitiendo la reconfiguración parcial de la PL por parte del PS. No obstante, mientras el PS y el PCAP controlan la lógica de configuración, tanto la PL como el ICAP están bloqueados para acceder a esta lógica. Debido a que el Controlador SEM IP se aloja en la Lógica Programable, se debe garantizar el acceso a ella, de lo contrario el Controlador no funcionaría. Para que el controlador funcione adecuadamente, es necesario transferir el acceso a la lógica de configuración al ICAP, lo cual se logra borrando el bit PCAP_PR (bit 27) en el registro de control de configuración del dispositivo PS (DEVCFG CTRL, dirección 0xF8007000).

El controlador no tiene un método simple para detectar si la ICAP está bloqueada. Durante el estado de inicialización, el controlador sondea la ICAP intentando leer el registro IDCODE de la lógica de configuración, hasta que detecta el valor de identificación del fabricante esperado. Sin embargo, si el PS borra PCAP_PR mientras el controlador intenta acceder al ICAP, la lógica de configuración podría recibir una transacción ICAP malformada, lo que resultaría en un comportamiento impredecible.

Para eliminar esta posibilidad, es necesario que la PS controle la entrada icap_grant del controlador a través un GPIO (General Purpose Input Output – Entrada-Salida de Propósito General) y prevenga que el controlador entre en el estado de inicialización hasta que PCAP_PR haya sido borrado. El GPIO utilizado puede ser un EMIO del PS o un GPIO en la PL (se ha escogido esta segunda opción), pero debe ser inicializado de modo que el controlador observe que la icap_grant esté desactivada inmediatamente después la configuración de la PL.

Cuando el software en el PS ha completado toda la actividad necesaria de PCAP, este borra el bit del registro PCAP_PR y luego activa el GPIO conectado a la entrada icap_grant del controlador, permitiendo que el controlador proceda con la inicialización. La señal aplicada a la entrada icap_grant debe estar sincronizada adecuadamente con la señal icap_clk.

La implementación de la explicación anterior se logra mediante la modificación combinada del código fuente tanto del Controlador (en VHDL), ubicado en la PL, como el del software (en el lenguaje de programación C), ubicado en la PS.

En la figura 29 se muestra parte del código del diseño global donde se precisa realizar la modificación sobre la señal icap_grant para que esta sea una señal de entrada externa que se gestiona desde el software que está alojado en la PS, garantizando que el SEM IP solo intente acceder al ICAP cuando tenga permiso explícito para hacerlo, es decir, cuando icap_grant esté en alto (‘1’ Lógico), indicando que el acceso está permitido. El código completo del diseño global del Controlador SEM se puede encontrar en el ANEXO A.

Para añadir un puerto o señal a una entidad, basta con acceder a la sección del código donde se declara la entidad y añadir el nombre de la señal, indicando si es de entrada o de salida y el número de bits de los que constará. Una entidad en Diseño Digital es una descripción de alto nivel de un componente o módulo definiendo sus entradas y salidas, así como su funcionamiento general actuando como una interfaz entre el diseño y el entorno que lo rodea, es decir, cómo se puede interactuar con el componente desde fuera.

```
entity sem_0_sem_example is
port (
  clk                    : in    std_logic;
  status_heartbeat       : out   std_logic;
  status_initialization  : out   std_logic;
  status_observation     : out   std_logic;
  status_correction      : out   std_logic;
  status_classification  : out   std_logic;
  status_injection       : out   std_logic;
  status_essential       : out   std_logic;
  status_uncorrectable   : out   std_logic;
  inject_strobe          : in    std_logic;
  inject_address         : in    std_logic_vector(39 downto 0);
  monitor_tx             : out   std_logic;
  monitor_rx             : in    std_logic;

  -- Haciendo la señal "icap_grant" una entrada externa
  icap_grant             : in    std_logic
  );
end entity sem_0_sem_example;
```

Figura 29. Adición de la señal *icap_grant* como entrada externa de 1 bit.

Prosiguiendo con la edición del código, una vez que *icap_grant* deja de ser una señal interna, para no perder su funcionalidad anterior, se crea otra señal que la sustituya, llamándola al

iguales que otras señales internas con la terminación ***_internal***, quedando como ***icap_grant_internal***. Esta señal interna se añade a la sección donde se encuentran declaradas las demás señales internas bajo el comentario "*Declare signals*" como se puede ver en la figura 30.

```
-- Manteniendo la funcionalidad interna con nombre diferente por la duplicidad.
-- Antes: signal icap_grant_internal    : std_logic;
-- Ahora:
signal icap_grant_internal    : std_logic;

signal icap_clk               : std_logic;
```

Figura 30. Conservación de la señal *icap_grant* como *icap_grant_internal*.

Para finalizar, teniendo las dos señales icap_grant y icap_grant_internal, para que cualquier transición en el estado de la icap_grant (controlada desde fuera como señal externa por parte del software en la PS), sea vea automáticamente reflejada en la señal icap_grant_internal, se realiza un mapeo (ver figura 31) para indicar que dichas señales se interconectan y que tendrán el mismo nivel lógico.

```
icap_rdwrb => icap_rdwrb,
icap_clk => icap_clk,
icap_request => icap_unused,

-- Mapeando la señal "icap_grant" externa a su análoga interna "icap_grant_internal"
icap_grant => icap_grant_internal

);

-- Antes: icap_grant <= '1';
-- Ahora:
-- Para establecer una sincronización correcta de la señal icap_grant
-- con el reloj icap_clk empleamos el reloj global clk_ibufg que
-- a su vez alimenta la señal icap_clk.
icap_sync_proc:
process(clk_ibufg)
begin
  if clk_ibufg'event and clk_ibufg = '1' then
      icap_grant_internal <= icap_grant;
  end if;
end process;

status_heartbeat <= status_heartbeat_internal;
```

Figura 31. Mapeo de icap_grant_internal con icap_grant y la sustitución de la habilitación permanente de icap_grant

En lo que respecta a la arquitectura del sistema descrito, cabe destacar una modificación significativa en el manejo de las señales. Originalmente, la señal denominada icap_grant estaba configurada para permanecer activa de manera continua, es decir, su estado lógico se mantenía constante en '1'. Esta funcionalidad ha sido reemplazada, permitiendo que la activación de la nueva señal interna, icap_grant_internal, pueda ser habilitada o deshabilitada. Tal cambio se efectúa en respuesta a la señal externa icap_grant y en sincronización con la señal de reloj global clk_ibufg, que también coordina el reloj icap_clk. Anteriormente, la señal icap_grant debía sincronizarse con icap_clk; sin embargo, dado que icap_clk a su vez depende del reloj clk_ibufg, se ha decidido implementar una sincronización global utilizando el último, lo cual no presenta repercusiones adversas sobre el sistema.

Tras la implementación de las modificaciones previamente delineadas, se procede a la conservación de estos cambios. En este punto, se dispone de la capacidad necesaria para

proceder con el empaquetamiento del Controlador SEM IP, preparándolo así para su integración en un diseño más amplio que incorporará submódulos adicionales.

Desde el proyecto ejemplo que se abrió y se editó el código del Controlador, se accederá a la pestaña de **Tools > Create and Package New IP** y se debe elegir una ruta para guardar la IP para su posterior importación al catálogo de IPs.

La figura 32 muestra el aspecto del bloque del Controlador SEM IP que se acaba de crear y tras importarlo en un nuevo proyecto.

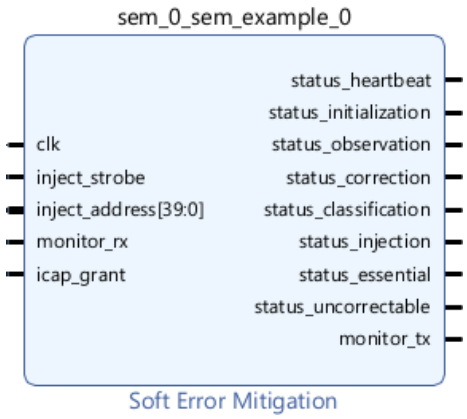


Figura 32. Bloque Controlador SEM IP

A continuación, se describe el funcionamiento del Controlador SEM IP para posteriormente exponer su integración con el resto del diseño o con el resto de los bloques IP que conforman el diseño completo. La figura 33 es el diagrama de autómatas de estado formado por siete estados.

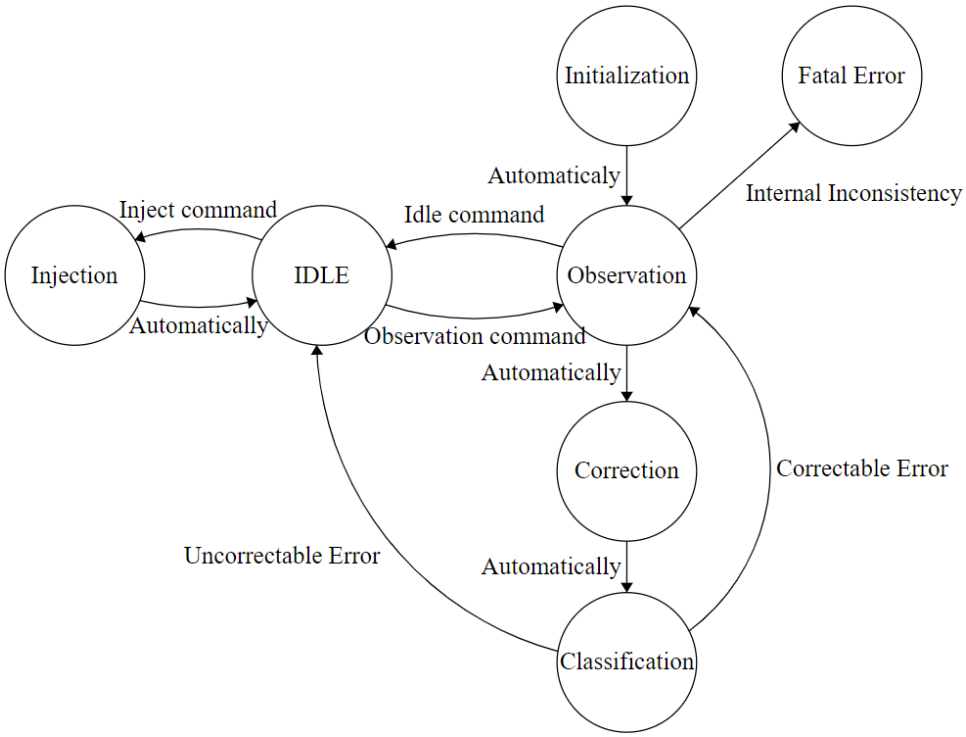


Figura 33. Diagrama de los Autómatas de Estado del Controlador SEM IP

El comportamiento interno del Controlador SEM IP se rige fundamentalmente por el estado en que se encuentra en cada momento específico, y se puede describir de la siguiente manera, empezando por el estado de Inicialización:

- Initialization: Previamente se describieron todos los pasos que debían ocurrir antes de entrar al estado de Inicialización, es decir, que se concediera el control sobre el Controlador SEM IP. Durante el arranque del sistema, el Controlador chequea la señal de entrada icap_grant (grant, del inglés, conceder; otorgar) que como su nombre indica, concede el permiso para entrar al estado de Inicialización y para manejar el puerto de acceso a la configuración interna (ICAP), cuando pasa al estado lógico alto (1 Lógico).

  La interfaz ICAP permite leer o escribir a la Configuración de Memoria (Configuration Memory) de la FPGA o PL. Su importancia radica en que a través de ella se pueden realizar cambios en la lógica del circuito, ajustar la configuración o incluso alterar la funcionalidad de los módulos en tiempo real, que en condiciones de funcionamiento sin intervención sería inmutable.

- Observation: A este estado se llega automáticamente una vez que la Inicialización ha completado. El Controlador pasa la mayor parte del tiempo en este estado. En este estado, el puerto status_observation siempre está en el nivel lógico '1' indicando que el Controlador se encuentra en observación de la Memoria de Configuración de la FPGA por si ocurre algún posible error. Si no hay errores y el Controlador recibe un comando de la interfaz de Inyección de Error, entonces, se sale de este estado y procesa el comando. Los comandos que se pueden recibir en el estado de Observacións son:
  - Enter IDLE: Este comando se aplica en la Interfaz de Inyección para pasar al estado de IDLE (estado ocioso, o de "no hacer nada") con el propósito de que otros posibles comandos se puedan aplicar sobre el Controlador que, de otro modo, no sería posible.
  - Status Report: Comando que provee de información de diagnóstico y es un mecanismo útil para hacer un sondeo sobre el Controlador y ver en qué estado se encuentra. Este comando solamente se puede utilizar mediante la Interfaz del Monitor, que queda fuera de las pretensiones de este proyecto.

  Si se detecta algún error, el Controlador lee información adicional del hardware para prepararse a intentar corregir el error. Para una posible corrección de error, el Controlador pasa al estado de Corrección (Correction).

- Correction: En este estado, se intenta corregir errores detectados durante el estado de Observación. El Controlador siempre pasa por este estado de Corrección. Una vez en este estado, la señal de estado status_correction siempre está en el estado lógico alto.

  - Si el error es de CRC solamente, el Controlador pone la señal de status_uncorrectable a '0' o '1' indicando si el error se puede corregir o no, respectivamente. Y después, pasaría al estado de Classification, que, en lo que atañe a este proyecto, se dejó deseleccionado cuando se configuró el Controlador SEM IP.
  - Si el error no es CRC solamente, entonces el comportamiento del Controlador dependerá de cómo ha sido configurado para corregir los errores. En este proyecto, se dejó por defecto la corrección por reparación (Repair) que intenta corregir los errores mediante métodos algorítmicos, aunque no se entrará los detalles de dichos algoritmos. Si el error es corregible, el Controlador realiza una reconfiguración parcial activa y reescribe la trama de bits alterados con el contenido corregido y reconfigura la señal de status_uncorrectable. Sin embargo, si el error no se puede corregir, dicha señal queda en estado lógico alto. En ambos casos, el Controlador genera un reporte y pasa al estado de Classification.
- Classification: Este es el estado en el que se clasifican los errores. El Controlador pasa por este estado, aunque la clasificación de errores esté desactivada, como se ha hecho en este proyecto. Durante este estado, la señal de status_classification está en un nivel lógico alto.
  - Todos los errores que se clasifican como incorregibles son errores esenciales. El único motivo por el que un error no se puede corregir es porque no se puede localizar o ubicar en la Memoria de Configuración de la FPGA.Y si este es el caso, el Controlador no puede buscar el error para determinar si es esencial o no lo es. En estos casos, el Controlador pone la señal status_essential en un nivel lógico alto, genera un reporte y cambia al estado de IDLE. Además, si el Controlador se encuentra con un error incorregible, no continúa buscando errores, sino que habrá que reconfigurar la FPGA mediante un reinicio de esta.
  - Los errores que sean clasificados como corregibles durante el estado de Corrección depende de la configuración de la opción del Controlador. Si la clasificación de errores está desactivada, todos los errores corregibles se clasifican como esenciales, por tanto, no serían corregibles. Si, por el contrario, la clasificación de errores está activada, el Controlador genera una petición de datos de clasificación en la Interfaz Fetch. En cualquier caso, el Controlador genera un reporte, cambia el valor de status_essential a un nivel lógico bajo y pasa al estado de Observación para seguir buscando errores.

- IDLE: Este estado es similar al estado de Observación, exceptuando que en este estado no se está observando la Memoria de Configuración de la FPGA para indicar alguna condición de error. Si se recibe un comando en el estado de IDLE, que será a través de la Interfaz de Inyección de Error (en el caso del proyecto) o a través de la Interfaz Monitor) y el Controlador procesa el comando. En este estado, solamente se pueden usar los comandos de Error Injection o de Software Reset. El estado IDLE se indica

mediante la desactivación de los cinco bits de estado en el Status Interface (Interfaz de Estado).

- El comando de "Enter Observation" (Entrar en el estado de Observación) se emplea para devolver al Controlador al estado de Observación para poder detectar errores.
- El comando de "Status Report" (Reporte del Estado) provee de información de diagnóstico, una manera de conocer el estado en el que se encuentra el Controlador. Este comando solo se aplica desde la Interfaz Monitor.

Cualquier conjunto de comandos de Error Injection se puede aplicar mediante la Interfaz de Inyección de Error o mediante la Interfaz Monitor. Estos comandos llevan al Controlador a realizar las inyecciones de errores.

La razón de la existencia de este estado de IDLE es para detener temporalmente las acciones tomadas en respuesta a detecciones de errores. Esto se hace para permitir la construcción o corrección de errores que involucran múltiples bits.

- El comando de "Software Reset" se puede aplicar tanto a través de la Interfaz de Inyección de Error como a través de la Interfaz Monitor. Este comando lleva al Controlador a realizar un reinicio del software.

- Injection: Es en este estado cuando el Controlador puede inyectar los errores. El Controlador siempre pasa por este estado en respuesta a un comando válido de Error Injection desde el estado de IDLE, aunque Error Injection esté deshabilitado; esto puede ocurrir si los comandos de inyección de error se presentan en la Interfaz Monitor, ya que esta interfaz existe incluso cuando la opción de inyección de errores esté desactivada.

  El proceso de inyección de errores desde que se ordena hasta que se inyecta el error en la Memoria de Configuración consiste en leer-modificar-escribir para invertir un bit de la dirección especificada como parte del comando de inyección de error. Tras la inyección, el Controlador siempre pasa del estado de Inyección a IDLE.

  Se pueden construir errores múltiples, es decir, inyectar errores sobre múltiples bits en lugar de sobre un solo bit, repitiendo en bucle los comandos de inyección de errores, suponiendo cada inyección una transición al estado de Inyección. Al finalizar la inyección de errores, el Controlador debe transitar del estado de inactividad al estado de Observación nuevamente.

- Fatal Error: El Controlador entra en este estado cuando detecta una inconsistencia interna. Aunque es poco probable, es posible que el Controlador se detenga debido a soft errors que afecten a la Memoria de Configuración relacionada con el Controlador o los elementos del estado de diseño del Controlador.

El estado de Fatal Error se indica mediante la activación de los cinco bits de estado de la Status Interface junto con un mensaje de reporte. Esta condición no es recuperable y la FPGA debe ser reconfigurada.

Es importante rectificar una percepción equivocada con respecto a la influencia de la inyección de errores por medio del Controlador SEM IP en el software. La mencionada inyección de errores no incide directamente sobre el código de software generado tras la creación del Bitstream. Esto se debe a que las variables del código, una vez compiladas y traducidas a lenguaje máquina, residen en la memoria RAM del sistema durante la ejecución, y no en la configuración hardware de la FPGA ni, mucho menos, en la Memoria de Configuración de la FPGA, que es precisamente donde se inyectan los errores. La alteración provocada por la inyección de errores en el Bitstream afecta únicamente la configuración del hardware, mas no la del software. Asimismo, es pertinente enfatizar que el propósito fundamental del Controlador SEM IP es monitorear, inyectar y corregir errores en la Memoria de Configuración, asegurando la integridad y el correcto funcionamiento del hardware en entornos susceptibles a perturbaciones.

En la fase de integración del Controlador SEM IP al diseño completo, se realizará una exposición detallada de su funcionamiento práctico. Se abordarán específicamente los mecanismos por los cuales este dispositivo transita de un estado a otro, así como los comandos que facilitan la inyección de errores en la configuración hardware. Este análisis incluirá una descripción de los comandos operativos que permiten activar y controlar estas transiciones y acciones, proporcionando así una comprensión clara de cómo el Controlador SEM IP interactúa con el sistema más amplio.

4.7.2 Integración del Controlador SEM IP al diseño completo

Como se mencionó anteriormente, se han llevado a cabo dos diseños, de los cuales, en este apartado, se expondrá la solución que da el fabricante Xilinx con la tecnología SEM IP que se integrará en una solución mucho más amplia. En el diseño en cuestión, desarrollado en Vivado Xilinx, se observa una arquitectura meticulosamente estructurada que se enfoca en la utilización e interconexión óptima de diversos bloques IP (Propiedad Intelectual), centralizando su funcionalidad en torno al Controlador SEM IP.

El diagrama de bloques, referido en la figura 34, ilustra la integración sistemática de estos módulos IP, cada uno con funciones específicas y complementarias. La arquitectura completa no solo es una representación de interconexión, sino también una demostración de cómo el diseño modular puede facilitar la expansión y la escalabilidad en aplicaciones de hardware.

Posteriormente, el manejo de este hardware se realizará a través de Vitis IDE, también de Xilinx, lo que proporciona un entorno integrado para el desarrollo que soporta tanto el software de alto nivel como las tareas específicas de programación del hardware.

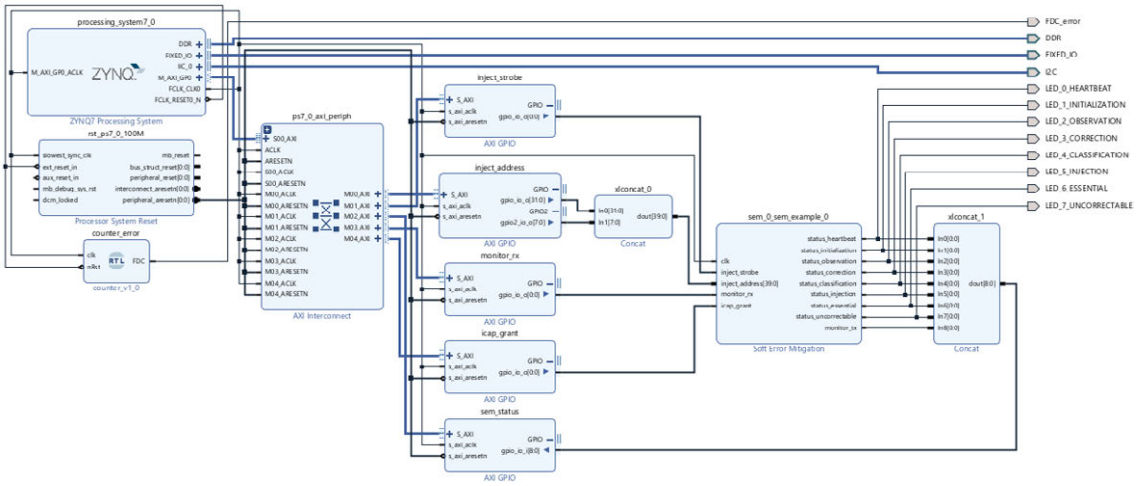


Figura 34. Diagrama de Bloques del Hardware del Diseño en Vivado

A continuación, se procederá a dar una explicación breve de cada componente y su relación de interconexión con el resto de los bloques:

- ZYNQ7 Processing System (Sistema de Procesamiento ZYNQ7): El Processing System IP Zynq7 (abreviatura de la familia Zynq-7000) es la interfaz de software alrededor del Sistema de Procesamiento (PS) de Zynq-7000. El Wrapper (Envoltorio) del Processing System IP actúa como una conexión lógica entre el PS y el PL, asistiendo en la integración de IPs personalizados y embebidos con el sistema de procesamiento utilizando el integrador de IP de Vivado. En este contexto, el término Wrapper o envoltorio se refiere a una capa de abstracción que encapsula la funcionalidad de un componente más complejo. En el caso del Wrapper ZYNQ7 Processing System, este actúa como una envoltura alrededor del Processing System (PS) del Zynq-7000 [38].
- Processor System Reset (Reinicio del Sistema de Procesamiento): El módulo central del Xilinx LogiCORE™ IP Processor System Reset proporciona mecanismos de reinicio personalizables para un sistema de procesador completo, incluyendo el propio procesador, las interconexiones y los periféricos. Este núcleo permite a los usuarios adaptar sus diseños a las necesidades específicas de sus aplicaciones mediante la configuración de ciertos parámetros para habilitar o deshabilitar características [39].
- AXI Interconnect (Advanced eXtensible Interface Interconnect – Interconexión de Interfaz Avanzada y Extensible): Es una entidad de diseño jerárquico utilizada exclusivamente dentro del entorno del integrador de IPs en Vivado. Actúa como un conducto configurado y conectado durante la sesión de diseño del sistema, permitiendo la interconexión de múltiples dispositivos maestros y esclavos AXI que pueden diferir en ancho de banda, dominio de reloj y subprotocolos AXI específicos. Este núcleo facilita la comunicación entre dispositivos que de otro modo serían incompatibles, mediante la inferencia y conexión automática de núcleos de infraestructura apropiados que realizan las conversiones de interfaz necesarias. En resumen, el 'AXI Interconnect' es fundamental para la cohesión arquitectónica y funcional de un sistema embebido complejo diseñado con Vivado [40].

Cabe mencionar que tanto Processor System Reset como AXI Interconnect son módulos generados automáticamente (opción seleccionable en el diagrama de bloques bajo el nombre de Run Connection Automation) cuando el resto de los bloques IP han sido insertados al diseño (también desde el diagrama de bloques bajo el nombre de Add IP) que se encuentran almacenados en el catálogo de IPs por defecto del entorno de Vivado o como el personalido, SEM IP en un directorio de trabajo especificado en el momento de la generación del paquete IP del Controlador.

- AXI GPIOs (Advanced eXtensible Interface General Purpose Input/Output – Interconexión de Interfaz Avanzada y Extensible para Entradas y Salidas de Propósito General): El diseño de AXI GPIO proporciona una interfaz de entrada/salida de propósito general hacia una interfaz AXI Interface. Este dispositivo puede ser configurado como un canal único o doble, con la anchura de cada canal siendo ajustable de manera independiente. Esta estructura permite una flexibilidad y adaptabilidad significativas en la gestión de señales de entrada y salida en sistemas basados en AXI. En el diagrama de bloques del diseño hay cinco y todos van conectados a las señales de entrada del Controlador SEM IP, que son:
    - inject_strobe: GPIO de salida de 1 bit manejado desde el software que sirve para habilitar la inyección de errores una vez que se haya indicado la dirección donde se va a efectuar tal acto.
    - inject_address: GPIO de salida de 40 bits manejado desde el software también, que entrega al Controlador SEM IP la dirección donde se van a inyectar los errores.
    - monitor_rx: GPIO de salida de 1 bit que, utilizando la Interfaz de Monitorización y las señales monitor_rx y monitor_tx, la función supervisora a nivel de sistema puede transmitir periódicamente un comando de estado y confirmar la recepción del reporte de estado esperado. Sin embargo, esta funcionalidad queda fuera del alcance de este proyecto y, por tanto, no se usa. Forma parte del diseño debido a que es una parte inextricable del Controlador SEM IP y si no tiene un manejo adecuado, el Controlador no funcionaría y es por ello que se optó por asignarle un GPIO, aunque jamás se accederá ni se modificará el estado de esta señal.
    - icap_grant: GPIO de salida de 1 bit del mismo nombre que la señal que otorga el permiso para el manejo de la ICAP. Señal controlada desde el software de la PS.
    - sem_status: GPIO de entrada de 9 bits que coincide con el número de las señales de estado (8 bits) del Controlador SEM IP además de su salida monitor_tx de 1 bit. Se implementa este GPIO con el propósito de sondeo interno desde el software de la PS para tener información de en qué estado se encuentra el Controlador en cada instante.
- Concat: En el diseño presente, se utilizan dos bloques denominados Concat, específicamente xlconcat_0 y xlconcat_1, cuya funcionalidad esencial, conforme a su designación, consiste en la concatenación de señales. Esta capacidad se aplica

particularmente en el contexto de xlconcat_0, que se encarga de concatenar dos puertos que constituyen el GPIO para la señal inject_address. Esta señal es de suma importancia, pues especifica la dirección dentro de la Configuración de Memoria donde se procederá a la inyección de errores.

Dado que, por diseño, la señal inject_address debe tener una longitud de 40 bits y los GPIO estándar tienen una limitación de 32 bits, se hace necesario realizar una expansión interna del GPIO. Esta expansión incorpora 8 bits adicionales, con el objetivo de completar los 40 bits requeridos. Es crucial, para la correcta operación del sistema, que estos segmentos de bits sean tratados y transmitidos como una entidad única; la división de la señal en dos componentes (32 + 8 bits) no sería funcional ni aceptable para el Controlador SEM IP, el cual requiere una recepción unificada de 40 bits. Por consiguiente, se recurre al uso del bloque Concat para fusionar todos los bits necesarios y proporcionar una salida integrada de 40 bits.

Por otro lado, el bloque xlconcat_1 desempeña un papel distinto, pero igualmente crítico. Este se encarga de unificar las señales de estado provenientes del Controlador SEM IP. Estas señales, una vez unificadas, son dirigidas hacia un GPIO de entrada denominado sem_status. Esta configuración facilita el acceso desde el software a cada una de estas señales, permitiendo una supervisión y manejo efectivo de la información de estado relevante al funcionamiento del sistema. Además, dentro de este esquema de conexión, se incluye la señal de monitor_tx. Sin embargo, es importante señalar que, en el contexto de este proyecto específico, la señal de monitor_tx no se encuentra en uso.

- SEM IP: La SEM IP está interconectada mediante interfaces de propósito general de entrada y salida, las cuales facilitan la transferencia de señales pertinentes a la inyección de errores y al monitoreo del estado operativo del sistema. Esta IP es de carácter central para el proyecto en cuestión, y su relevancia e implicaciones han sido objeto de un análisis pormenorizado previamente. Información adicional referente a esta IP será proporcionada en el momento apropiado, siguiendo una secuencia lógica y metódica acorde con la estructura de este discurso.
- counter_error: Se ha concebido un bloque IP de diseño propio, cuya implementación interna se ha realizado utilizando el lenguaje VHDL. Este módulo se caracteriza por ser un contador lineal que opera en sincronía con el reloj del sistema global establecido a una frecuencia de 100 MHz, equivalente a 10^8 ciclos por segundo. Así, al completar una cuenta de 10^8 ciclos en este reloj, se consigue demarcar un intervalo de tiempo equivalente a un segundo.

  El cometido de esta construcción es doble. Primero, proporcionar una señal de salida que oscile entre ‘0’ y ‘1’, es decir, transite de un estado lógico bajo a uno alto y viceversa, en una cadencia de 1 Hz. Este cambio de estado se manifiesta visiblemente a través de la activación de uno de los diodos emisores de luz (LED)

en la Blackboard, produciendo así un efecto de parpadeo que puede ser observado con facilidad.

La razón subyacente para esta manifestación visual no es meramente estética, sino funcional, pues permite el diagnóstico inmediato y directo de la integridad del contador. Cualquier inyección de error que perturbe la secuencia de conteo alteraría la frecuencia del parpadeo del LED, desviándolo de su ritmo establecido de 1 Hz. Esta propiedad convierte al módulo no solo en una herramienta de temporización, sino también en un instrumento de diagnóstico para validar el comportamiento correcto y la robustez del sistema ante perturbaciones inesperadas.

- LEDs de estado: Si bien, las señales que gobernarán el encendido y apagado de los LEDs de la Blackboard no son Bloques IP sino meras señales, se indican como flechas en forma de pentágono como se muestra en la figura 35.

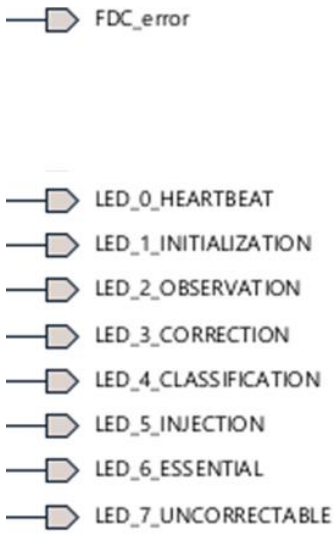


Figura 35. LEDs de diagnóstico del estado del Controlador SEM IP

Cada diodo LED se encuentra en una conexión directa con un estado distinto del Controlador, con el propósito de suministrar una indicación visual precisa del estado operativo actual en el que se halla el Controlador. El encendido de un LED específico sirve como indicador, señalando que el Controlador está en el estado correspondiente al nombre asociado con ese LED en particular. Esta metodología permite una interpretación inmediata y clara de la condición del sistema sin la necesidad de herramientas de diagnóstico adicionales.

El estado de activación del diodo LED, que responde al cambio de estado del contador previamente mencionado, se encuentra regulado por la señal designada como FDC_error. El acrónimo FDC denota “Fin De Cuenta”, sugiriendo que esta señal representa la culminación de un ciclo de conteo. La adición del término ***_error*** a la señal, por otra parte, delata que dicha señal es susceptible a ser afectada por errores intencionalmente inyectados en el sistema. Este mecanismo de denominación no sólo esclarece la función de la señal, sino que también advierte sobre su rol específico en la simulación o detección de fallos dentro del contexto de la arquitectura diseñada. La figura 36 muestra las señales que la ejecución de conexión automática de Vivado genera y se da una breve descripción de estas señales a continuación:

- Bus de señales DDR, FIXED_IO e I2C: El Bus DDR (Double Data Rate – Tasa de Datos Doble) actúa como interfaz entre la FPGA y la memoria chip DDR3 externa a Zynq permitiendo la ejecución de algoritmos avanzados que dependen del acceso inmediato y simultáneo a grandes volúmenes de datos. Esta interfaz aprovecha, a diferencia de su precursor SDR (Single Data Rate – Tasa de Datos Simple), el flujo de datos tanto en los flancos de subida como en los de bajada del reloj, que, en el caso de este proyecto, se ha generado automáticamente, aunque no se hace uso de este bus, debido a que no se necesita hacer uso de ello.
- El Bus FIXED_IO (Entrada-Salida Fijos) se refiere al conjunto de periféricos integrados en el chip, en este caso en el SoC Zynq, que están disponibles para ser habilitados durante la configuración del bloque IP Zynq7 Processing System. Estos periféricos fijos incluyen interfaces de comunicación, puertos de entrada/salida y otras funciones esenciales que están físicamente conectadas a pines específicos del chip y no pueden ser reasignadas a otros pines. En otras palabras, son “fijos” en el sentido de que sus asignaciones de pines no cambian y su configuración en términos de conexión física es constante. Sin embargo, este bus tampoco ha sido necesario utilizarlo, pero merece su mención debido a que se ha generado automáticamente y forma parte del diseño.
- El Bus I2C emplea el protocolo de dicho nombre (I2C), que a través de sus líneas —SCL para el reloj y SDA para los datos—, posibilita la transmisión de información entre diversos dispositivos electrónicos que emplean el mismo protocolo. En el contexto de nuestro diseño, incorporado en la tarjeta Blackboard, las líneas I2C han sido instauradas no como un componente activo y esencial del funcionamiento actual, sino como un elemento de prueba y experimentación, con la perspectiva de que pudieran servir como cimientos para futuras expansiones o iteraciones del proyecto.

Figura 36. Buses DDR, FIXED_IO e I2C

Una vez expuesto cada uno de los bloques que forman parte del diagrama de bloques del diseño, cabe señalar que, en dicha representación gráfica, cada bloque IP es una abstracción de una funcionalidad o una unidad de hardware dentro del SoC ZYNQ-7000. Sin embargo, estos componentes por sí mismos carecen de la instrucción operativa necesaria para realizar alguna tarea significativa en el dominio del hardware. Es a través de la incorporación de software, específicamente el software de síntesis y de implementación que proporciona Vivado, que estos esquemas se transforman en algo concreto.

La síntesis convierte las instrucciones y comportamientos abstractos en una disposición concreta de puertas lógicas, flip-flops y otras primitivas, o mejor conocido como, una netlist, que es una representación intermedia que detalla la lógica combinatoria, las funciones secuenciales y las interconexiones que deben ser físicamente realizadas en el dispositivo.

La implementación, proceso que sigue a la síntesis, como su nombre indica, traduce la netlist generada por la síntesis en un diseño físico que puede ser programado en el hardware del dispositivo, en este caso, la FPGA o Programmable Logic (PL). Es particularmente un proceso complejo, y por ello, automatizado dentro de Vivado, que de no haber alguno en el diseño, estos procesos se llevarían a cabo mediante simples clicks. Debido a su complejidad, trataremos este proceso en el siguiente subapartado.

4.7.3 Implementación del diseño en Vivado

Antes de adentrarse en presentar la fase de implementación, se muestra en la figura 37 la disposición del SoC ZYNQ-7000 dentro de Vivado.

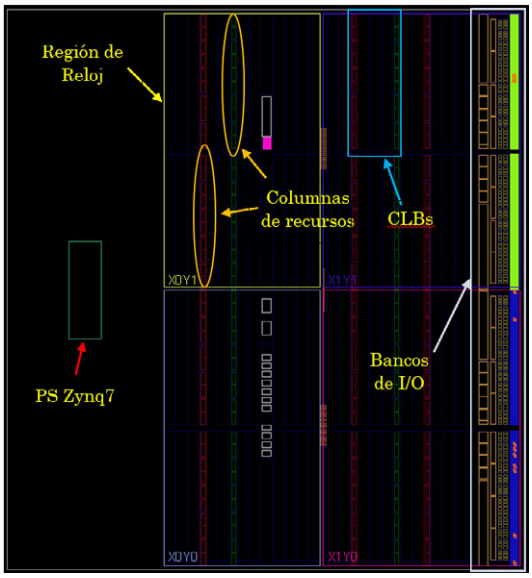


Figura 37. Disposición de la FPGA de la Familia Artix-7 antes de la Implementación

Esta vista es accesible desde la opción de Open Synthesized Design una vez que la síntesis haya sido completada, en la ventana de Device. Se puede observar la inexistencia de conexionado alguno debido a la carencia de recursos asignados a cada componente o bloque IP de los vistos en el diagrama de bloques del diseño.

Se tiene, por un lado, el Processing System (PS) Zynq7 aislado de lo que es, por otro lado, el Programmable Logic (PL) o la FPGA. Esta última, está subdividida en filas y columnas de las regiones de reloj (Clock Region). Una región de reloj contiene bloques de lógica configurable (CLBs), bancos de entrada y salida (I/O Banks), segmentos de procesamiento de señales (DSP), bloques de RAM, recursos de interconexión y de sincronización mediante relojes asociados. Cada banco de I/O contiene pines de entrada capaces de manejar relojes para introducir relojes del sistema o de la placa en el dispositivo y hacia los recursos de enrutamiento de reloj.

Se observan cuatro regiones de reloj etiquetadas como X0Y0, X0Y1, X1YO, X1Y1. Cada una de estas regiones se caracterizan por tener sus propios recursos de reloj.

El avance técnico de estructurar una FPGA en múltiples regiones de reloj, cada una dotada de sus propios recursos de reloj, en contraposición a una FPGA que opera bajo una única región

de reloj, reside fundamentalmente en la optimización del rendimiento y la eficiencia en la gestión de las señales de reloj. En una configuración donde la FPGA dispone de una sola región de reloj para el conjunto del dispositivo, esta debe enfrentarse al problema del skew o desfase de reloj. Este fenómeno de skew se manifiesta cuando distintas partes del chip reciben la señal de reloj en momentos ligeramente desfasados debido a las diferencias en la longitud de los trayectos de reloj y las cargas de interconexión. Tal situación repercute directamente en la frecuencia máxima operativa del dispositivo, puesto que la velocidad de funcionamiento del sistema completo se ve limitada por la del componente más lento. Por lo tanto, las ventajas de implementar múltiples regiones de reloj incluyen la notable reducción del skew; la libertad para seleccionar en qué región de reloj operará un componente específico y, en casos de fallo de componentes en una región, la disponibilidad de otras regiones que pueden continuar operando eficazmente.

Se ha optado por aislar el bloque de counter_error en la región de reloj X1Y1, como podría haber sido cualquiera de las otras tres, para el ejercicio de usar varias regiones de reloj, y desde el punto de vista de que dicho componente es el dispositivo sobre el cual se desearía tener un error, en el contexto del diseño digital, a dispositivos bajo testeo o bajo análisis, se les denomina DUT (Device Under Test). El resto de los componentes se ha dejado a la elección de Vivado que, por defecto, comenzará a colocarlos en la región X0Y0, por ser la posición más baja.

En la Figura 38 se ilustra el conexionado y la colocación de los componentes tras completar la implementación. Es observable cómo se establecen cientos de conexiones y cómo el componente counter_error es asignado específicamente a la región de reloj que le ha sido reservada mediante la herramienta **Draw Pblock**. Esta precisa colocación no es coincidencia, sino el resultado de una implementación previa que verificó el número necesario de bloques lógicos. Adicionalmente, el Controlador SEM IP utiliza dos regiones de reloj contiguas, junto con otros recursos adicionales.

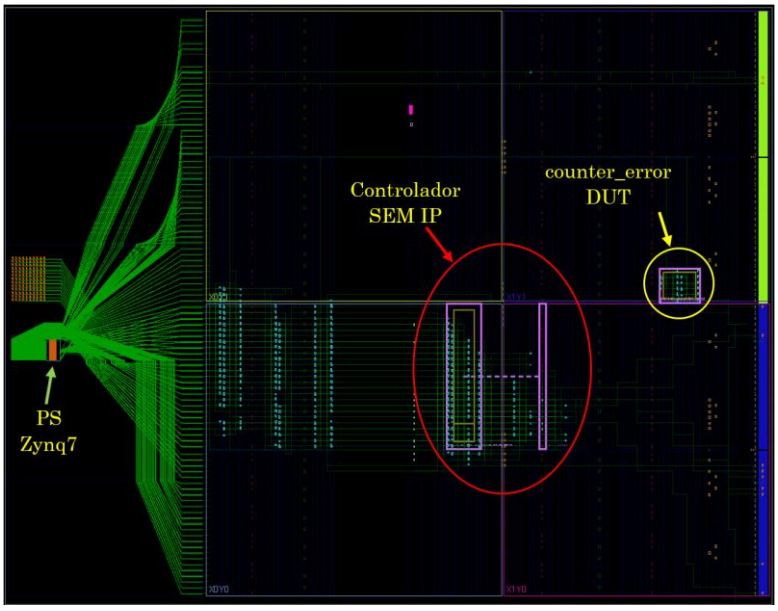


Figura 38. Disposición de la SoC después de la Implementación

Los bloques de GPIO funcionan como interfaces que conectan la FPGA con el procesador del sistema (PS) y las señales que regulan dispositivos periféricos, tales como es el caso de los LEDs, se especifican dentro de un archivo de restricciones, o constraint file. Este archivo es primordial ya que dicta las normativas de asignación de pines y las restricciones eléctricas y temporales aplicables al diseño implementado. El constraint file asegura que todas las interacciones entre la FPGA y los componentes externos, como los LEDs, se adhieran a los parámetros de diseño establecidos para garantizar funcionalidad.

Se muestra en la figura 39 la nomenclatura o instrucción que se ha usado para determinar cómo la señal etiquetada como LED_0_HEARBEAT se conectará al LED del pin físico N20 de la FPGA.

```
set property -dict {PACKAGE_PIN N20 IOSTANDARD LVCMOS33} [get ports LED_0_HEARTBEAT]
```

Figura 39. Línea extraída del archivo de restricciones (.xdc)

La propiedad IOSTANDARD establecida a LVCMOS33 define el estándar eléctrico para ese puerto, que en este caso corresponde a un voltaje de operación de 3.3 V de la tecnología CMOS de baja tensión. Se trata de una instrucción de asignación y configuración eléctrica para el mapeo de hardware en el proceso de síntesis de diseño de la FPGA.

4.7.4 Generación del Bitstream del diseño en Vivado

Una vez completada la implementación del diseño en el SoC, se torna indispensable la generación de un fichero que, de alguna manera, represente la configuración del hardware. El fichero en cuestión es el Bitstream en sistemas basados en FPGA.

En el entorno de desarrollo de Vivado, este paso se ha refinado hasta alcanzar un grado de automatización tal que la generación del Bitstream se efectúa con la mínima intervención del usuario mediante un simple click, pero no debe ser interpretado como una trivialidad, ya que, este paso es el último eslabón que vincula el diseño de alto nivel con la materialización física en el hardware. Sin este, la FPGA no puede ser configurada para desempeñar la función específica para la cual fue programada.

Antes de proceder a la generación del Bitsteam, es de interés para este proyecto, establecer una restricción adicional en el constraints file (utilizado en la fase de síntesis e implementación) para que se genere el fichero imagen de los bits esenciales con la extensión (.ebc) haciendo referencia a (Essential Bits Configuration – Configuración de Bits Esenciales) y otro fichero con extensión (.ebd) que hace referencia a (Essential Bits Data – Datos de Bits Esenciales).

En el examen de los procedimientos que rodean la corrección de errores, particularmente cuando se emplean métodos de sustitución o clasificación junto con estrategias correctivas (a saber, reparación, reparación mejorada o reemplazo), la herramienta BitGen en Vivado se configura para utilizar la opción ***"-g essentialbits:yes***" para la generación de archivos EBC y EBD u optar por incluirlo en el fichero de *constraints* como se muestra en la figura 40.

```
#Essential Bits
set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_design]
```

Figura 40. Instrucción para generar los ficheros EBD y EBC

El fichero EBC actúa como un documento de referencia en código ASCII [41] que encapsula el contenido de las celdas de memoria. Este contenido es el mismo que se lee por el controlador SEM durante la lectura de SEU del FPGA. Es importante señalar que este archivo no es idéntico al Bitstream utilizado para el proceso de configuración del dispositivo.

El fichero EBD muestra los bits de la lectura de SEU que están marcados como esenciales. El fichero EBD se utiliza para enmascarar el fichero EBC, lo que significa que un '1' en el fichero EBD corresponde a un bit esencial en el fichero EBC que puede ser tanto un '1' como un '0'.

Antes de observar los ficheros EBD y EBC generados, un parámetro importante que revisar es el número total de bits esenciales que se han generado, un dato que en lo que respecta a este proyecto, se traduciría como el número de veces totales que se podría llegar a inyectar errores sobre bits esenciales de forma independiente en direcciones diferentes. Este parámetro se encuentra dentro de los ficheros de diagnóstico que la plataforma Vivado genera automáticamente, denominado runme.log donde su contenido arroja los resultados de la figura 41.

```
Creating bitstream...
Writing bitstream ./design_1_wrapper.bit...
Creating bitstream...
Writing bitstream ./design_1_wrapper.ebc...
Creating essential bits data...
This design has 316108 essential bits out of 12924768 total (2.45%).
Creating bitstream...
Writing bitstream ./design_1_wrapper.ebd...
INFO: [Vivado 12-1842] Bitgen Completed Successfully.
```

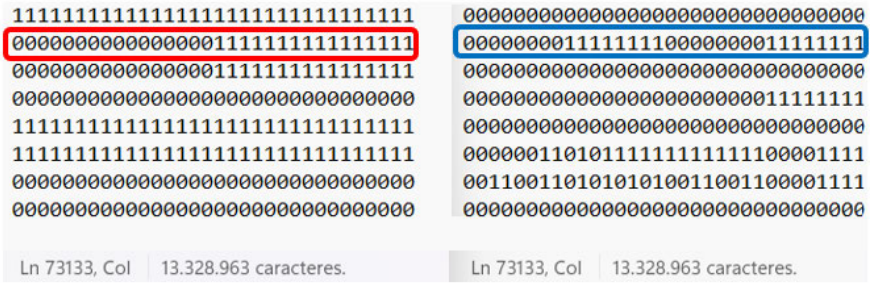
Figura 41. Parte del Diagnóstico de la Generación del Bitstream

Visto en la figura inmediatamente anterior, se tiene en la línea más larga que, el diseño generado alberga un total de 316.108 bits esenciales (2,45%) de entre los 12.924.768 totales (100%) que podría llegar a tener la FPGA. Además de los ficheros EBD y EBC, se genera el fichero con extensión (.bit) que contiene toda la información necesaria para programar el dispositivo.

Desde el punto de vista de la fiabilidad, un menor porcentaje de bits esenciales (2,45%) puede indicar que el diseño es más robusto frente a errores aleatorios causados por SEUs, ya que una menor proporción del total de bits afectaría el comportamiento general del sistema si fueran perturbados. Esto podría sugerir que el diseño es relativamente resistente y puede tolerar cierta cantidad de fallos sin una pérdida de funcionalidad crítica. Sin embargo, también es importante tener en cuenta la distribución y el rol de estos bits esenciales; si están agrupados o asociados con funciones críticas, incluso una pequeña cantidad podría resultar en fallas significativas.

En la práctica, la aceptabilidad de este porcentaje también depende del contexto de aplicación. Por ejemplo, en aplicaciones críticas para la seguridad, como la aviación o la medicina, incluso un pequeño porcentaje de bits esenciales representa un riesgo que debe mitigarse con estrategias adicionales de protección, como la redundancia o la corrección de errores. En aplicaciones menos críticas, un 2,45% podría considerarse un riesgo aceptable.

Con todo esto, se procede a analizar la correlación entre el fichero EBD y EBC mostrado en la figura 42.



a) Fichero EBD **b) Fichero EBC**

Figura 42. Correlación entre los ficheros EBD y EBC

Como era de prever, el fichero EBD cumple la función de una máscara sobre el archivo EBC, de tal modo que un bit con valor '1' en el EBD señala, de manera exclusiva, la presencia de un bit esencial en la ubicación correspondiente del fichero EBC. Esto es, un bit '1' en el EBD indica que su homólogo en el EBC es de importancia crítica en cuanto a vulnerabilidad frente a SEUs se refiere, pudiendo dicho bit esencial en el EBC adoptar el estado de '1' o '0'. La visualización de la información esclarece este punto: la línea resaltada en rojo muestra, en su mitad derecha, bits esenciales; mientras que, en su mitad izquierda, ceros denotan bits de menor relevancia. Paralelamente, la línea resaltada en azul refleja los bits esenciales correspondientes, no obstante, la coincidencia no es completa, sino que se extiende únicamente hasta el decimosexto bit. Esto se debe a que el fichero EBD manifiesta '1s' hasta el bit número 16, determinando así que los subsiguientes hasta el bit 32 son considerados no esenciales.

Entrando en detalles, ambos ficheros mencionados contienen, y deben contener, exactamente el mismo número de líneas, ya que, son complementarios. En lo que respecta a los objetivos de este proyecto, solamente se hará uso del fichero EBD, puesto que, únicamente es necesario conocer las posiciones de los bits esenciales y no sus valores asociados del fichero EBC. Esto es así, debido a que el Controlador SEM IP, requiere solo las posiciones donde se necesitan realizar las inyecciones, encargándose él de localizar el bit e inducir el fallo (bit flip – cambio de estado o valor del bit).

Obsérvese en la figura 43 cómo en el fichero EBD se establece una cabecera informando sobre qué tipo de fichero es, la fecha de generación, tipo, bits totales, etc. Lo que realmente interesa son las líneas de bits (con ‘0s’ y ‘1s’), conteniendo cada línea palabras de 32 bits que

representan un marco dentro de la Configuración de Memoria, siendo el bit de más a la derecha, el bit de menor peso.

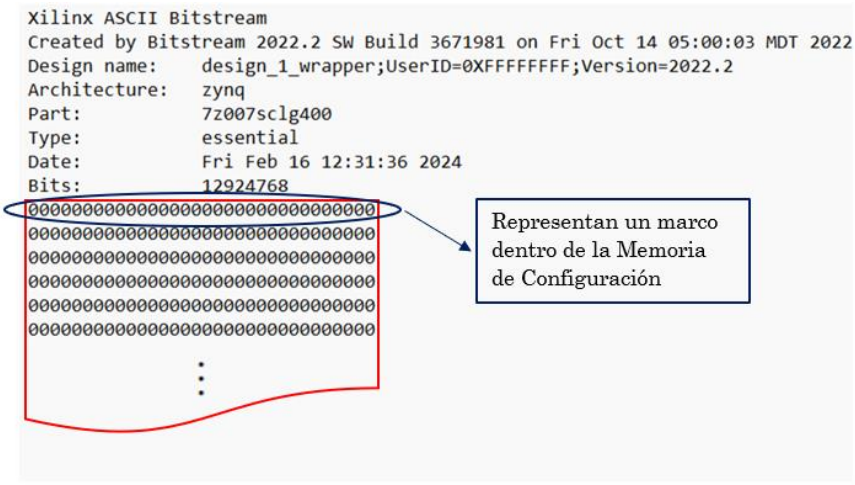


Figura 43. Formato del fichero EBD

En la Memoria de Configuración de la FPGA, un marco (frame) es una unidad de dicha memoria que puede ser direccionada y accedida. Esta memoria está estructurada mediante marcos, que son bloques de bits que contienen información para configurar los bloques lógicos y los recursos de interconexión. La referencia a esquemas de direccionamiento de marcos implica que hay diferentes maneras de identificar y localizar estos marcos para operaciones como la inyección de errores a través del Controlador SEM IP. Estos esquemas, en el caso del dispositivo empleado para este proyecto, son:

- Direccionamiento Lineal de Marcos (Linear Frame Addressing): Es el método más sencillo de emplear que utiliza direcciones secuenciales sin proporcionar información detallada sobre el tipo o la ubicación física del marco dentro de la arquitectura de la FPGA, útil si no se tiene conocimiento de la estructura física del dispositivo. Para el caso de este proyecto, se ha optado por emplear este método debido a su facilidad de manejo. Se muestra en la figura 44 se muestra el formato que debe tener el comando para inyectar un error en una ubicación elegida por parte del usuario.

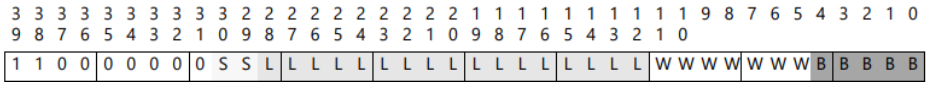


Figura 44. Comando para Inyección de Error mediante Direccionamiento Lineal de Marcos

Tal y como se observa en la figura inmediatamente anterior, se tiene información valiosa sobre cómo está estructurada la Memoria de Configuración (los 9 bits de mayor peso son fijos y no deben ser modificados y los valores que ahí se indican, deben ser insertados en el comando como están), siendo:

  - SS (2 bits): (Hardware Super Logic Region (SLR) for Stacked Silicon Interconnect (SSI) – Región Super Lógica de Hardware para Interconexión de Silicio Apilado). La SSI es una técnica que permite apilar físicamente múltiples trozos de silicio o die, los cuales contienen circuitos integrados, en un solo paquete de

semiconductor. La SLR se refiere a una región dentro de un FPGA que agrupa una cantidad considerable de recursos lógicos y de enrutamiento. En los FPGAs que utilizan SSI, puede haber múltiples SLRs apiladas o dispuestas en capas, cada una de las cuales puede operar de manera parcialmente independiente o en concierto con otras SLRs en el dispositivo. La SSI no es una técnica que se pueda aplicar en el dispositivo con el que se trabaja en este proyecto, por tanto, por indicación del fabricante en la documentación, se ajustará este campo de 2 bits con 00.

- LLLLLLLLLLLLLLLLL: Estos 17 bits indican la dirección de marco a la que se desea acceder. Puede haber un máximo de 2^{17} marcos direccionables.
- WWWWWWW: Estos 7 bits permiten localizar una palabra de 32 bits dentro del marco especificado. Con 7 bits se pueden direccionar como máximo 2^7 palabras, es decir, 128 palabras de 32 bits cada una.
- BBBBB: Estos 5 bits permiten localizar 1 bit dentro de los 2^5 bits, es decir, 32 bits que son el número de bits direccionables dentro de una palabra.

La memoria de configuración de la FPGA de la serie 7 está organizada en marcos que están dispuestos en mosaico alrededor del dispositivo. Estos marcos son los segmentos más pequeños direccionables del espacio de memoria de configuración de la FPGA de la serie 7, y todas las operaciones deben, por lo tanto, actuar sobre marcos de configuración completos [42]. Cada marco consta de 101 palabras de 32 bits, por tanto, el campo WWWWWWW operará hasta la palabra número 101 dentro de un marco, una vez superado este número (102, 103, etc) no habrá efecto alguno sobre el dispositivo.

Las direcciones de marco, a partir de ahora (LFA – *Linear Frame Address*), comienzan su numeración desde 0, es decir, la primera dirección de marco LFA es 0, la siguiente, LFA=1, y así sucesivamente. La LFA=0 corresponde con una ***pad frame***, o marco de datos de relleno o de separación antes de los datos relevantes que se procesan. El propósito del pad frame es simplemente actuar como una barrera o marcador entre el inicio del flujo de datos y los datos de configuración que siguen. El resto de las direcciones LFA, que comienzan desde LFA=1, forman parte del flujo de datos de la configuración.

Obviamente, no todos los marcos (LFAs) tendrán un '1' como indicativo de que ese marco en concreto alberga un bit esencial. Por tanto, se les llama active frames o marcos activos a aquellos marcos que contengan al menos un bit esencial. Debido al tamaño exageradamente grande del fichero EBD, que, en el caso del diseño realizado, se han obtenido 403.907 líneas, teniendo en cuenta las 8 primeras líneas de cabecera, se ha tornado necesario el buscar un método para automatizar y agilizar la búsqueda de active frames. El método de automatización ha sido la creación propia de un script de Python (lfa.py) que se adjunta en el ANEXO B, que recibe como entrada el fichero EBD y genera otro fichero de salida (inject_address.h) que será el fichero que

contendrá todas aquellas direcciones de marco LFA que serán activas, es decir, que contengan al menos un bit esencial.

Con esto, ya se tienen todas las LFAs que se pueden emplear como potenciales direcciones de Memoria de Configuración sensibles a SEUs, para realizar la inyección de errores. Esta forma de proceder agiliza el que un error altere directamente un bit esencial, error que podrá ser corregido o no por el Controlador SEM IP. Esto es revelador, puesto que, si se inyectan errores en marcos no activos, es decir, carentes de bits esenciales, el Controlador no procederá a su identificación y corrección. Esto no supone que los bits no esenciales no desempeñen funciones importantes y que su corrupción no pueda tener efectos secundarios no deseados, aunque no sean inmediatamente perjudiciales.

Si, por ejemplo, se quisiera inyectar un error en el bit número 3 de la palabra número 2 de la dirección de marco lineal 1, el comando sería como se muestra a continuación (ver figura 45):

1 1 0 0	0 0 0 0	0 S S L	L L L L	L L L L	L L L L	L L L L	W W W W	W W W B	B B B B
1 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0	0 0 1 0	0 0 1 1

Figura 45. Representación de la estructura de las Direcciones de Marco Lineales

A modo de representación, se muestra en la figura 46 un diagrama que resume la estructura del fichero EBD para abordar la inyección de errores en direcciones específicas de la Memoria de Configuración que previamente se delinearon.

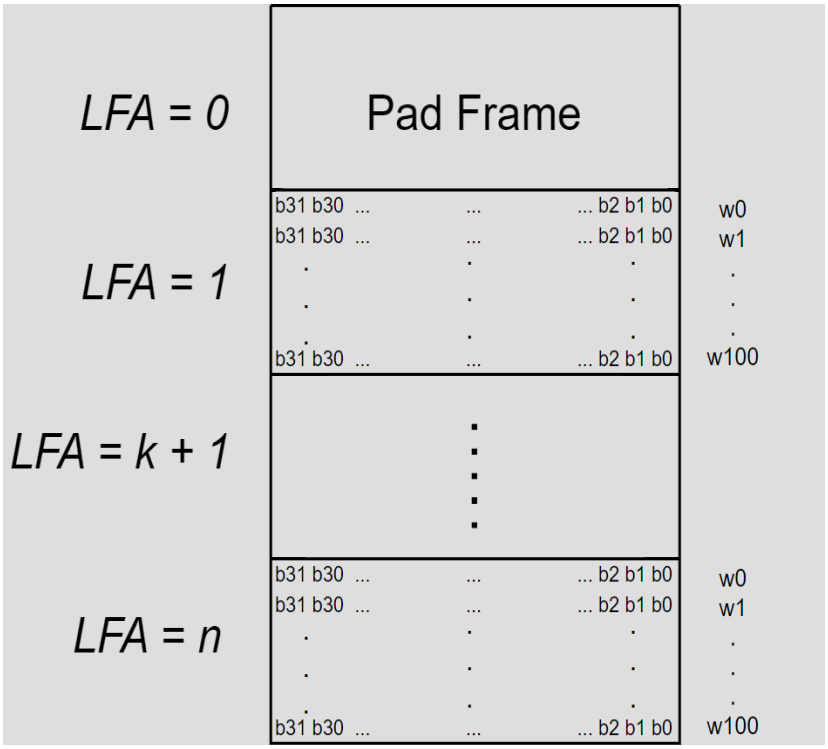


Figura 46. Mapa de las Direcciones de Marco Lineales

- Direccionamiento Físico de Marcos (Physical Frame Addressing): Proporciona una forma de dirigirse a los marcos que refleja la disposición física y el tipo de recursos dentro del FPGA. Este método es más complejo, pero es esencial cuando es necesario comprender la localización física de los marcos para operaciones específicas, como la optimización del rendimiento o la mitigación de errores. No se darán más detalles específicos de este método debido a que queda fuera del ámbito de estudio de este proyecto, pero sí la estructura que debe tener el comando para la inyección de error por este método como se muestra en la figura 47.

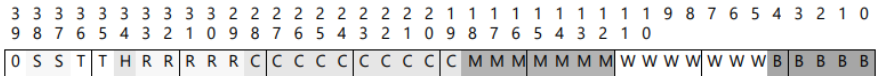


Figura 47. Comando para Inyección de Error mediante Direccionamiento Físico de Marcos

Siendo cada uno de los campos:

- SS (2 bits): Idem (véase la explicación anterior en la sección de Direccionamiento Lineal de Marcos).
- TT (2 bits): Block Type (Tipo de Bloque).
- H (1 bits): Half Address (Media Dirección).
- RRRRR (5 bits): Row Address (Dirección de Fila).
- CCCCCCCCCC (10 bits): Column Address (Dirección de Columna).
- MMMMMMM (7 bits): Minor Address (Dirección Menor)
- WWWWWWW (7 bits): Idem (véase la explicación anterior en la sección de Direccionamiento Lineal de Marcos).
- BBBBB (5 bits): Idem (véase la explicación anterior en la sección de Direccionamiento Lineal de Marcos).

4.7.5 Exportación del Hardware y Código de Aplicación en Vitis IDE

Una vez generado el Bitstream, se obtiene un componente crucial para la manipulación del hardware previamente diseñado. La descripción de este hardware reside en un archivo denominado XSA (Xilinx Shell Archive), el cual es producido dentro del entorno de Vivado. Este archivo XSA alberga la información sobre la estructura del hardware diseñado, que, al ser implementados en el dispositivo correspondiente, permiten la posterior programación del software de aplicación directamente sobre dicho hardware.

El desarrollo de este código de aplicación se efectúa a través del entorno Vitis IDE, proporcionando las herramientas necesarias como cualquier otro entorno de desarrollo para la programación de sistemas basados en microprocesador.

La operación de exportación se realiza accediendo desde el menú de navegación de Vivado en la opción de ***File > Export > Export Hardware***, seleccionando la opción de ***Include Bitstream*** que garantiza la exportación de no solamente la especificación hardware para las herramientas de software sino también la implementación de hardware completa y el Bitstream correspondiente.

4.7.6 Creación de un Proyecto Nuevo en Vitis IDE

Tras la inicialización del entorno del Vitis IDE, el procedimiento para establecer un nuevo proyecto se articula a través de una serie de pasos meticulosos. Estos pasos son fundamentales para asegurar que la plataforma de hardware previamente generada desde Vivado se integre adecuadamente dentro del Vitis. Esta integración permite la manipulación y el desarrollo directo sobre dicha plataforma. Los pasos son los siguientes:

- Accediendo al menú de navegación File > New >Application Project se abre una ventana que asistirá en la creación del proyecto de aplicación. Avanzando con Next se llega a la ventana de Platform donde se seleccionará la opción Create a new platform from hardware (XSA) desde donde buscaremos la ruta en el cual se haya guardado el archivo XSA.
- Tras indicar el archivo XSA, pasando a la siguiente fase mediante Next, se debe indicar un nombre para el proyecto y seleccionar el tipo de procesador sobre el que se quiere construir el proyecto, en este caso, ps7_cortexa9_0 que hace referencia al microprocesador o Processing System (PS) de la arquitectura ARM Cortex-A9 integrado en el SoC ZYNQ-7000.
- Se continúa mediante Next hasta llegar a la fase llamada Template donde se puede elegir una plantilla de proyectos previamente diseñados que facilitarán el desarrollo del proyecto. En este caso, se elegirá la opción de Empty Project (C), que hace referencia a un proyecto vacío que se programará en el Lenguaje de Programación C. Se presiona en Finish y se despliega el Proyecto Nuevo.

En el ANEXO C se presenta el código desarrollado en lenguaje C, el cual se deriva parcialmente de un prototipo proporcionado por Xilinx para la gestión de los GPIO. Dicho código ha sido no solo adoptado, sino también ampliado y mejorado para atender a las exigencias específicas de este proyecto. Se ha priorizado la elaboración de un código con una trazabilidad meticulosa, lo cual simplifica enormemente el proceso de depuración en los casos en que el programa no ejecute las operaciones previstas de manera satisfactoria.

Con el objetivo de facilitar la comprensión del código y en consecución de una mayor claridad y orden, se ha procedido a encapsular cada una de las instrucciones, agrupándolas por su funcionalidad en funciones independientes.

Aunque no se abordará cada detalle del código, se expondrán los aspectos significativos previamente mencionados que son cruciales para el acceso y la operatividad del Controlador SEM IP. Sin esta exposición detallada, el funcionamiento del Controlador sería inviable.

4.7.7 Código de Aplicación en C y Resultados

Como cualquier código en C, la función principal que empieza a ejecutarse es la función *main()*, en la que se ejecutan una serie de funciones que sirven para inicializar el hardware (*hardware_init()*) que se va a emplear. Se muestra en la figura 48 la inicialización de los puertos

GPIO; la configuración necesaria para que el dispositivo sea reconocido (*device_config_init()*); y la habilitación del puerto ICAP (*XDcfg_EnableICAP(&deviceConfig)*).

```
hardware_init();
device_config_init();
XDcfg_EnableICAP(&deviceConfig);
```

Figura 48. Funciones para la Inicialización del Dispositivo

La función de *hardware_init()* se inicializa previo al acontecimiento de cualquier ejecución que tenga que ver con el uso del Controlador SEM IP. Se inicializan los puertos GPIO que se van a emplear, y posteriormente, es necesario garantizar que la señal icap_grant esté deshabilitada, puesto que, mientras el dispositivo se está inicializando, tienen lugar las actividades relacionadas con el arranque del dispositivo y la descarga del Bitstream en la FPGA y su puesta en marcha, que son denominadas actividades de la PCAP. Mediante la función *disable_icap_grant()* se asegura que no se haya concedido permiso para el funcionamiento del Controlador SEM IP dentro de la FPGA, hasta que las actividades mencionadas no hayan finalizado. Cuando este proceso termina, tanto la PS como la PCAP continúan teniendo la lógica de configuración bajo control, pero esta no es una situación deseable, debido a que, en estas condiciones, no se podría tener control sobre el Controlador SEM IP, por lo que, se procederá a aplicar en código lo que se mencionó en [4.1.1].

Después de que el dispositivo haya sido inicializado, tras ejecutar la función de *device_config_init()* es hora de habilitar el puerto ICAP con la función *XDcfg_EnableICAP(&deviceConfig)*. Especial atención se debe prestar a la figura 49, que contiene las líneas de instrucciones que hacen posible dar el acceso a la ICAP, en acompañamiento de la figura 50, que ya se mostró en el apartado [4.1.1], para dar contexto a los comentarios que en el código se muestra.

```
/* Setting the PCAP_MODE (bit 26) in the Multiplexer below
 * to have Full Access to the PL Config Module */
XDcfg_WriteReg(InstancePtr->Config.BaseAddr,
        XDCFG_CTRL_OFFSET, (RegisterHandler | DEVC_CTRL_PCAP_MODE_MASK));

/* Clearing the PCAP_PR (bit 27) in the Multiplexer above
 * to Grant Access to ICAP Path */
XDcfg_WriteReg(InstancePtr->Config.BaseAddr,
        XDCFG_CTRL_OFFSET, (RegisterHandler & (~DEVC_CTRL_PCAP_PR_MASK)));

/* Now it's time to ENABLE the icap_grant signal
 * through GPIO to be able to interact with
 * the SEM Controller
 */
enable_icap_grant();
```

Figura 49. Código para dar acceso al Controlador SEM IP en la PL

En el código, la variable *DEVC_CTRL_PCAP_MODE_MASK* no es más que una variable global de 32 bits en la que se establecen todos sus bits a '0' salvo el bit 26, que se establece a '1' para que, a través del registro de configuración, el multiplexor demarcado en color amarillo en la figura 50 seleccione la ruta de configuración PCAP, que representando su valor en hexadecimal sería 0x04000000. Posteriormente, en la instrucción siguiente, se escribe en el registro de configuración para establecer el bit 27 a '0' (0x08000000) que corresponde a la

línea de conexión PCAP_PR en la figura 50, que hará que el multiplexor demarcado en color rojo seleccione la ruta de configuración ICAP que permite a la PL tener acceso a la lógica de configuración (*Configuration Logic*), una vez que la línea de conexión de nombre ICAP_EN controlado desde la PL se establezca a '1' y el multiplexor demarcado en color azul conceda el acceso, y ello se logra mediante la última instrucción del código que es la función *enable_icap_grant()*.

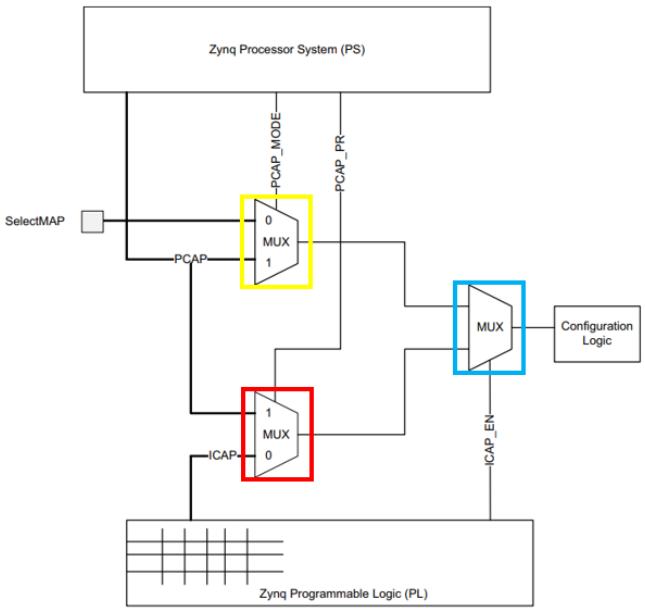


Figura 50. Esquema para la Configuración del Acceso Total a la PL

Las líneas PCAP_MODE y PCAP_PR que entran en los dos multiplexores de la izquierda, representan las señales o bits que se están configurando desde la PS, mientras que la línea ICAP_EN es manejada desde la PL. La línea ICAP resalta la ruta de señalización hacia y desde el bloque de lógica programable PL, ya que el código tiene la finalidad de habilitar y permitir acceso a esta ruta de configuración específica.

No hay que dejar de lado la interfaz ***SelectMAP***, que, aunque no se hace uso de ella, permitiría la configuración de la FPGA utilizando flujo de datos en paralelo, generalmente proveniente de un dispositivo externo como un microprocesador o un controlador de memoria.

El último paso previo para comenzar a inyectar fallos desde el Controlador SEM IP es tener las direcciones donde se quieren inyectar los fallos. Teniendo en cuenta que solamente se usarán direcciones de marcos activos, es decir, con al menos un bit esencial, el script desarrollado en Python automatiza este proceso para la obtención dichas direcciones y ordenándolas en orden en el fichero *inject_address.h* que, tras ser generada, se sitúa directamente en la ruta especificada en el script, que, como es de prever, será la misma ruta del código fuente del proyecto de Vitis. Se muestra en la figura 51 la consola de comandos de **MinGW64** (*Minimalist GNU for Windows 64 bits*), herramienta que permite compilar aplicaciones para el Sistema Operativo de Windows usando herramientas que son comunes en ambientes tipo Unix, como Linux, entre otros.

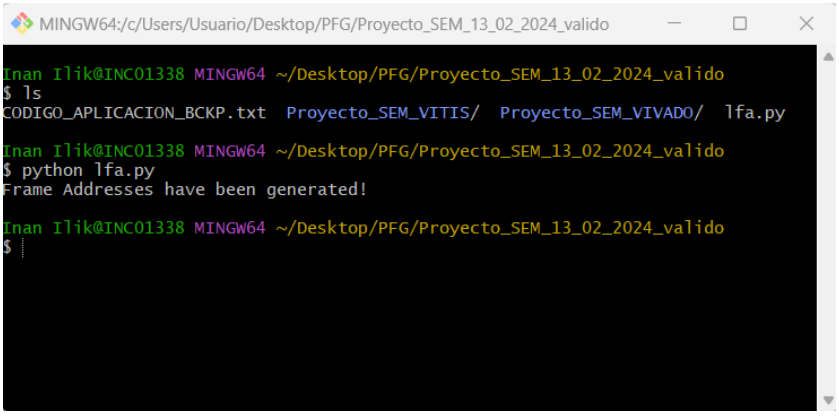


Figura 51. Ventana de la Consola de Comandos de MinGW64

Con el comando **ls** (*List* – Listar), se lista el contenido del directorio actual, que, como se puede observar, el script de Python lfa.py se encuentra en la ruta especificada en color verde. La ejecución del script se lleva a cabo con una simple instrucción anteponiendo la palabra "**python**" seguido del nombre del script con extensión (**.py**). El script genera un mensaje de indicación que el fichero con todas las direcciones de marco LFA se han generado y se encuentra en la ubicación especificada dentro del script de Python.

Por último, cabe exponer la parte del código que toma cada una de las direcciones del fichero ***inject_address.h*** de forma aleatoria, que se produce en una inyección de error en posiciones aleatorias de la Memoria de Configuración.

En los niveles más profundos del mapeo de bits, específicamente en las direcciones de los marcos LFA de la Memoria de Configuración, Xilinx clasifica esta información como confidencial. En consecuencia, no se proporcionan detalles ni herramientas que permitan la identificación precisa de cada celda de configuración. Esto ha resultado en que, a pesar de intensas investigaciones para determinar la localización exacta del componente **counter_error**, no se ha podido acceder a su ubicación de marco específica para realizar inyecciones de errores directamente sobre este. La intención era observar cómo las alteraciones en counter_error afectarían la operación de un LED controlado por la temporización de un segundo que gestiona dicho componente.

Desde la experiencia práctica con la inyección de errores, no se ha encontrado valor en intentar identificar bits específicos. Además, el motivo de la inyección aleatoria de errores es útil desde el punto de vista de que este método emula la característica de los SEUs.

Se tiene un total de 316.116 direcciones posibles donde se pueden inyectar los errores. Estas direcciones pueden estar o no dentro de un mismo marco y pueden ser bits adyacentes o no. Se ha implementado una función llamada *generate_ERROR_INJECTION_ADDRESS()* que de todas las posibles direcciones, escoge una de forma aleatoria comprendido en ese rango. En la figura 52 se muestran las primeras posibles direcciones en el fichero de *inject_address.h*. Debido a la extensión de este fichero, no se añade al apartado de los Anexos.

```
#ifndef __INJECT_ADDRESS_H
#define __INJECT_ADDRESS_H
const unsigned int LFA[] = {
    0x00000068,
    0x0000006B,
    0x00000078,
    0x00000055,
    0x000000D6,
    0x000000D9,
    0x000000DD,
    0x000000EE,
```

Figura 52. Primeras Posibles Direcciones para Inyectar Errores

La función *generate_ERROR_INJECTION_ADDRESS()* ocupa un sitio de preeminencia. Esta función selecciona direcciones de entre un conjunto predefinido, previamente generado y almacenado en el fichero *inject_address.h*. Las direcciones se eligen siguiendo un esquema escalonado, específicamente a intervalos de diez posiciones. Esto implica que, si se elige inicialmente la dirección en la posición número 10, la siguiente elección será la dirección en la posición 20, y así sucesivamente.

Este procedimiento se implementa de esta manera debido a las peculiaridades del Controlador SEM IP, que se ha configurado para corregir desviaciones individuales en bits esenciales—los llamados *Single Bit Upsets* (SBUs)—en lugar de corregir múltiples desviaciones simultáneas, conocidas como *Multi Bit Upsets* (MBUs). La selección de direcciones separadas por al menos dos posiciones asegura que los errores inducidos afecten a bits no adyacentes, garantizando así la eficacia del proceso de corrección.

La razón de adoptar este método de selección específico surge de la necesidad de obviar la función ***rand()*** de las librerías de C, la cual en el entorno Vitis ha mostrado fallos inusuales que en otras plataformas no mostraba. Es crucial destacar que, aunque la elección de direcciones sigue un patrón escalonado, esto no implica que las direcciones de bits esenciales se encuentren ordenadas o distribuidas de manera uniforme. La estructura del archivo *inject_address.h* puede, de hecho, presentar una distribución donde un bit esencial aparezca en un marco determinado y luego no se encuentren bits esenciales en varias decenas de marcos subsiguientes. Esta característica subyacente de la distribución de bits esenciales conserva un elemento de aleatoriedad, a pesar del método de selección estructurado. Por lo tanto, si bien se reduce la dependencia de la función *rand()* y sus potenciales fallas, el carácter intrínsecamente aleatorio del contexto en el que se inyectan los errores sigue siendo una variable relevante en la evaluación de la efectividad del proceso.

La función mencionada es invocada por otra función superior, *PERFORM_ERROR_INJECTION()*, que tiene la responsabilidad de transmitir esta dirección al Controlador SEM IP. Según el diagrama de bloques de diseño observado en Vivado (véase figura 34), la transmisión se realiza a través de dos canales de GPIO de forma separada que luego se concatenan en una única trama de bits que se recibe desde el Controlador como entrada. Es notable que cada canal está limitado a manejar un máximo de 32 bits, aunque el comando completo abarca 40 bits.

De estos 40 bits, los 9 bits de mayor peso son fijos tal y como se indica en la hoja técnica del SEM IP (véase figura 44) y por ende se declaran como constantes en el código. Esto deja los 31 bits de menor peso, los cuales son configurables y son utilizados para almacenar la dirección específica donde se inyectará el error.

Como ya se había anticipado en el apartado [4.1.1] en la figura 33, el comportamiento interno del Controlador SEM IP se puede modelar mediante un autómata de estados. Dicho autómata que consta de 7 posibles estados, se ha reflejado en el código de software dentro de la función principal *main()*. Los únicos estados con los que se debe interactuar para indicar el comienzo de la inyección de fallos son:

- IDLE: Desde este estado, mediante la función *PERFORM_ERROR_INJECTION()*, se ordena, además de la dirección del bit esencial donde se desea inyectar el fallo, la habilitación de la señal ***inject_strobe*** que, hasta que se le asigne un estado lógico alto, es decir, un '1', no se llevará a cabo la inyección. A continuación, tras activar *inject_strobe*, el Controlador accede internamente al estado de INJECTION de forma automática (no se tiene control sobre este proceso interno de transición, sino únicamente el conocimiento del estado actual en el que se encuentra). Tras finalizar la inyección en el estado anterior, se realiza un retorno automático al estado de IDLE. Para obtener información sobre los resultados de la inyección, se ordena (aquí el usuario es el que tiene el control) la transición al estado de OBSERVATION desde donde se comprueba si es necesario realizar una corrección.
- OBSERVATION: En este estado, al haber configurado el Controlador SEM IP para realizar correcciones por reparo, se cambia al estado de CORRECTION, que simplemente emplea un método algorítmico y si el error es corregible, realiza una reconfiguración parcial activa para reescribir el contenido alterado. Como el error que se intenta corregir debe ser previamente clasificado, se haya habilitado o no está configuración, desde el estado de CORRECTION se pasa al estado de CLASSIFICATION, desde donde se decide si el error encontrado es esencial o no. Para el caso de este proyecto se deshabilitó la clasificación de error, por tanto, todos los errores encontrados se clasificarán como esenciales, independientemente de si son o no corregibles.

  Si el error inyectado se puede corregir, se ejecuta el algoritmo de reparación, se vuelve de nuevo al estado de OBSERVATION para comprobar si hay errores nuevos que corregir. Como el autómata diseñado inyecta errores de forma iterativa en direcciones de bits esenciales, de uno en uno, es imposible que en este estado se observe otro error hasta que no se vuelva a realizar la inyección. Por lo que, para realizar una nueva inyección, se debe pasar al estado de IDLE mediante la función ENTER_IDLE_STATE().

- ESSENTIAL: Este estado no existe como tal dentro del Controlador SEM IP, sino que es una invención propia del diseño en el código software, para determinar que tras no haberse podido clasificar el error en el estado de CLASSIFICATION, se realiza una transición al estado de ESSENTIAL automáticamente, indicativo de que ha habido un

error de clasificación (como era de prever). Si realmente hubiera un error esencial (clasificado o no) no corregible, la Interfaz de Estado del Controlador SEM IP activaría su señal de salida ***status_uncorrectable***, y habría que volver a configurar o reprogramar la FPGA. Sin embargo, si resulta que la clasificación de errores siempre determina que los errores son esenciales, activándose la señal de salida ***status_essential*** del Controlador, entonces, es posible continuar inyectando errores en el modo de funcionamiento normal del autómata sin que haga falta reiniciar o reconfigurar la FPGA. Al estado de ESSENTIAL se llega si el autómata se encuentra con un error esencial (al no clasificarse, todos son esenciales), que al tratarse de un soft error, no se saldrá de este estado hasta que no se aplique en el Controlador el comando de *Software Reset* (Reinicio del Software), como se indica en la hoja técnica del Controlador SEM IP. Por tanto, se sale de este estado aplicando tal comando y ejecutando la función de *ENTER_IDLE_STATE* para seguir inyectando errores. El comando de *Software Reset* no tendría sentido aplicarlo si hubiera un error esencial no corregible, que de ser este el caso, el comando no tendría efecto.

- UNCORRECTABLE: Este no es un estado inherente al diseño interno del Controlador, sino una adición del código software. Este estado se alcanza cuando, a través de la interfaz de estado del Controlador, se activa la señal status_uncorrectable. Dicha señal indica que la dirección del bit esencial donde se inyectó el error no puede ser corregida, ya sea porque el error es de naturaleza incorregible o porque la capacidad de corrección ha sido superada.

  Antes de abandonar el estado de CORRECTION, el Controlador activa la señal anterior, reflejando así la irrecuperabilidad del bit afectado. Una vez activada, esta señal indica un bloqueo en el autómata, impidiendo la corrección de errores adicionales y, por consiguiente, el sistema queda inoperante hasta que se realice una intervención decisiva.

  La única alternativa para restablecer la funcionalidad del controlador bajo estas circunstancias es el reinicio de la FPGA. Este proceso de reinicio, en el marco de este proyecto, involucra reprogramar la FPGA desde el entorno de Vitis, normalmente operado desde un ordenador personal. Sin embargo, en aplicaciones más integradas, como en un vehículo espacial, la falta de herramientas adecuadas a bordo para reconfigurar la FPGA (por ejemplo, la ausencia de una memoria con la información necesaria del diseño del circuito y la plataforma hardware) puede resultar en que no sea posible realizar tal reconfiguración, destacando así una vulnerabilidad crítica en el diseño y la operatividad del sistema en contextos de misión crítica.

El código software desarrollado comenzará a realizar las inyecciones de forma escalonada, como se ha mencionado previamente, y hará un recuento del total de errores inyectados. El curso normal del funcionamiento del Controlador será inyectar un error y corregirlo, estado que seguirá hasta cubrir todas las direcciones de bits que con el método implementado se pueda cubrir. Este flujo se interrumpirá hasta que exista un bit esencial que, tras haber recibido el error, el Controlador no lo haya podido corregir, estado que se indica mediante la

mencionada señal status_uncorrectable¸ conectada al LED correspondiente que quedará permanentemente encendida junto al LED correspondiente al estado de status_essential. Esta situación se interpretará visualmente como la existencia de un error esencial no corregible. Sin embargo, para tener conocimiento de la dirección exacta del bit esencial afectado, nuevamente, el código será la entidad que informará de esta situación, de la dirección de bit afectada, además de informar de la necesidad de reinicializar la FPGA.

A continuación, se muestra en la figura 53 los comandos anteriormente mencionados, además del informe (véase figura 54) que genera el programa software a medida que inyecta los fallos y el estado crítico en el que informa de la necesidad de reconfigurar la FPGA, desde la Terminal integrada en Vitis.

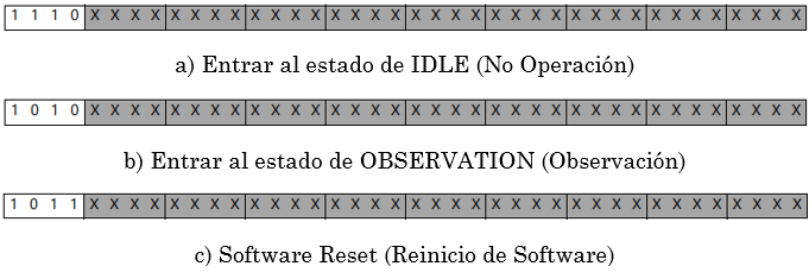


Figura 53. Comandos de la Interfaz de Inyección de Error

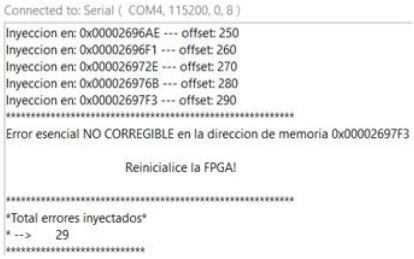
Para mejorar la operatividad y la comprensibilidad del programa encargado de la gestión del Controlador SEM IP, se ha adoptado una estructura modular en la codificación. Cada comando relevante para la transición de estados dentro del autómata se encapsula en funciones distintas. Esta metodología no solo simplifica el proceso de control del programa, permitiendo cambios de estado con una reducción significativa de la repetición de código, sino que también aumenta la legibilidad y la mantenibilidad de este.

La figura 54 muestra el resultado de 3 campañas de inyecciones de errores.

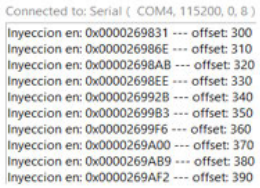
En este contexto, donde la variable offset indica el número tal y como fueron ordenadas cada una de las direcciones de los bits esenciales en el fichero *inject_address.h* y el número a su izquierda es la dirección real del bit esencial dentro de la Memoria de Configuración RAM donde se inyecta el error. Se han realizado más campañas de inyecciones de error siguiendo esta misma línea de experimentación. Se hace constar que, cuando alcanza una dirección de bit esencial que no puede ser corregido y, por tanto, el programa software informa sobre dicha incidencia, que para seguir teniendo el Controlador SEM IP en funcionamiento, habrá de reinicializar la FPGA que, de no ser así, el Controlador quedará permanentemente en un estado de bloqueo con las señales de *status_essential* y *status_uncorrectable* activas de manera constante, paralelamente observables en sendos LEDs correspondientes a dichas señales. Si no se tuviera una herramienta como la Terminal de Vitis que facilitase mensajes de diagnóstico, siempre son útiles los LEDs a modo de depuración y señalización.

Prestando nuevamente atención a la campaña de inyecciones (véase figura 54), se explicará a continuación su principio de funcionamiento.

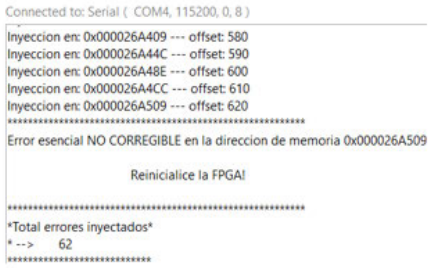
a.1) 10 Primeras Direcciones de Bits Esenciales



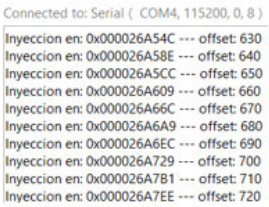
b.1) 5 Últimas Inyecciones Antes de un Error Esencial No Corregible



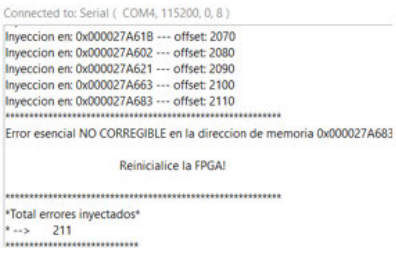
a.2) 10 Primeras Direcciones desde la última Dirección de Bit Esencial con Error No Corregible



b.2) 5 Últimas Inyecciones Antes de otro Error Esencial No Corregible



a.3) 10 Primeras Direcciones desde la última Dirección de Bit Esencial con Error No Corregible



b.3) 5 Últimas Inyecciones Antes de otro Error Esencial No Corregible

Figura 54. Terminal de Vitis para Diagnóstico de la Campaña de Inyección de Errores

En la figura anterior compuesta por varias, las indicadas con letra '**a**' muestran solo las 10 primeras direcciones, por el simple hecho de que no sería adecuado incluir todas las

direcciones donde se han inyectado los errores ocupando un enorme espacio del documento, donde se han inyectado los fallos y consecuentemente, se han corregido, hasta que por un bit esencial no corregible el Controlador se ha bloqueado notificando tal incidencia mediante un mensaje de depuración creado en el propio código para ser mostrado por el Terminal que mediante el protocolo UART (*Universal Asynchronous Receiver Transmitter* – Receptor Transmisor Asíncrono Universal) comunica en serie la Terminal de Vitis del ordenador personal con la placa Blackboard.

Por otro lado, las representaciones con la letra '**b**' muestran las últimas direcciones afectadas, donde todos los fallos, salvo el último, han sido rectificados. Como indica el mensaje emergente, se impone la necesidad de reinicializar la FPGA, procedimiento efectuado en cada instancia de un error esencial no corregible. Al offset que corresponde a la dirección del bit esencial no corregido, se le adicionará 10, configurando así la nueva base para la siguiente dirección de bit esencial en una campaña sucesiva de inyecciones. Esta metodología incrementará de manera proporcional las reconfiguraciones de la FPGA, hasta abarcar la última de las posibles direcciones de bits esenciales. Al finalizar esta secuencia, se habrá realizado inyecciones sobre un décimo de todas las posibles direcciones de bits esenciales.

Como se mencionó anteriormente, las señales *status_essential* y *status_uncorrectable* están conectadas simultáneamente a sendos LEDs como se puede observar en la figura 55 y ambos en estado activo ilustrando la detección de un error esencial no corregible, en este caso particular.

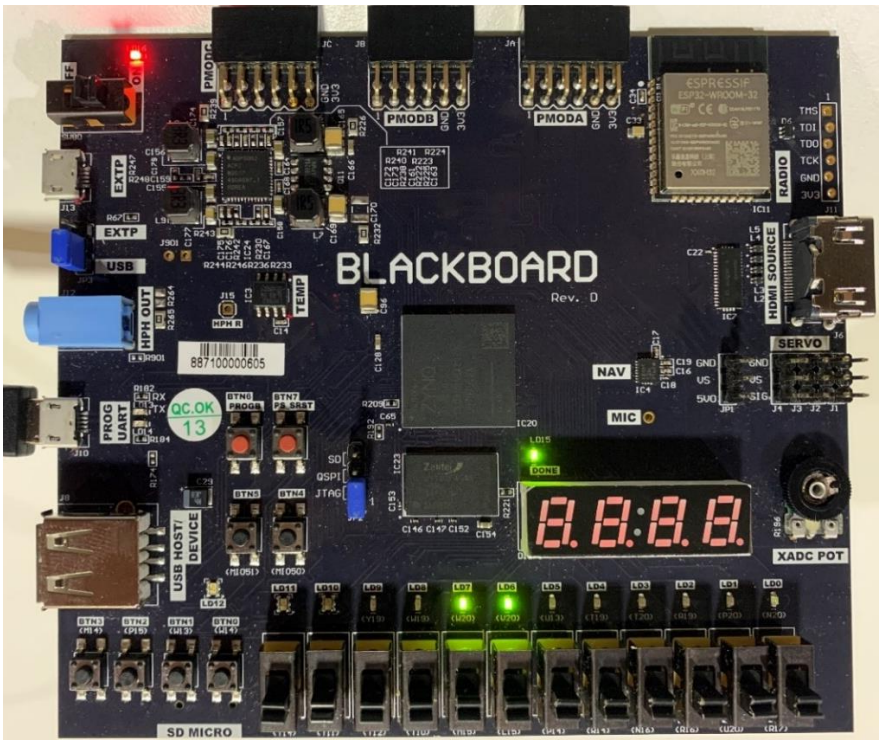


Figura 55. Señales *status_essential* y *status_uncorrectable* (en la parte inferior de derecha a izquierda) activas

El intervalo temporal programado entre cada inyección, establecido en 100 milisegundos mediante una función de temporización, no refleja la irregularidad con la que los SEUs ocurren en condiciones naturales. Sin embargo, esta discrepancia entre la periodicidad artificial y la aleatoriedad de los SEUs en su estado natural es una adaptación pragmática ante la limitación del tiempo como recurso. El propósito de tal diseño es permitir una ejecución de pruebas en un lapso temporal reducido y manejable, facilitando así una evaluación rápida y efectiva de la resistencia del dispositivo bajo condiciones simuladas de radiación. A continuación, se muestra una tabla extraída de la guía del usuario de Xilinx concerniente a la fiabilidad de sus dispositivos que muestra la tasa de errores que puede llegar a ocurrir en el dispositivo Zynq 7000, concretamente, en la Memoria de Configuración RAM (CRAM – *Configuration Random Access Memory*) sometido a radiación real.

Como se puede observar en la Tabla 3 que suministra el fabricante en su documentación técnica sobre la fiabilidad de sus dispositivos SoC [43], que muestra en términos de probabilidad de tasa de fallos en tiempo (FIT – *Failure In Time*) por Megabit, o sea, FIT/Mb y en términos de sección transversal por bit (LANSCE (*Los Alamos Neutron Science Center*) *Neutron Cross-Section per bit*), que es una métrica que indica la probabilidad de que un neutrón interaccione con un bit en un chip de memoria o procesador. Tomando como ejemplo el caso de (**FIT/Mb (*Alpha Particle*)**) se tendría para el SoC Zynq 7000, el utilizado en este proyecto, que la tasa de fallos en tiempo por Megabit debido a partículas alfa es de 50 para la Memoria de Configuración RAM (CRAM), lo cual informa de que se esperan 50 fallos por 1.000.000.000 horas de funcionamiento por cada Megabit de memoria debido a partículas alfa y el error asociado con esta tasa, que indica los valores (-34%;+56%) como la incertidumbre en la medición, es decir, que la tasa de fallos indicada podría sufrir variaciones en los porcentajes mostrados.

Tech Node	Product Family	LANSCE Neutron Cross-Section per Bit[2]		FIT/Mb (Thermal Neutrons)		FIT/Mb (Alpha Particle)[3]		FIT/Mb[4] (Real-Time Soft Error Rate Per Event)[5, 7]	
		CRAM	Error	CRAM	Error[6]	CRAM	Error[6]	CRAM	Error[6]
90 nm	Virtex-4	1.55×10^{-14}	±18%					263	±11%
90 nm	Spartan-3	2.40×10^{-14}	±18%					190	–50% +80%
90 nm	Spartan-3E, Spartan-3A	1.31×10^{-14}	±18%					104	–80% +90%
65 nm	Virtex-5	6.70×10^{-15}	±18%					165	–13% +15%
45 nm	Spartan-6	1.00×10^{-14}	±18%	21	–11% +13%	88	–50% +100%	177	–10% +11%
40 nm	Virtex-6	1.26×10^{-14}	±18%	0.7	–11% +13%	7	–45% +97%	105	–10% +11%
28 nm	Artix 7, Spartan 7, and Zynq 7000	6.99×10^{-15}	±18%	29	–10% +10%	50	–34% +56%	74	–8% +9%
28 nm	Kintex 7 and Virtex 7	5.69×10^{-15}	±18%	1.1	–15% +18%	50	–34% +56%	40	–17% +21%
20 nm	UltraScale	2.55×10^{-15}	±18%	0.5	–13% +16%	9	–64% +374%	32	–10% +12%
16 nm	UltraScale+	2.67×10^{-16}	±18%	0.35	–16% +20%	0.1	–20% +20%	5	–16% +20%
7 nm	Versal	2.6×10^{-17}	±20%	0.1	±50%	0.1	±50%	0.7	±50%

Tabla 3. Pruebas de Haz Experimental y Tasas de Error Suave en Tiempo Real para CRAM [43]

Una perturbación en cualquier bit no engendra necesariamente un error funcional suave de forma aislada. Para que ocurra tal error, el bit afectado debe ser crítico para la función del dispositivo. Los bits esenciales generalmente entran en la categoría de bits críticos. La existencia de bits no utilizados y bits no críticos disminuye la tasa efectiva de errores suaves en función de lo que se denomina el factor de vulnerabilidad del dispositivo (DVF, por sus siglas en inglés). Para un diseño típico, el DVF es del 5%, lo que significa que una de cada veinte perturbaciones provoca un error suave funcional, en promedio. En el peor de los casos, el DVF nunca supera el 10%, es decir, nunca más del 10% de las perturbaciones resultan en un error funcional suave. Por lo tanto, la tasa de errores funcionales suaves de un diseño operando en lógica programable (PL) es considerablemente inferior a la que se predeciría mediante cálculos basados en los datos de tablas específicas. Un factor significativo que contribuye a un DVF bajo es que la mayoría de los recursos de enrutamiento de la lógica programable no se utilizan en una implementación particular [43].

Se pueden recorrer todas las direcciones de bits esenciales siguiendo la lógica de avance secuencial de forma escalonada o recorrer las direcciones de una a una. Estimando que las 3 campañas mostradas llevaron aproximadamente de 5 a 6 minutos entre reconfigurar la FPGA e inyectar los fallos alcanzando el número de inyecciones en direcciones diferentes de 211 que, redondeando serían una media de:

$$\frac{211}{6} = 35 \left[\frac{inyecciones}{minuto}\right]$$

Si hubiera que realizar campañas para cubrir todas las direcciones de los bits esenciales que son un total de 316.108 direcciones, dividiendo esta cantidad por 35, tendríamos un tiempo estimado de:

$$\frac{316.108\ inyecciones}{35\ \left[\frac{inyecciones}{min}\right]} \approx 9.032\ minutos \approx 150{,}5\ horas$$

Este cálculo revela que, bajo las condiciones actuales, el tiempo requerido para realizar pruebas exhaustivas de fiabilidad es considerablemente elevado, especialmente teniendo en cuenta que la reconfiguración de la FPGA se realiza manualmente, lo cual puede resultar en una tarea extremadamente laboriosa. A menos que se introduzca una automatización del proceso, lo cual no está contemplado en el alcance de este proyecto, la viabilidad de llevar a cabo pruebas de esta magnitud se ve significativamente comprometida. Esta situación subraya la importancia de considerar métodos más eficientes o de limitar el alcance de las pruebas a un subconjunto representativo de direcciones, si se desea mantener un equilibrio entre la profundidad de la prueba y la practicidad operativa, ya que el objetivo de este proyecto no es sino el de exponer una metodología de análisis.

Tras los experimentos realizados, se concluye con que el instante o la dirección de bit esencial en el que va a ocurrir un error esencial no corregible resulta ser totalmente aleatorio después

de cada reconfiguración de la FPGA. Aunque no se actualizase la variable *offset* y todas las campañas comenzasen desde el *offset* = 0, el error esencial no corregible, no sería en la misma dirección de bit esencial donde ocurrió la última vez.

En el análisis detallado del experimento realizado mediante el software Vivado, se integró un componente específico denominado counter_error, un contador cuya salida está conectada al LED8 (PIN T10) en la placa Blackboard. Este LED, oscilando a una frecuencia de 1 Hz, sirve como un indicador visual, destinado a manifestar alteraciones en su secuencia de parpadeo que denoten la presencia de errores inducidos en el circuito. Sin embargo, a pesar de haber realizado numerosas campañas de inyección, específicamente cincuenta en total, no se han observado las alteraciones esperadas en el parpadeo del LED. Esto es debido a que, a priori, no se tiene conocimiento de en qué direcciones de bits esenciales podría estar ubicada la lógica de dicho componente porque Xilinx no provee de tal información de mapeo entre la implementación de los recursos y su posición en el Bitstream. Además, aunque el error incidiera en uno de los bits esenciales del componente mencionado, este es corregido inmediatamente, sin dejar la posibilidad de ver dicha alteración.

No obstante, es relevante destacar que la inobservancia de alteraciones en el parpadeo del LED no debe interpretarse como una deficiencia, sino más bien como una confirmación del funcionamiento eficiente del Controlador SEM IP en la corrección rápida de errores esenciales y SEUs. Se debe recalcar que estas pruebas se han llevado a cabo en un entorno simulado y no bajo condiciones reales de radiación.

Los errores esenciales alteran algún recurso, a saber, flip-flop, CLBs, entre otros, de los utilizados en la FPGA, pero mientras por azar no se trate de un recurso del contador counter_error, dicho contador podrá seguir operando al no existir una reconfiguración.

La omisión de mensajes de depuración en la Terminal, que indicarían el estado en que se encuentra el Controlador en cada momento, obedece a la influencia perturbadora que una instrucción de impresión puede tener sobre el funcionamiento interno del Controlador, debido a las temporizaciones inherentes a este sistema. No obstante, el estado del Controlador puede ser inferido indirectamente a través de los indicadores LED.

A pesar de la falta de comunicación directa a través de la Terminal, se comprende que, para efectuar una inyección de error, el Controlador debe transitar inevitablemente por varios estados: IDLE, INJECTION, OBSERVATION, y ESSENTIAL. Posteriormente, en casos de detección de errores, seguiría a los estados de CORRECTION y CLASSIFICATION; sin embargo, en esta serie de pruebas, el Controlador ha sido configurado para omitir la clasificación de los errores, tratando todos los errores detectados como esenciales. Además, el estado de UNCORRECTABLE es explícitamente visible desde la Terminal, proporcionando una indicación clara cuando un error no puede ser corregido.

Cabe mencionar que el sistema no ha experimentado el estado de FATAL_ERROR, lo cual requeriría una inconsistencia interna significativa. Es plausible que tal estado podría manifestarse en entornos más desafiantes, como condiciones espaciales, donde la exposición

a radiación y otros factores ambientales extremos podrían inducir fallos que no se presentan en un entorno controlado.

En circunstancias reales de operación en que un vehículo tripulado tuviera implementada esta tecnología, sería poco eficiente tener al dispositivo el cual implementa al Controlador SEM IP conectado a un ordenador para observar los mensajes de consola, por tanto, el uso de los LEDs a modo de notificación cobra relevancia. Se muestra en la ya mostrada anteriormente (véase nuevamente figura 55) cómo los LEDs conectados a las salidas del Controlador SEM IP status_essential y status_uncorrectable permanecen encendidos (los dos LEDs encendidos en la parte inferior, de derecha a izquierda). Otros LEDs están presentes para informar de la correcta alimentación de la FPGA (el situado más arriba) y de que la FPGA está programada y ejecutando el circuito hardware, el etiquetado como DONE (en la parte central de la placa).

Para concluir esta fase de experimentación y análisis con el Controlador SEM IP y transitar hacia una evaluación más focalizada y especializada mediante el uso de un inyector de fallos en una Transformada Rápida de Fourier (FFT), es imperativo recapitular los hallazgos esenciales y las lecciones aprendidas de las pruebas precedentes. Durante estas, se ha evidenciado que el Controlador SEM IP posee una capacidad notable para administrar y corregir errores dentro de un contexto simulado. Esta habilidad es crucial para identificar las áreas específicas dentro de la Memoria de Configuración RAM que son susceptibles a SEUs, facilitando así el análisis detallado de qué configuraciones o combinaciones de componentes en el diseño del circuito son más propensas a experimentar SEUs de manera recurrente.

Sin embargo, a pesar de su eficacia en la gestión de errores, el Controlador SEM IP muestra limitaciones en cuanto a proporcionar una retroalimentación visual directa y tangible de los efectos de los errores. Esta limitación se manifiesta en la dificultad de visualizar directamente las alteraciones en señales complejas, tales como una onda sinusoidal que se degrada tras un SEU, mediante los indicadores estándar proporcionados por la Interfaz de Estado del SEM IP. La naturaleza de esta interfaz no permite discernir adecuadamente la forma o el carácter específico de los efectos causados por los errores.

La transición hacia el uso de un inyector de fallos diseñado específicamente para operar con FFT nos permite abordar esta limitación. Al aplicar errores intencionados en el proceso de la FFT, podemos observar directamente la degradación de la señal, lo cual no solo proporciona una validación visual de la incidencia del error, sino que también permite un análisis más intuitivo y tangible de los efectos de los SEUs.

Este contraste subraya la necesidad de integrar herramientas y técnicas adicionales que puedan ofrecer una representación más clara y detallada de cómo los SEUs afectan a las señales y operaciones complejas dentro del sistema. La implementación de un inyector de fallos en la Transformada Rápida de Fourier representa un paso adelante en esta dirección, permitiendo visualizar cómo estos perturban operaciones específicas y críticas como las transformadas de Fourier, esenciales para el procesamiento de señales, muy recurrentes en el ámbito de tecnología aeroespacial.

El aprendizaje obtenido del uso del Controlador SEM IP dota de conocimientos valiosos sobre la gestión y corrección de errores en sistemas embebidos, conocimientos que ahora se pueden aplicar para explorar y diseñar metodologías de pruebas y análisis más sofisticadas. El enfoque propuesto, que se presenta en el subapartado siguiente, que puede servir para testear y evaluar la tolerancia del cualquier circuito expuesto a errores, permite visualizar los efectos de los errores de manera más explícita mediante la alteración visible, en el contexto de este proyecto, de las ondas sinusoidales procesadas. Esta estrategia no solo amplía nuestro arsenal de herramientas de diagnóstico, sino que también enriquece nuestra comprensión de los impactos funcionales de los SEUs en componentes de procesamiento de señales. Y de qué modo se podrían mejorar los diseños de los circuitos para que estos fueran más fiables, estudio que, con previsión a futuro, podría ser una ampliación de este proyecto.

4.8 Inyector de Fallos mediante una Puerta XOR

Este subapartado en el que se expone una segunda metodología para el análisis de los SEUs en un entorno que simularía la operación del dispositivo SoC Zynq-7000 en un entorno espacial, debe su existencia, como se explicó en el subapartado anterior, a la falta de visibilidad y observación de los efectos de los SEUs sobre los circuito implementados en FPGAs que, en el contexto de este proyecto, se optó por que fuera un circuito que implementa la Transformada Rápida de Fourier, debido a su utilidad prevalente en el procesamiento de señales en diversas aplicaciones aeroespaciales, tales como radar, sonar, análisis espectral en astronomía y astrofísica, y procesamiento de imágenes.

La elección de un circuito que implementa la FFT no es arbitraria, sino que está alineada con los requerimientos técnicos y científicos de las misiones espaciales, donde la precisión y la fiabilidad del procesamiento de señales son de vital importancia. La FFT es una operación matemática que permite la descomposición de señales con dominio en el tiempo en componentes en el dominio de la frecuencia. No se entrará en los detalles intrínsecos sobre el algoritmo que se implementa en la FFT, sino que, simplemente se utilizará tal y como es suministrado como Bloque IP dentro del catálogo de IPs de Xilinx en su plataforma Vivado al igual que se hizo en el caso del Controlador SEM IP.

Mención aparte merece el inyector de fallos implementado mediante una puerta XOR (*eXclusive OR* – O Exclusivo) que, como su nombre indica, realiza una operación lógica en la que su salida es verdadera solo si las entradas son exclusivamente diferentes entre sí (véase figura 56).

Figura 56. Símbolo de Puerta Lógica XOR

Dependiendo de las posibles entradas, la función de salida variará. En la Tabla 4, se recogen las salidas de un bit (Y) de una operación XOR para las entradas de un bit (A y B).

A	B	Y = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 4. Operación XOR con 2 Entradas de un bit

Lo exposición anterior sugiere que la operación XOR se realiza bit a bit, es decir, independientemente del número de bits de los que se componen las entradas (2, 3, 4, 5, …) e independientemente del número de entradas.

La operación que implementa la puerta XOR se introduce como otro Bloque IP en el diagrama de bloques del diseño en Vivado. En la figura 57 se muestra el diseño realizado que integra tanto el bloque XOR como los bloques necesarios para la generación de una onda o señal sinusoidal y aplicarle una FFT.

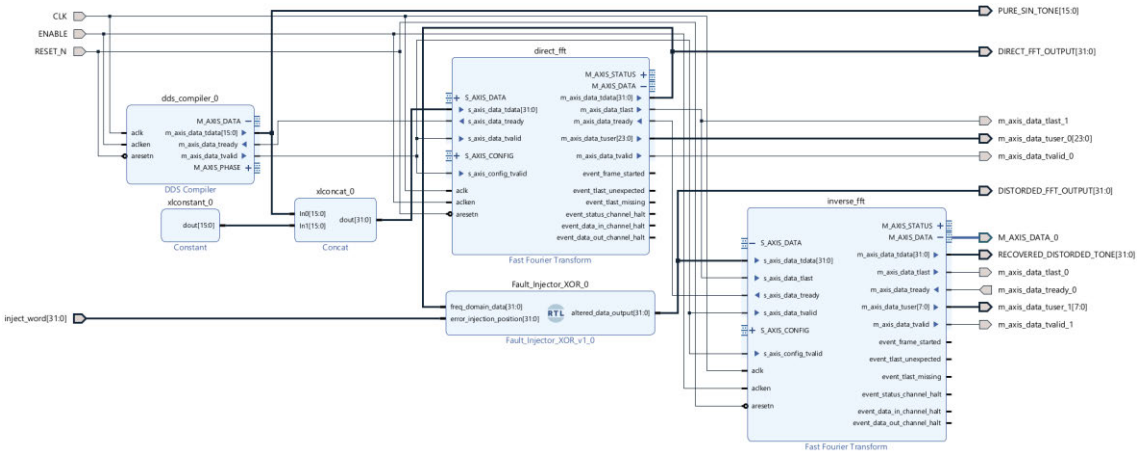


Figura 57. Diagrama de bloques del Diseño de FFT, iFFT e Inyector de Fallos

Es bien sabido que cuando ocurre un SEU, típicamente resulta en un bit flip, es decir, el valor del bit en cuestión se invierte. Este tipo de error es transitorio, lo que implica que el cambio en el estado del bit no altera físicamente el hardware, sino que afecta temporalmente la información almacenada hasta que el bit afectado sea reescrito o el dispositivo se reinicie.

A continuación, se expondrá la función que cumple cada Bloque IP que se ha integrado en el diagrama de bloques completo, todos controlados mediante un reloj de 100 MHz:

- DDS Compiler: Bloque IP de Xilinx entre los suministrados en el catálogo de IPs en Vivado, el DDS (*Direct Digital Synthesizer* – Sintetizador Digital Directo) implementa la generación de la onda sinusoidal al que se aplicará la Transformada Rápida de Fourier o FFT. Un DDS internamente consta de un Generador de Fase y una LUT una función seno o coseno (conversión de fase a sinusoide). La señal que genera a su salida es una onda sinusoidal como la siguiente expresión de su función de onda:

$$Asen(2\pi f_0 t + \theta) \tag{4}$$

  Para facilitar la interpretación de la amplitud A que consta de 16 bits con signo y se expresa típicamente en el rango $[-2^{n-1}, 2^{n-1} - 1]$, o sea, $[-2^{15}, 2^{15} - 1] \rightarrow [-32768, 32767]$, se introduce la normalización para que se ajuste al rango $[-1, +1]$ mediante una función de escalado:

$$A_{norm} = \frac{A}{2^{15}} \tag{5}$$

  Donde:

  - A es la amplitud original
  - 2^{15} es el valor absoluto máximo en el rango de un entero de 16 bits con signo.

Hay dos maneras válidas de expresar esta amplitud, aunque en las gráficas de la simulación se verá el rango total como sigue:

$$-2^{15} \leq A \leq 2^{15} - 1 \quad (6)$$

O, expresado en su forma normalizada:

$$-1 \leq \frac{A}{2^{15}} \leq 1 \quad (7)$$

En cuanto a la frecuencia y fase inicial, estas son de valor:

$$f_0 = 5\,MHz, \theta = 0\,rad$$

La onda sinusoidal generada no posee una naturaleza específica relacionada con aplicaciones de sonido o tensión eléctrica, sino que más bien, se trata de una onda de propósito general con una simple funcionalidad instrumental para el estudio de SEUs y su relevancia se centra en la distorsión que adquiere una vez inyectado el fallo en la FFT.

- Fast Fourier Transform (FFT): Se emplean dos bloques de FFT de entradas y salidas configuradas para aceptar datos de 32 bits, subdivididos entre las partes real e imaginaria, cada una de 16 bits.
    - El primero módulo etiquetado como direct_fft recibirá como entrada la onda sinusoidal pura. Este módulo, al procesar la señal generada por la DDS, que es capaz de suministrar una señal de hasta 16 bits, emplea dichos bits para construir la componente real de la entrada. A esta componente se anexan 16 bits adicionales (todos configurados a cero con el bloque Constant) para formar la componente imaginaria, mediante Bloques IP de concatenación Concat. Por tanto, con una entrada de 32 bits válida, transformará la señal sinusoidal con dominio en el tiempo, en una representación en el dominio de la frecuencia.
    - El segundo módulo aplica exactamente el mismo algoritmo de operación, pero como se puede ver en el diagrama, este bloque recibe a su entrada la señal proveniente del primer bloque de FFT en el dominio de la frecuencia y es exactamente a esta señal a la que se hará incidir un fallo en uno de sus 32 bits que lo componen. El procedimiento es introducir dicha señal en una de las entradas del bloque XOR y su otra entrada se configurará para contener otros 32 bits controlada mediante la señal inject_word que será la encargada de decidir el bit que se va a alterar.

      Tras procesar dicha señal alterada, generará a su salida, no otra señal en el dominio de la frecuencia, sino que se recuperará la señal sinusoidal original en el dominio del tiempo, pero esta vez alterada.

- XOR: El bloque XOR, en su función esencial, lleva a cabo la operación lógica que su denominación sugiere. Esta operación es particularmente distinguida por su capacidad de modificar específicamente un único bit de una señal original de 32 bits en el dominio de la frecuencia, lo cual subraya su utilidad en contextos de manipulación precisa de datos.

  Consideremos, por ejemplo, un caso hipotético en el que la señal original en el dominio de la frecuencia consista en una trama de 4 bits (1010) en lugar de 32. Supongamos

además que existe otra señal de entrada, denominada inject_word, que designa específicamente la posición del bit a ser modificado. Si la intención fuera alterar el segundo bit de menor peso (contando desde la derecha), la trama de inject_word se presentaría como (0010), con un '1' en la posición del bit deseado y los demás bits en cero.

La operación de XOR entre estas dos señales resulta en una nueva trama (1000), donde todos los bits se mantienen invariantes excepto el segundo, que experimenta una transición de '1' a '0'. La singularidad de la puerta XOR yace precisamente en su habilidad para efectuar este tipo de alteración selectiva, conservando el estado de todos los otros bits no señalados para cambio. Esta característica es la razón por la cual se prefirió esta puerta lógica sobre otras alternativas, debido a su precisión y eficacia en la modificación puntual de señales.

En el caso del diseño de la aplicación, este bloque tiene dos entradas (freq_domain_data) y (error_injection_position), esta última controlada mediante un puerto externo mencionado previamente (inject_word) y una salida llamada (altered_data_output) que contiene la señal original en el dominio de la frecuencia, pero con uno de sus bits alterados. Esta señal va a parar en la entrada del bloque FFT que entregará a su salida, una onda sinusoidal en el dominio del tiempo, pero malformado o degradado.

- Concat: Bloques de concatenación que, como su nombre indica, anexa 16 bits (todas sus posiciones con valor cero), que será la componente imaginaria, con los 16 bits de la señal sinusoidal que compondrá la componente real.
- Constant: Bloque que permite manejar valores constantes en el tiempo. Este bloque de 16 bits se aprovecha para formar la componente imaginaria (con valor cero) de la onda sinusoidal.

Una vez expuestas de manera general las funciones individuales de cada bloque y su interrelación dentro del conjunto del sistema, se establecerá una desviación de la metodología elaborada y aplicada en el diseño anterior del apartado [4.1] que incorporaba el uso del Controlador SEM IP. En particular, no se emprenderá la síntesis ni la implementación requeridas para programar la FPGA con el diseño hardware generado. Cabe destacar que, aunque la implementación de este segundo diseño en una FPGA sería completamente factible y operativamente correcta, el objetivo de esta configuración específica trasciende la mera verificación de su funcionalidad en el dispositivo físico.

El propósito primordial que se explora en esta segunda metodología de análisis es generar información visual mediante gráficas sobre cómo un SEU altera una señal sinusoidal (pudiera haber sido otra forma de onda) tras su procesamiento y reconstrucción a través de dos bloques de Transformada Rápida de Fourier (FFT). Se busca, de esta manera, visualizar y comprender la naturaleza de la alteración y su comportamiento. La plataforma Vivado ofrece herramientas de simulación para diseños de hardware antes de su implementación en una FPGA, lo cual constituye un paso esencial para la verificación de la funcionalidad del diseño.

La metodología de análisis que se adopta permite una representación visual clara y directa de las perturbaciones causadas por los SEUs, complementando así el entendimiento más allá del simple funcionamiento técnico de los bloques FFT, aspecto que se relega en este estudio. El análisis se enriquece considerando que el entorno aeroespacial hace uso de diversas técnicas de procesamiento de señales, tales como la Transformada de Ondaletas, Filtrado Digital, Codificación y Decodificación de Error, y Análisis de Telemetría, entre otros. Cualquiera de estas técnicas resulta adecuado para observar cómo se altera una señal bajo la influencia de un SEU y se optó por el empleo del FFT por su simplicidad.

4.8.1 Simulación del diseño en ausencia de SEU

Previo a implementar cualquier diseño hardware mediante VHDL, es esencial llevar a cabo una simulación, etapa crucial para validar el correcto funcionamiento del diseño. Por tanto, un entorno de simulación integrado en la propia plataforma del diseño sería deseable para poder identificar y corregir errores funcionales. Vivado dispone de tal herramienta de simulación para lo cual es necesario que el hardware diseñado sea expuesto a estímulos en sus puertos de entrada, los cuales serán procesados por el diseño y este generará una salida, etapa a la que se prestará atención para validar el funcionamiento. Es necesario para este propósito un banco de pruebas que, de manera secuencial genere los estímulos que permitirán excitar el sistema.

En el contexto de FPGAs y VHDL, se denomina al banco de pruebas como ***test bench*** desarrollado exclusivamente para el diseño particular que, automatiza la ejecución de múltiples pruebas. En el caso de este proyecto se establecen dos pruebas diferentes. Una simulación en ausencia de SEUs para verificar que tanto la señal procesada como la reconstruida son iguales y que no experimenta alteración alguna. Una segunda simulación estará enfocada en inyectar un fallo para modelar un entorno con presencia de SEUs. El *test bench* elaborado se encuentra en el ANEXO D.

El propósito de implementar una primera simulación en ausencia de SEUs es mostrar cómo el hardware procesa la señal suministrada a su entrada mediante los bloques FFT y ofrece a su salida la misma señal sin perturbaciones.

La señal sinusoidal se puede muestrear mediante la FFT escogiendo mayor o menor número de muestras que, en el caso de este diseño, se optó por 1024 muestras, es decir, una resolución suficiente para capturar la integridad y la precisión de la señal durante el proceso de transformación de Fourier. Esta resolución de muestreo se establece para garantizar que el espectro obtenido refleje con exactitud la señal original, minimizando las distorsiones y asegurando una representación fidedigna en el dominio de la frecuencia, aunque a mayor resolución, mayor exactitud.

En la figura 58 se muestra el detalle de la ventana de las gráficas generadas tras la simulación iniciada con el botón **Run Simulation**.

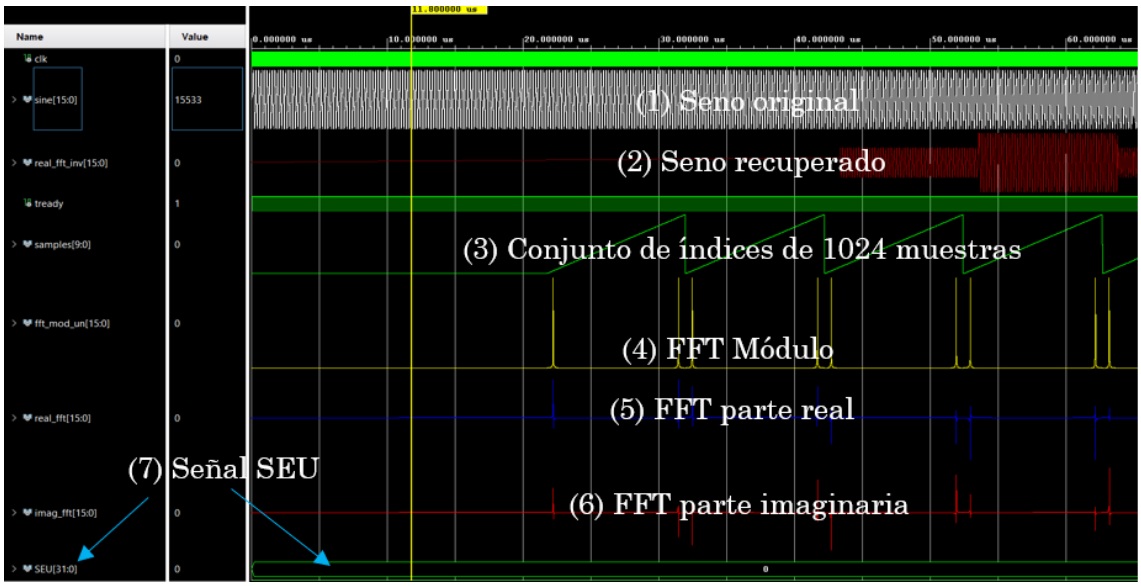


Figura 58. Simulación en ausencia de SEUs

La simulación en su totalidad precisa únicamente del análisis de 4 etapas marcadas por el conjunto de las 1024 muestras, etiquetada como ***samples***, de las que se hace uso para graficar la señal sinusoidal en su dominio en la frecuencia. Cada conjunto de muestras está representado mediante las señales rampa, pero no hay que perder de vista que no es una señal rampa propiamente dicha, sino que es una señal índice, típicamente empleada para transferir información de control o de estado que acompaña del flujo de datos que son las muestras procesadas de la señal sinusoidal por el bloque FFT.

Si se presta atención a la sección previa a la primera etapa, mostrada de manera más ampliada en la figura 59, se puede ver que existe un cierto periodo de latencia que depende de la Frecuencia de Reloj y Rendimiento de Datos con los funciona el bloque que va desde el inicio de la toma de muestras de la señal sinusoidal y procesado hasta que se finaliza el procesamiento hasta que se entrega a la salida el seno original en su dominio de la frecuencia [44].

Figura 59. Etapa de Muestreo y Procesamiento de la señal sinusoidal

Se muestra en la figura 60(a) la primera etapa en la que se grafica la señal seno en su dominio de la frecuencia. Son concretamente las señales etiquetadas como **fft_mod_un**, **real_fft** y **imag_fft**, siendo la primera de ellas: el módulo de la señal de color amarillo; y las otras dos: las componentes real e imaginaria de colores azul y rojo, respectivamente. Es importante prestar a atención a varios detalles para verificar que la FFT funcionó de manera correcta. Un buen indicador es comprobar que, en el dominio de la frecuencia, una señal sinusoidal pura se representa como un pico en la frecuencia del tono fundamental de la señal. Dicho tono fundamental debe ubicarse en la muestra 51 de entre las 1024 muestras totales con la siguiente fórmula:

$$M = \frac{N * f_0}{f_s} \tag{8}$$

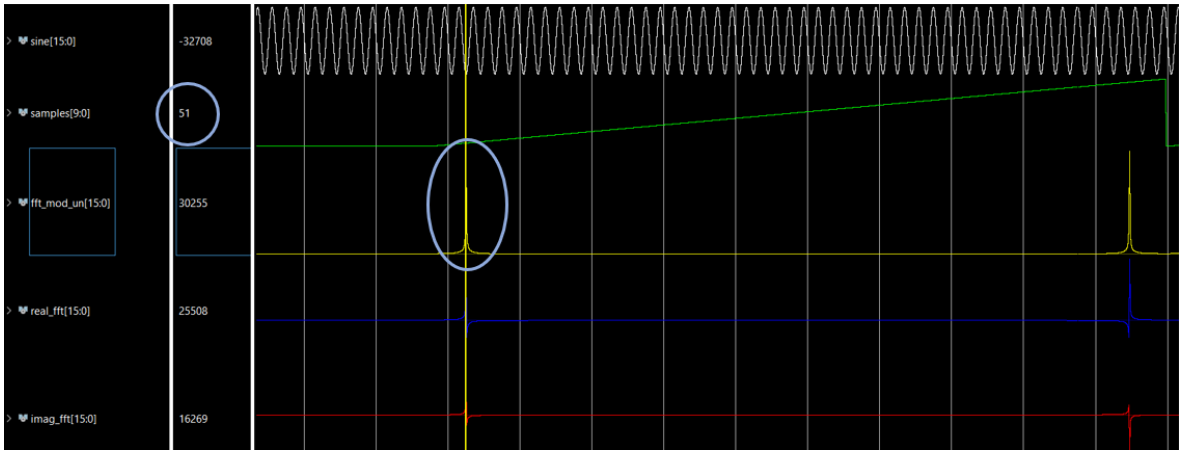
Donde:

- M es la muestra donde se encontrará el pico del tono fundamental.
- N es el número total de muestras que son 1024.
- f_0 es la frecuencia de oscilación de la señal sinusoidal original que es de 5 MHz.
- f_s es la frecuencia de muestreo que son 100 MHz. Por el Teorema de Nyquist, la frecuencia máxima representable es la mitad de la señal de la frecuencia de muestreo máxima, es decir, 50 MHz, por lo que, con 1024 muestras la resolución resultante debe ser aproximadamente de 100 kHz.

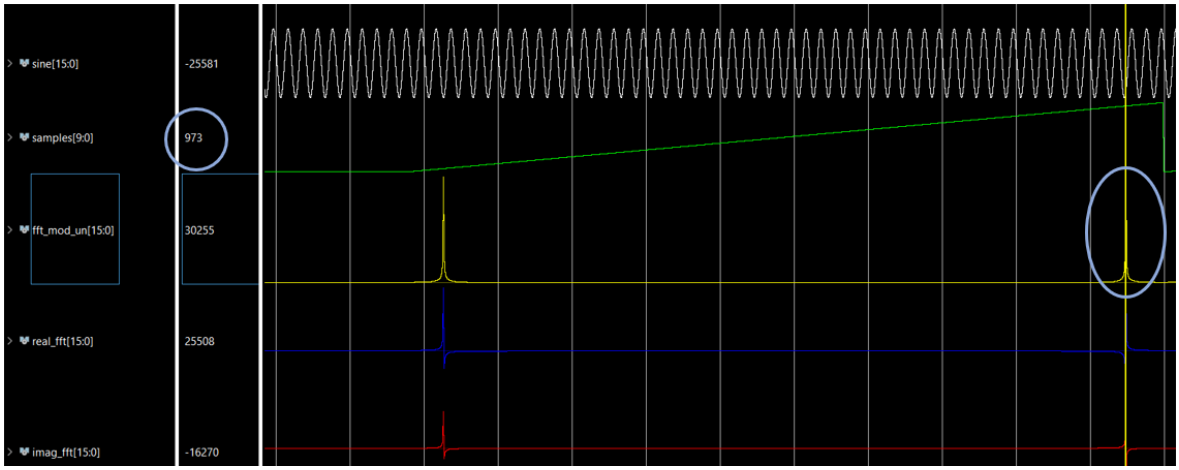
Además, como se puede observar en la figura 60(b), el pico que se ve reflejado en el otro extremo de la gráfica se debe a que la FFT produce un espectro que es simétrico con respecto a la frecuencia cero o fundamental conocido como imagen espectral y que se encuentra concretamente en la muestra:

$$M' = N - M$$
$$M' = 1024 - 51$$
$$M' = 973$$

Una vez comprobado el correcto funcionamiento de la FFT, se puede observar cómo en la segunda etapa se sigue mostrando la misma señal en el dominio de la frecuencia en cuanto a módulo y que seguirá en este sentido indefinidamente porque se está graficando la misma transformada de la misma señal una y otra vez. Sin embargo, la segunda etapa es particularmente importante desde el punto de vista de que marca la etapa durante la cual la señal en el dominio de frecuencia está siendo muestreada y procesada por el segundo bloque de FFT etiquetado como ***inverse_fft*** de cuya salida se obtendrá la señal en el dominio del tiempo, es decir, la señal sinusoidal original que comenzará a graficarse.



a) Pico del Tono Fundamental en la muestra M = 51



b) Pico de la Imagen Espectral del Tono Fundamental en la muestra M = 973

Figura 60. Primera Etapa de las 1024 muestras de la Transformada de la señal sinusoidal

Una vez comprobado el correcto funcionamiento de la FFT, se puede observar cómo en la segunda etapa se sigue mostrando la misma señal en el dominio de la frecuencia en cuanto a módulo y que seguirá en este sentido indefinidamente porque se está graficando la misma transformada de la misma señal una y otra vez. Sin embargo, la segunda etapa es particularmente importante desde el punto de vista de que marca la etapa durante la cual la señal en el dominio de frecuencia está siendo muestreada y procesada por el segundo bloque de FFT etiquetado como ***inverse_fft*** de cuya salida se obtendrá la señal en el dominio del tiempo, es decir, la señal sinusoidal original que comenzará a graficarse.

Al observar la figura 61 se pueden apreciar las 4 etapas, cada una de las cuales con su conjunto de 1024 muestras graficadas.

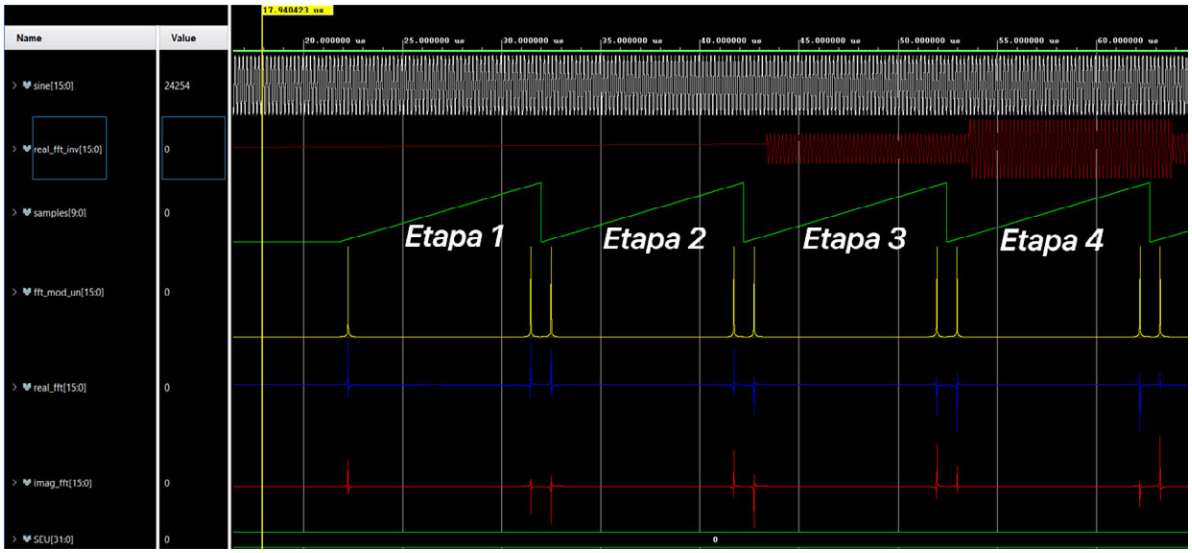


Figura 61. Las gráficas de las 4 Etapas en ausencia de SEU

Requieren especial atención las etapas 3 y 4 debido a que representan los conjuntos de muestras con las que se grafican la señal sinusoidal recuperada etiquetada como ***real_fft_inv***. Estableciendo una ampliación (véase figura 62) a la señal recuperada junto con la señal sinusoidal original de color blanco etiquetado como sine, se puede percibir cómo ambas señales son de la misma frecuencia, aunque desfasadas, además de la latencia inherente de los bloques FFT que atrasan en el tiempo la gráfica, aunque esto no es el análisis que se busca, sino que con esta demostración se pretende enfatizar en que la señal recuperada no se ve distorsionada, salvo en el punto crítico de la transición entre las etapas 3 y 4 que experimenta un aumento en la amplitud debido a que esta varía en función de la fase de la señal con respecto a cada conjunto de 1024 muestras. Sin embargo, cabe hacer énfasis en que se buscan alteraciones del tipo degradación o ruido en la señal recuperada y ciertamente no hay presencia de tales fenómenos.

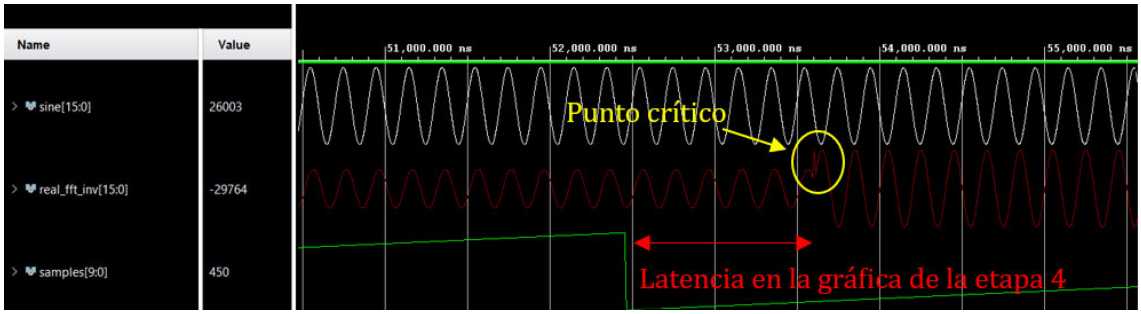


Figura 62. Señal sinusoidal recuperada

Se verá en el subapartado siguiente, de manera más breve, el efecto que genera un error inyectado en los conjuntos de 1024 muestras que contienen la información de la señal sinusoidal en el dominio de la frecuencia, haciéndolo incidir en una de las posiciones (en el bit) de una de sus muestras (cada muestra consta de 32 bits). Se busca de esa manera observar la degradación en la amplitud de la señal recuperada que supone una distorsión o error que afectaría al sistema al que dicha señal fuera a alimentar.

4.8.2 Simulación del diseño en presencia de SEU y Resultado

Esta segunda simulación introduce todos los componentes que se expuso en el subapartado anterior [4.2.1], pero en esta ocasión, se llevará a cabo la inyección de error para observar de nuevo la señal que se recupera en la etapa 4. En la figura 63 se muestra el cambio realizado en el *test bench* para simular tal efecto.

```
133   Inject_SEU : process
134   begin
135   SEU <= "00000000000000000000000000000000";
136   wait for 34 us;
137   report"INJECT SEU------------";
138   SEU <= "00000000000000000100000000000000";
139   --SEU <= "00000000000000000000000000000000";
140   wait for 10 ns;
141   SEU <= "00000000000000000000000000000000";
```

Figura 63. Señal sinusoidal recuperada

La señal ***inject_word*** está internamente conectada a la señal SEU y es a través de esta última con la que se inyecta el error y se produce el *bit flip* o SEU. Cuando el sistema arranca, la señal SEU toma un valor de 32 bits configurados a '0', lo que se debe interpretar como una ausencia de SEUs. Tras realizar una espera de 34 microsegundos, se procede con el cambio del valor de la señal en un único bit, exactamente, el bit número 15 contando desde la derecha. Permanece en este estado durante 10 nanosegundos, es decir, un periodo del ciclo de reloj de frecuencia 100 MHz, y posteriormente, la señal SEU vuelve a su estado original (todos sus bits con valor cero), análogo a la naturaleza de los ***soft errors***. Se deja avanzar la simulación en este estado indefinidamente.

Como se puede apreciar en la Figura 64, la introducción a conciencia de un SEU, es decir, la alteración instantánea (un periodo de reloj) del estado del bit de una muestra del conjunto de las 1024 muestras, produce un tono adicional en el momento exacto de la inyección del fallo, anomalía que se conoce como ***glitch***. Este fenómeno transitorio sería el análogo al que la radiación real produciría en caso de incidir en un bit de dato almacenado en uno de los flip-flop que, extrapolado a nivel físico, se trataría de la alteración del estado lógico en un transistor CMOS.

Este es precisamente el efecto que en la simulación buscaba emular. El fenómeno producido, aunque transitorio, tiene un impacto significativo debido a la naturaleza interna de la FFT. Cada bit en una representación de FFT puede influenciar múltiples componentes de la señal recuperada o reconstruida. Esto es debido a que la FFT manipula representaciones frecuenciales y temporales de la señal, donde cada coeficiente (o bit en su representación) tiene un impacto en todas las partes de la señal a través de la segunda transformada (iFFT) mediante el bloque ***inverse_fft***.

Este es precisamente el efecto que en la simulación buscaba emular. El fenómeno producido, aunque transitorio, tiene un impacto significativo debido a la naturaleza interna de la FFT. Cada bit en una representación de FFT puede influenciar múltiples componentes de la señal recuperada o reconstruida. Esto es debido a que la FFT manipula representaciones

frecuenciales y temporales de la señal, donde cada coeficiente (o bit en su representación) tiene un impacto en todas las partes de la señal a través de la segunda transformada (iFFT) mediante el bloque ***inverse_fft***.

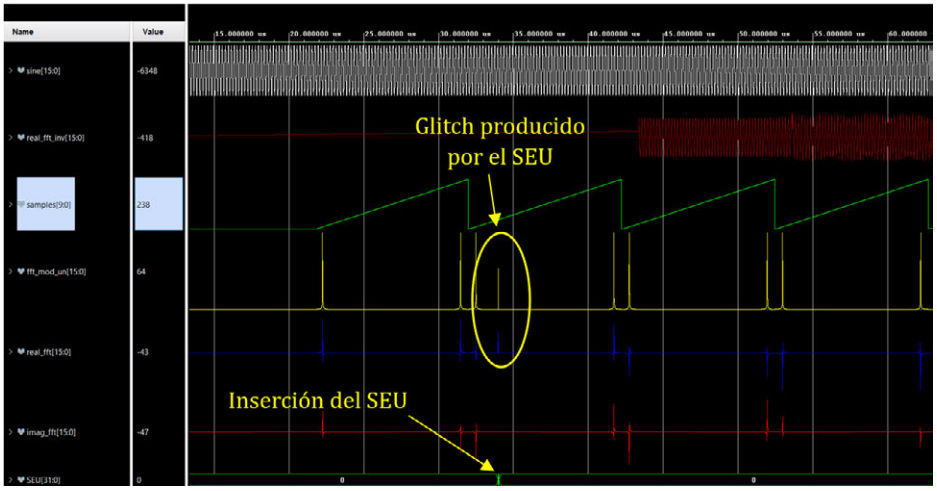


Figura 64. Las gráficas de las 4 Etapas con presencia de SEU

Al alterar un bit en la FFT, dependiendo de cuál sea el bit afectado, se alteran una o más frecuencias que componen la señal. Durante la FFT inversa, estos cambios alterados en el dominio de la frecuencia se traducen nuevamente al dominio del tiempo, manifestándose en la señal recuperada en forma de degradación y comportamiento anómalo y totalmente aleatorio.

La Figura 65 es la representación fidedigna de tales efectos exhibidos en la señal recuperada que, como se puede observar en la transición del primer conjunto de muestras graficadas al segundo conjunto de muestras (la elección del segundo conjunto como víctima del SEU es intencionada, justamente para observar esta transición) la señal reconstruida del segundo conjunto de muestras sigue un patrón que intenta parecerse al seno original pero muy defectuosa y diferente en cada ciclo completo.

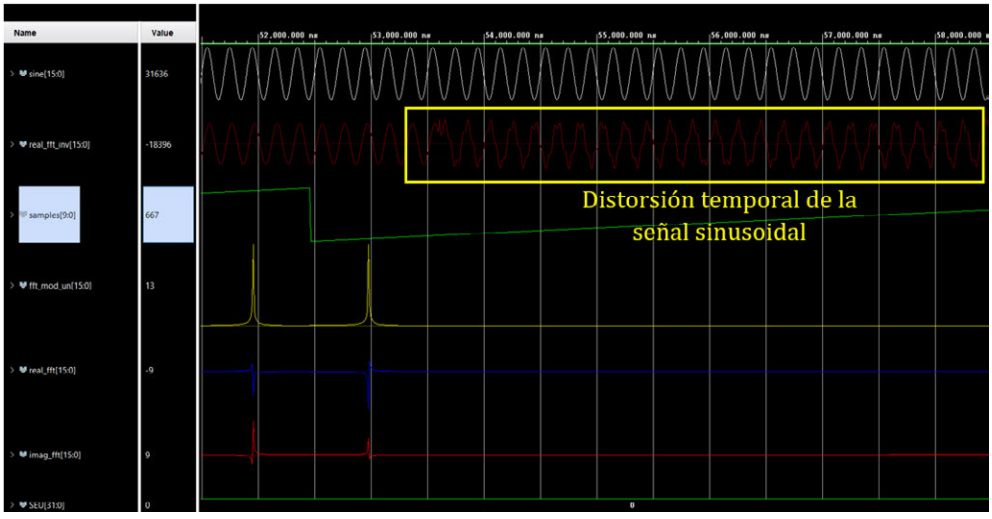


Figura 65. Transición entre las Etapas 3 y 4 y efecto del SEU en la señal sinusoidal recuperada

Esta es la observación o análisis que se convirtió en el motivo necesario que dio lugar a esta segunda metodología de análisis que complementa de manera visual los efectos de los SEUs que el Controlador SEM IP, en su defecto, no era capaz de realimentar el diseñador.

Para comprobar que realmente el efecto del SEU es transitorio, aumentamos el tiempo de la simulación que, como se puede ver en la Figura 66, la señal sinusoidal que se recupera vuelve a su estado original, puesto que un *bit flip* no es un error permanente.

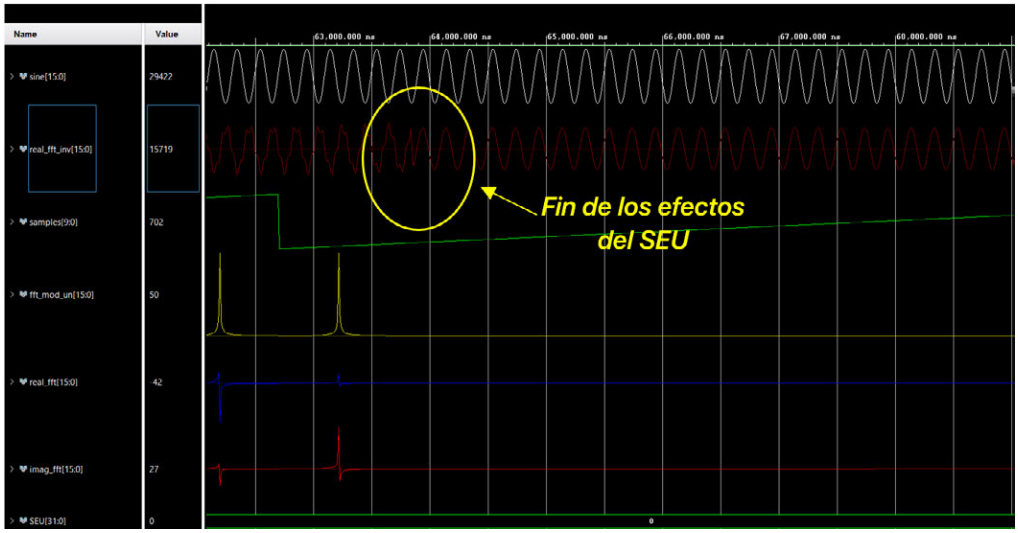


Figura 66. Etapa 5 con la señal sinusoidal libre de los efectos del SEU

Con todo el análisis realizado en este subapartado, se da por finalizada esta segunda metodología llevada a cabo mediante simulaciones. Se hace constar que los bloques FFT y el propio algoritmo matemático de dicha transformada no han sido en ningún momento el objeto de estudio de esta metodología, sino que, más bien han sido herramientas útiles que han contribuido en el entendimiento de uno de los posibles efectos o consecuencias que los SEEs pueden tener en los dispositivos semiconductores, en este caso, los SEU.

5 Presupuesto

A continuación, se expone el coste económico que la realización de este proyecto ha supuesto (ver tabla 5). Para el caso particular del ordenador portátil, el costo total de su uso para el proyecto sería de:

$$Costo\ total = Días\ de\ uso * Costo\ diario$$

Donde:

- $Costo\ diario = \frac{Amortización\ anual}{Días\ laborales\ al\ año} = \frac{117{,}80€}{254\ días} \approx 0{,}46\ [\frac{€}{día}]$
- $Amortización\ anual = \frac{Costo\ del\ ordenador}{Vida\ útil} = \frac{589{,}00€}{5\ años} = 117{,}80\ [\frac{€}{año}]$

El resultado final de usar el ordenador por 160 días sería de:

$$Costo\ total = 160 * 0{,}47 = 73.6€$$

El año 2024 tiene 254 días hábiles y se estiman 5 años de vida útil para el ordenador.

Dispositivos Físicos, Software y Documentación		
Concepto	Descripción	Costo (€)
Material de investigación	Libros de texto; Acceso a publicaciones científicas de IEEEXplore; Guías de usuario y de producto de Xilinx; Esquemáticos y manual de referencia de la placa Blackboard, etc.	0,00
Equipo	Ordenador portátil prestado (HP 250 15.6 pulgadas G9 Notebook PC) Intel Core i5	73,60
	Placa de prototipado Blackboard de RealDigital	137,11
	Programas informáticos Vivado Y Vitis	0,00
Ingeniería	Costes laborales del investigador/estudiante a 1500€ mínimos mensuales por un tiempo de 320 horas dedicadas al proyecto (2h/diarias * 160 días ó 2 meses de jornada completa a 8h/diarias).	3000
Total		3210,71

Tabla 5. Desglose del presupuesto económico

6 Impacto del proyecto

Se desglosan a continuación una serie de implicaciones, que representan el impacto del presente proyecto:

- Implicaciones Sociales: El proyecto aborda dos metodologías que establecen un marco de referencia basado en experimentos simulados de SEEs en SoCs (System on Chip) utilizados en aplicaciones aeroespaciales. La fiabilidad de estos sistemas es crucial en misiones espaciales, cada vez con un mayor porcentaje de uso, por tanto, un error en la operación de un componente puede comprometer la seguridad de la misión y la integridad de los datos de comunicación. Este proyecto provee de métodos de analizar los SEEs y contribuir a aumentar la fiabilidad y disponibilidad de estos sistemas, lo que tiene un impacto positivo en la sociedad al garantizar la seguridad de las misiones espaciales tripuladas y no tripuladas.
- Implicaciones Económicas: La fiabilidad mejorada de los SoCs puede traducirse en ahorros significativos en costos de misión, ya que se reduce la necesidad de mantenimiento y reemplazo de componentes defectuosos. Las misiones espaciales son extremadamente costosas, y cualquier mejora en la fiabilidad de los componentes puede resultar en ahorros considerables. Además, la tecnología desarrollada puede ser aplicada en otros sectores industriales, generando nuevas oportunidades económicas.
- Implicaciones Tecnológicas: Este proyecto avanza en el campo de la tecnología aeroespacial al proporcionar una metodología para analizar y facilitar la concepción de nuevos métodos o ideas para mitigación SEEs en SoCs, aunque estos no se expongan explícitamente. La metodología desarrollada está abierta a mejoras y ampliaciones que sirve de una base sólida para nuevos trabajos futuros.
- Aportación a los ODS (Objetivos de Desarrollo Sostenible): El proyecto contribuye a varios de los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas, incluyendo:
    - ODS 9: Industria, Innovación e Infraestructura: Promueve la innovación y el desarrollo de infraestructuras tecnológicas avanzadas.
    - ODS 12: Producción y Consumo Responsables: Mejora la eficiencia de los recursos y reduce el desperdicio, contribuyendo a una producción más sostenible.

7 Conclusiones

En la investigación para la elaboración de una metodología de análisis de Efectos de Evento Único o Singular (SEEs) en Sistemas en Chip (SoCs) destinados a operaciones aeroespaciales, se llega a una coyuntura donde convergen diversas tecnologías avanzadas. Este proyecto, de índole interdisciplinaria, requiere profundos conocimientos en SoCs, FPGAs, microprocesadores de arquitectura ARM, y entornos de desarrollo integrado como Vivado y Vitis, además del empleo efectivo del dispositivo Blackboard y la programación en Python. Simultáneamente, se profundiza en la teoría subyacente a los SEEs, sus variantes, y sus efectos en dispositivos semiconductores, lo que contribuye en mayor medida la creación de metodologías de análisis pertinentes.

Se opta por la adopción de dos metodologías: la implementación del Controlador SEM IP proporcionado por Xilinx y la simulación de SEUs mediante inyecciones de fallos con la puerta lógica XOR. Se logra una aproximación detallada al comportamiento de los sistemas electrónicos digitales frente a la radiación ionizante, sin embargo, se reitera que ninguna de las pruebas llevadas a cabo se somete a radiación real, sino que, se establece una aproximación a nivel de simulaciones mediante software, aunque sería totalmente factible someter el dispositivo a radiación, puesto que las operaciones internas pueden seguir siendo las mismas independientemente de si el entorno es simulado o real.

Mediante la primera metodología, se observa que inyecciones iteradas de fallos en distintos puntos de la Memoria de Configuración RAM de la FPGA pueden bloquear el Controlador al alterar un solo bit. Aunque esto no afecta a la circuitería interna, como se comprueba con el parpadeo de un LED controlado por la FPGA, sí impide que el Controlador gestione futuros SEUs. Dado el tamaño extenso del fichero de bits esenciales, solo se trabaja con las primeras direcciones para validar la metodología y su potencial de análisis exhaustivo. Se propone, en cambio, para futuros trabajos, recorrer todas las direcciones de la Memoria de Configuración para identificar los bits críticos y aplicar técnicas de mitigación como redundancia, chequeo cíclico de errores o aislamiento de circuitos sensibles.

La segunda metodología surge para complementar la primera, ya que esta última carece de una retroalimentación visual que permita observar cómo se degradan las señales cuando ocurre un SEU. Si bien, los LEDs serven para indicar en qué estado se encuentra el Controlador en cada instante y que alerta de la ocurrencia de errores esenciales, corregibles o no, carece de la efectividad y suficiencia que una gráfica de formas de onda puede exponer. Para ello, se diseña un circuito FFT que recibe una señal sinusoidal a su entrada que, tras ser procesada, se inyecta un error en dicha señal procesada en el dominio de la frecuencia. El efecto del SEU que pudiera ocasionar en la señal no es visible hasta que no se recupera de nuevo mediante una FFT inversa. Después de recuperarla, se puede comprobar una señal muy degradada en sus amplitudes siendo diferentes en cada ciclo, aunque la frecuencia se conserve. Esta segunda metodología se centra en el Inyector de Fallos implementado mediante una puerta XOR como componente generador de errores y, las simulaciones posteriores sientan las bases

sólidas para el análisis detallado de los efectos de los SEUs sobre las señales internas de una FPGA a nivel visual.

Finalmente, cabe resaltar que los objetivos propuestos para este proyecto se lograron mediante la combinación o, mejor dicho, complementación de dos metodologías: una, a nivel de Memoria de Configuración RAM y; la otra, a nivel visual.

7.1 Trabajos futuros

A partir de los logros alcanzados en este estudio y con el fin de extender nuestra comprensión y capacidad para gestionar los Efectos de Eventos Singulares (SEEs) en Sistemas en Chip (SoCs), se proponen varios enfoques para trabajos futuros que puedan profundizar y expandir las metodologías desarrolladas hasta ahora.

Estos enfoques se enumeran a continuación:

- Desarrollo de un Programa con Interfaz Gráfica para Visualización de Formas de Onda: Se propone el diseño de una aplicación con interfaz gráfica que permita visualizar formas de onda, para comparar formas de ondas libres de ruidos o errores y aquellas que han sido alteradas por fallos post-FFT o cualquier otro circuito implementado. Este programa debería ser capaz de comunicarse con la FPGA mediante UART, facilitando así la transmisión de datos en tiempo real y permitiendo una observación detallada y dinámica de los efectos de los SEUs en las señales procesadas mientras la FPGA está en funcionamiento. La interfaz gráfica proporcionaría herramientas para manipular y analizar visualmente las diferencias y anomalías resultantes de las inyecciones de fallos, contribuyendo a una mejor comprensión y respuesta a estos eventos adversos, lo que permitiría observar la degradación de las señales por la reacción de la FPGA, lo que con una simulación se puede, de alguna manera, anticipar, pero no igualar, ya que, se trata de la ejecución de un circuito ya implementado en un dispositivo físico.
- Mapeo de Recursos Internos y Circuitos en la FPGA: Es crucial avanzar en la investigación sobre cómo se mapean los recursos internos y los circuitos implementados dentro de la FPGA en el fichero o archivo de configuración de bits esenciales (EBC). Este estudio debería enfocarse en identificar precisamente qué bits son críticos para la operación del circuito que se vaya a implementar. A través de este mapeo, se podría realizar inyecciones de fallos dirigidas exclusivamente a circuitos seleccionados dentro de la FPGA, observando su comportamiento bajo condiciones controladas. Así, se podría llevar a cabo el desarrollo de técnicas de mitigación más efectivas y específicas basadas en la localización y función de los bits afectados.
- Exposición del Dispositivo SoC a Radiación Real: Finalmente, se considera esencial llevar a cabo experimentos donde el mismo circuito previamente estudiado sea expuesto a radiación real. Esto complementaría las simulaciones y pruebas virtuales con datos empíricos, proporcionando una validación de las metodologías y técnicas de mitigación en un entorno realista. La exposición a radiación real ayudaría a entender

el comportamiento o reacción física real del circuito implementado en los dispositivos basados en silicio.

8 Referencias

[1] Arquimea, "FPGA GEO SCAU Satélite Quantum," [en línea]. Disponible: https://www.arquimea.com/es/casos-exito/fpga-geo-scau-satelite-quantum/. [Última consulta: febrero de 2024].

[2] J. T. Wallmark and S. M. Marcus, “Minimum Size and Maximum Packing Density of Non-Redundant Semiconductor Devices,” Proc. IRE, vol. 50, pp. 286–298, Mar. 1962.

[3] D. Binder, E. C. Smith and A. B. Holman, "Satellite Anomalies from Galactic Cosmic Rays," in IEEE Transactions on Nuclear Science, vol. 22, no. 6, pp. 2675-2680, Dec. 1975.

[4] T. C. May and M. H. Woods, "Alpha-Particle-Induced Soft Errors in Dynamic Memories," IEEE Trans. Electron Devices, vol. ED-26, no. 2, pp. 2-9, 1979.

[5] J. C. Pickel and J. T. Blandford, "Modeling for Single Event Error Rate Prediction," DNA-TR-84-317, June 1984.

[6] Los Alamos National Laboratory, "The Invisible Neutron Threat," [en línea]. Disponible: https://web.archive.org/web/20210210130840/https://www.lanl.gov/science/NSS/issue1_2012/story4full.shtml. [Última consulta: febrero de 2024].

[7] G. E. Moore, “Cramming more components onto integrated circuits,” Electronics, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[8] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted MOSFET’s with very small physical dimensions,” IEEE J. Solid-State Circuits, vol. SC-9, no. 5, pp. 256–268, Oct. 1974.

[9] RealDigital, "Blackboard," [en línea]. Disponible: https://www.realdigital.org/hardware/blackboard. [Última consulta: Abril de 2024].

[10] Xilinx, "Zynq 7000 SoC Technical Reference Manual," [en línea]. Disponible: https://docs.xilinx.com/viewer/bookattachment/mxcNFn1EFZjLI1eShoEn5w/pnoMLQXFIWQ6Jhoj0BUsTQ. [Última consulta: Marzo de 2024].

[11] A. Lindoso, M. Garcia-Valderas, L. Entrena, Y. Morilla, and P. Martin-Holgado, "Evaluation of the Suitability of NEON SIMD Microprocessor Extensions Under Proton Irradiation," IEEE Transactions on Nuclear Science, vol. 65, no. 8, pp. 1835–1842, Aug. 2018.

[12] A. E. Delgado, J. Mira, and S. Dormido Canto, "Teoría de Electrónica Digital," 2nd ed., pp. Apéndice A, 2001.

[13] Xilinx, "7-Series Architecture Overview," [en línea]. Disponible: https://xilinx.eetrend.com/files-eetrend-xilinx/forum/201509/9204-20390-7_series_architecture_overview.pdf. [Última consulta: Abril de 2024].

[14] Xilinx, "7 Series FPGAs Data Sheet: Overview, DS180 (v2.6.1)," Sep. 8, 2020, [en línea]. Disponible: https://docs.amd.com/v/u/en-US/ds180_7Series_Overview. [Última consulta: Abril de 2024].

[15] E. Petersen, "Single Event Effects in Aerospace," 1st ed., pp. 1-3, 2011.

[16] T. C. Carusone, D. A. Johns, and K. W. Martin, "Analog Integrated Circuit Design," 2nd ed., pp. 38, 2011.

[17] B. C. Devnath, "Study on MOSFET (Metal Oxide Semiconductor Field Effect Transistor) and CNTFET (Carbon Nanotube Field Effect Transistor) also VLSI Circuit Design Using Both MOSFET & CNTFET," Oct. 2017, [en línea]. Disponible: https://www.researchgate.net/publication/352520431_STUDY_ON_MOSFET_METAL_OXIDE_SEMICONDUCTOR_FIELD_EFFECT_TRANSISTOR_AND_CNTFET_CARBON_NANOTUBE_FIELD_EFFECT_TRANSISTOR_ALSO_VLSI_CIRCUIT_DESIGN_USING_BOTH_MOSFET_CNTFET. [Última consulta: Abril de 2024].

[18] E. Petersen, "Single Event Effects in Aerospace," 1st ed., pp. 13-14, 2011.

[19] D. Kobayashi, "Scaling Trends of Digital Single-Event Effects: A Survey of SEU and SET Parameters and Comparison With Transistor Performance," IEEE Transactions on Nuclear Science, vol. 68, no. 2, pp. 157-164, Feb. 2021.

[20] J. M. Armani, G. Simon, and P. Poirot, "Low-energy neutron sensitivity of recent generation SRAMs," IEEE Transactions on Nuclear Science, vol. 51, no. 5, pp. 2811-2816, Oct. 2004.

[21] J. Baggio, D. Lambert, V. Ferlet-Cavrois, P. Paillet, C. Marcandella, and O. Duhamel, "Single event upsets induced by 1-10 MeV neutrons in static-RAMs using mono-energetic neutron sources," IEEE Transactions on Nuclear Science, vol. 54, no. 6, pp. 2149-2155, Dec. 2007.

[22] A. Haran, J. Barak, L. Weissman, D. David, and E. Keren, "14 MeV neutrons SEU cross sections in deep submicron devices calculated using heavy ion SEU cross sections," IEEE Transactions on Nuclear Science, vol. 58, no. 3, pp. 848-854, July 2011.

[23] F. Wrobel, J. M. Palau, M. C. Calvet, O. Bersillon, and H. Duarte, "Incident of multi-particle events on soft error rates caused by n-Si nuclear reactions," IEEE Transactions on Nuclear Science, vol. 47, no. 6, pp. 2580-2585, Dec. 2000.

[24] F. Wrobel, J. M. Palau, M. C. Calvet, and P. Iacconi, "Contribution of SiO2 in neutron-induced SEU in SRAMs," IEEE Transactions on Nuclear Science, vol. 50, no. 6, pp. 2055-2059, Dec. 2003.

[25] F. Darracq, T. Beauchene, V. Pouget, H. Lapuyade, D. Lewis, P. Fouillat, and A. Touboul, "Single-event sensitivity of a single SRAM cell," IEEE Transactions on Nuclear Science, vol. 49, no. 3, pp. 1486-1490, June 2002.

[26] R. Baumann, "Soft Errors in Advanced Semiconductor Devices-Part I: The Three Radiation Sources," IEEE Transactions on Device and Materials Reliability, vol. 1, no. 1, pp. 17-22, 2001.

[27] F. Wang and V. D. Agrawal, "Single Event Upset: An Embedded Tutorial," 21st International Conference on VLSI Design (VLSID 2008), Hyderabad, India, 2008, pp. 429-434.

[28] C. Detcheverry, C. Dachs, E. Lorfevre, C. Sudre, G. Bruguier, J. M. Palau, J. Gasiot, and R. Ecoffet, "SEU Critical Charge and Sensitive Area in A Submicron CMOS Technology," IEEE Transactions on Nuclear Science, vol. 44, no. 6, pp. 2266-2273, Dec. 1997.

[29] K. Soliman and D. K. Nichols, "Latchup in CMOS devices from heavy ions," IEEE Transactions on Nuclear Science, vol. 30, no. 6, pp. 4514-4519, Dec. 1983.

[30] J. M. Hutson, J. A. Pellish, A. D. Tipton, G. Boselli, M. A. Xapsos, H. Kim, M. Friendlich, M. Campola, S. Seidleck, K. Label, A. Marshall, X. Deng, R. Baumann, R. A. Reed, R. D. Schrimpf, R. A. Weller, and L. W. Massengill, "Evidence for lateral angle effect on single-event latchup in 65 nm SRAMs," IEEE Transactions on Nuclear Science, vol. 56, no. 1, pp. 208-213, Feb. 2009.

[31] G. Bruguier and J. M. Palau, "Single particle induced latchup," IEEE Transactions on Nuclear Science, vol. 43, no. 2, pp. 522-532, Apr. 1996.

[32] G. Boselli, V. Reddy, and C. Duvvury, "Latch-up in 65 nm CMOS technology: A scaling perceptive," in Proc. Int. Rel. Phys. Symp., San Jose, CA, USA, 2005, pp. 137-144.

[33] A. H. Johnston, B. W. Hughlock, M. P. Baze, and R. E. Plaag, "The effect of temperature on single-particle latchup," IEEE Transactions on Nuclear Science, vol. 38, no. 6, pp. 1435-1441, Dec. 1991.

[34] M. Strzempa-depre, J. Harter, C. Werner, H. Skapa, and R. Kassing, "Static and transient latchup simulation of VLSI-CMOS with an improved physical design model," IEEE Transactions on Electron Devices, vol. 34, no. 6, pp. 1290-1296, Jun. 1987.

[35] EIA/JEDEC, "Test Procedures for the Measurement of Single Event Effects in Semiconductor Devices from Heavy Ion Irradiation," Electronic Industries Association, Engineering Department, Arlington, VA, 1996.

[36] AMD, "LogiCORE IP Soft Error Mitigation Controller v3.1 User Guide (UG764)," [en línea]. Disponible: https://docs.amd.com/v/u/en-US/ug764_sem. [Última consulta: Abril de 2024].

[37] AMD, "Soft Error Mitigation Controller v4.1 LogiCORE IP Product Guide (PG036)," [en línea]. Disponible: https://docs.amd.com/r/en-US/pg036_sem/Soft-Error-Mitigation-Controller-v4.1-LogiCORE-IP-Product-Guide. [Última consulta: febrero de 2024].

[38] AMD, "LogiCORE™ IP Processing System 7 v5.5 Product Guide (PG082)," [en línea]. Disponible: https://docs.amd.com/v/u/en-US/pg082-processing-system7. [Última consulta: Mayo de 2024].

[39] AMD, "LogiCORE™ IP Processor System Reset Module Core Product Guide (PG036)," [en línea]. Disponible: https://docs.amd.com/r/en-US/pg036_sem/Soft-Error-Mitigation-Controller-v4.1-LogiCORE-IP-Product-Guide. [Última consulta: Mayo de 2024].

[40] AMD, "AXI Interconnect v2.1 LogiCORE IP Product Guide (PG059)," [en línea]. Disponible: https://docs.amd.com/r/en-US/pg059-axi-interconnect. [Última consulta: Mayo de 2024].

[41] "Tabla del Código ASCII," [en línea]. Disponible: https://elcodigoascii.com.ar/. [Última consulta: Mayo de 2024].

[42] AMD, "7 Series FPGAs Configuration User Guide UG470 (v1.17)," Dec. 5, 2023, [en línea]. Disponible: https://docs.amd.com/v/u/en-US/ug470_7Series_Config. [Última consulta: Mayo de 2024].

[43] AMD, "Device Reliability Report User Guide (UG116)," [en línea]. Disponible: https://docs.amd.com/r/en-US/ug116. [Última consulta: Mayo de 2024].

[44] AMD, "Fast Fourier Transform v9.1 LogiCORE IP Product Guide PG109," May 4, 2022, [en línea]. Disponible: https://docs.amd.com/r/en-US/pg109-xfft. [Última consulta: Mayo de 2024].

Anexos

ANEXO A: Código del Controlador SEM IP (VHDL)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library unisim;
use unisim.vcomponents.all;

------------------------------------------------------------------------------
--
-- Entity
------------------------------------------------------------------------------
--

entity sem_0_sem_example is
port (
  clk                           : in    std_logic;
  status_heartbeat              : out   std_logic;
  status_initialization         : out   std_logic;
  status_observation            : out   std_logic;
  status_correction             : out   std_logic;
  status_classification         : out   std_logic;
  status_injection              : out   std_logic;
  status_essential              : out   std_logic;
  status_uncorrectable          : out   std_logic;
  inject_strobe                 : in    std_logic;
  inject_address                : in    std_logic_vector(39 downto 0);
  monitor_tx                    : out   std_logic;
  monitor_rx                    : in    std_logic;

  icap_grant                    : in    std_logic
  );
end entity sem_0_sem_example;

------------------------------------------------------------------------------
--
-- Architecture
------------------------------------------------------------------------------
--

architecture xilinx of sem_0_sem_example is

  ------------------------------------------------------------------------------
--
  -- Define local constants.
  ------------------------------------------------------------------------------
--

  constant TCQ : time := 1 ps;

  ------------------------------------------------------------------------------
--
  -- Declare non-library components.
```

```
  ------------------------------------------------------------------------
--

  component sem_0
  port (
    status_heartbeat            : out   std_logic;
    status_initialization       : out   std_logic;
    status_observation          : out   std_logic;
    status_correction           : out   std_logic;
    status_classification       : out   std_logic;
    status_injection            : out   std_logic;
    status_essential            : out   std_logic;
    status_uncorrectable        : out   std_logic;
    monitor_txdata              : out   std_logic_vector(7 downto 0);
    monitor_txwrite             : out   std_logic;
    monitor_txfull              : in    std_logic;
    monitor_rxdata              : in    std_logic_vector(7 downto 0);
    monitor_rxread              : out   std_logic;
    monitor_rxempty             : in    std_logic;
    inject_strobe               : in    std_logic;
    inject_address              : in    std_logic_vector(39 downto 0);
    fecc_crcerr                 : in    std_logic;
    fecc_eccerr                 : in    std_logic;
    fecc_eccerrsingle           : in    std_logic;
    fecc_syndromevalid          : in    std_logic;
    fecc_syndrome               : in    std_logic_vector(12 downto 0);
    fecc_far                    : in    std_logic_vector(25 downto 0);
    fecc_synbit                 : in    std_logic_vector(4 downto 0);
    fecc_synword                : in    std_logic_vector(6 downto 0);
    icap_o                      : in    std_logic_vector(31 downto 0);
    icap_i                      : out   std_logic_vector(31 downto 0);
    icap_csib                   : out   std_logic;
    icap_rdwrb                  : out   std_logic;
    icap_clk                    : in    std_logic;
    icap_request                : out   std_logic;
    icap_grant                  : in    std_logic
    );
  end component;

  component sem_0_sem_cfg
  port (
    fecc_crcerr                 : out   std_logic;
    fecc_eccerr                 : out   std_logic;
    fecc_eccerrsingle           : out   std_logic;
    fecc_syndromevalid          : out   std_logic;
    fecc_syndrome               : out   std_logic_vector(12 downto 0);
    fecc_far                    : out   std_logic_vector(25 downto 0);
    fecc_synbit                 : out   std_logic_vector(4 downto 0);
    fecc_synword                : out   std_logic_vector(6 downto 0);
    icap_o                      : out   std_logic_vector(31 downto 0);
    icap_i                      : in    std_logic_vector(31 downto 0);
    icap_clk                    : in    std_logic;
    icap_csib                   : in    std_logic;
    icap_rdwrb                  : in    std_logic
    );
  end component;

  component sem_0_sem_mon
  port (
```

```
    icap_clk                    : in    std_logic;
    monitor_tx                  : out   std_logic;
    monitor_rx                  : in    std_logic;
    monitor_txdata              : in    std_logic_vector(7 downto 0);
    monitor_txwrite             : in    std_logic;
    monitor_txfull              : out   std_logic;
    monitor_rxdata              : out   std_logic_vector(7 downto 0);
    monitor_rxread              : in    std_logic;
    monitor_rxempty             : out   std_logic
    );
  end component;

  ------------------------------------------------------------------------
  -- Declare signals.
  ------------------------------------------------------------------------

  signal clk_ibufg : std_logic;

  signal status_heartbeat_internal : std_logic;
  signal status_initialization_internal : std_logic;
  signal status_observation_internal : std_logic;
  signal status_correction_internal : std_logic;
  signal status_classification_internal : std_logic;
  signal status_injection_internal : std_logic;
  signal status_essential_internal : std_logic;
  signal status_uncorrectable_internal : std_logic;

  signal monitor_txdata         : std_logic_vector(7 downto 0);
  signal monitor_txwrite        : std_logic;
  signal monitor_txfull         : std_logic;
  signal monitor_rxdata         : std_logic_vector(7 downto 0);
  signal monitor_rxread         : std_logic;
  signal monitor_rxempty        : std_logic;
  signal fecc_crcerr            : std_logic;
  signal fecc_eccerr            : std_logic;
  signal fecc_eccerrsingle      : std_logic;
  signal fecc_syndromevalid     : std_logic;
  signal fecc_syndrome          : std_logic_vector(12 downto 0);
  signal fecc_far               : std_logic_vector(25 downto 0);
  signal fecc_synbit            : std_logic_vector(4 downto 0);
  signal fecc_synword           : std_logic_vector(6 downto 0);
  signal icap_o                 : std_logic_vector(31 downto 0);
  signal icap_i                 : std_logic_vector(31 downto 0);
  signal icap_csib              : std_logic;
  signal icap_rdwrb             : std_logic;
  signal icap_unused            : std_logic;
  signal icap_grant_internal    : std_logic;
  signal icap_clk               : std_logic;

  ------------------------------------------------------------------------
--
  --
  ------------------------------------------------------------------------
--

  begin

  ------------------------------------------------------------------------
  -- This design (the example, including the controller itself) is fully
```

```
-- synchronous; the global clock buffer is instantiated here to drive
-- the icap_clk signal.
--------------------------------------------------------------------

example_ibuf : IBUF
port map (
  I => clk,
  O => clk_ibufg
  );

example_bufg : BUFGCE
port map (
  I => clk_ibufg,
  O => icap_clk,
  CE => '1'
  );

--------------------------------------------------------------------
-- The controller sub-entity is the kernel of the soft error mitigation
-- solution.  The port list is dynamic based on the IP core options.
--------------------------------------------------------------------

example_controller : sem_0
port map (
  status_heartbeat => status_heartbeat_internal,
  status_initialization => status_initialization_internal,
  status_observation => status_observation_internal,
  status_correction => status_correction_internal,
  status_classification => status_classification_internal,
  status_injection => status_injection_internal,
  status_essential => status_essential_internal,
  status_uncorrectable => status_uncorrectable_internal,
  monitor_txdata => monitor_txdata,
  monitor_txwrite => monitor_txwrite,
  monitor_txfull => monitor_txfull,
  monitor_rxdata => monitor_rxdata,
  monitor_rxread => monitor_rxread,
  monitor_rxempty => monitor_rxempty,
  inject_strobe => inject_strobe,
  inject_address => inject_address,
  fecc_crcerr => fecc_crcerr,
  fecc_eccerr => fecc_eccerr,
  fecc_eccerrsingle => fecc_eccerrsingle,
  fecc_syndromevalid => fecc_syndromevalid,
  fecc_syndrome => fecc_syndrome,
  fecc_far => fecc_far,
  fecc_synbit => fecc_synbit,
  fecc_synword => fecc_synword,
  icap_o => icap_o,
  icap_i => icap_i,
  icap_csib => icap_csib,
  icap_rdwrb => icap_rdwrb,
  icap_clk => icap_clk,
  icap_request => icap_unused,
  icap_grant => icap_grant_internal
  );

-- Para establecer una sincronización correcta
-- de la señal icap_grant con el reloj icap_clk
```

```
-- empleamos el reloj global clk_ibufg que
-- a su vez alimenta la señal icap_clk
icap_sync_proc:
process(clk_ibufg)
begin
  if clk_ibufg'event and clk_ibufg = '1' then
      icap_grant_internal <= icap_grant;
  end if;
end process;

status_heartbeat <= status_heartbeat_internal;
status_initialization <= status_initialization_internal;
status_observation <= status_observation_internal;
status_correction <= status_correction_internal;
status_classification <= status_classification_internal;
status_injection <= status_injection_internal;
status_essential <= status_essential_internal;
status_uncorrectable <= status_uncorrectable_internal;

------------------------------------------------------------------------
-- The cfg sub-entity contains the device specific primitives to access
-- the internal configuration port and the frame crc/ecc status signals.
------------------------------------------------------------------------

example_cfg : sem_0_sem_cfg
port map (
  fecc_crcerr => fecc_crcerr,
  fecc_eccerr => fecc_eccerr,
  fecc_eccerrsingle => fecc_eccerrsingle,
  fecc_syndromevalid => fecc_syndromevalid,
  fecc_syndrome => fecc_syndrome,
  fecc_far => fecc_far,
  fecc_synbit => fecc_synbit,
  fecc_synword => fecc_synword,
  icap_o => icap_o,
  icap_i => icap_i,
  icap_csib => icap_csib,
  icap_rdwrb => icap_rdwrb,
  icap_clk => icap_clk
  );

------------------------------------------------------------------------
-- The mon sub-entity contains a UART for communication purposes.
------------------------------------------------------------------------

example_mon : sem_0_sem_mon
port map (
  icap_clk => icap_clk,
  monitor_tx => monitor_tx,
  monitor_rx => monitor_rx,
  monitor_txdata => monitor_txdata,
  monitor_txwrite => monitor_txwrite,
  monitor_txfull => monitor_txfull,
  monitor_rxdata => monitor_rxdata,
  monitor_rxread => monitor_rxread,
  monitor_rxempty => monitor_rxempty
  );

------------------------------------------------------------------------
```

```
  --
  ------------------------------------------------------------------------

end architecture xilinx;
```

ANEXO B: Código Python de (lfa.py)

Código Python (lfa.py)

```
#=============================================#
#                                             #
#  Este script de Python lee el fichero EBC   #
#  y genera direcciones LFA copiándolas al    #
#  fichero inject_address.h, siendo este      #
#  último ubicado dentro del proyecto, donde  #
#  se realizarán las inyecciones de errores.  #
#                                             #
#=============================================#


# Ruta del archivo
ruta_archivo =
'C:\\Users\\Usuario\\Desktop\\PFG\\Proyecto_SEM_13_02_2024_valido\\Proyecto
_SEM_VIVADO\\Proyecto_SEM.runs\\impl_1\\design_1_wrapper.ebd'

# Lista para almacenar resultados
resultados = []

# Contador de dirección de frame lineal
linear_frame_address = 0

# Apertura el archivo en modo de lectura
with open(ruta_archivo, 'r') as f:
    # Recorrer cada línea del archivo
    for num_linea, linea in enumerate(f, start=1):
        # Verificar si la línea contiene algún dato diferente de cero
        if '1' in linea:
            # Dirección del bit dentro de la palabra
            bit_position = linea.find('1')

            # Dirección de la palabra dentro del frame
            word_address = (num_linea - 1) // 32

            # Dirección completa
            binary_linear_frame_address = format(linear_frame_address,
'017b')
            binary_word_address = format(word_address, '07b')
            binary_bit_address = format(bit_position, '05b')
            binary_number = binary_linear_frame_address +
binary_word_address + binary_bit_address

            # Convertir el número binario a hexadecimal
            hexadecimal_number = format(int(binary_number, 2), '08X')

            # Agregar la dirección al resultado
            resultados.append(hexadecimal_number)

        # Incrementar la dirección de frame lineal después de cada conjunto
de 101 líneas
        if num_linea % 101 == 0:
            linear_frame_address += 1
```

```
# Convertir los números hexadecimales en una cadena de inicialización en C
cadena_c = "#ifndef __INJECT_ADDRESS_H\n"
cadena_c += "#define __INJECT_ADDRESS_H\n"
cadena_c += "const unsigned int LFA[] = {\n"
cadena_c += ",\n".join(["    0x" + num for num in resultados])
cadena_c += "\n};\n"
cadena_c += "#endif /* __INJECT_ADDRESS_H */"

# Guardar la cadena de inicialización en un archivo de salida
inject_address.h
ruta_salida =
'C:\\Users\\Usuario\\Desktop\\PFG\\Proyecto_SEM_13_02_2024_valido\\Proyecto
_SEM_VITIS\\xgpio_example_1\\src\\inject_address.h'
with open(ruta_salida, 'w') as f:
    f.write(cadena_c)

print('Frame Addresses have been generated!')
```

ANEXO C: Código de Aplicación en C (en Vitis)

```
/************************ Include Files ******************************/
#include "xparameters.h"
#include "xgpio_l.h"
#include "xgpio.h"
#include "xil_printf.h"
#include "stdio.h"
#include "stdbool.h"
#include "xdevcfg.h"
#include "xiicps.h"
#include "stdlib.h"
#include "string.h"
#include "inject_address.h"
/*****************************************************************************
***/

/*********************************** i2c
************************************/
#define IIC_DEVICE_ID  XPAR_XIICPS_0_DEVICE_ID
#define IIC_SLAVE_ADDR 0x40
#define IIC_SCLK_RATE  400000

int IicPsSlavePolledExample();
int iic_init(u16 DeviceId);

XIicPs Iic;
u8 data_w;
u8 data_r;
/*****************************************************************************
***/

/********************************** CONSTANTS
***********************************/
#define GPIO_REG_BASEADDR_ICAP_GRANT        XPAR_GPIO_0_BASEADDR
#define GPIO_REG_BASEADDR_INJECT_ADDRESS      XPAR_GPIO_1_BASEADDR
#define GPIO_REG_BASEADDR_INJECT_STROBE     XPAR_GPIO_2_BASEADDR
#define GPIO_REG_BASEADDR_MONITOR_RX        XPAR_GPIO_3_BASEADDR
#define GPIO_REG_BASEADDR_SEM_STATUS        XPAR_GPIO_4_BASEADDR

#define DEVICE_CONFIG_ID                    XPAR_XDCFG_0_DEVICE_ID
/*****************************************************************************
***/

/********************************** MASKS
***********************************/
#define INJECT_ADDRESS_MASK        0xFFFFFFFF
#define STATUS_IDLE_MASK           0b00000000
#define STATUS_INITIALIZATION_MASK 0b00000010
#define STATUS_OBSERVATION_MASK    0b00000100
#define STATUS_CORRECTION_MASK     0b00001000
#define STATUS_CLASSIFICATION_MASK 0b00010000
#define STATUS_INJECTION_MASK      0b00100000
#define STATUS_ESSENTIAL_MASK      0b01000000
#define STATUS_UNCORRECTABLE_MASK  0b10000000

#define DEVC_CTRL_PCAP_PR_MASK     0x08000000
#define DEVC_CTRL_PCAP_MODE_MASK   0x04000000
```

```
#define IDLE_MASK                   0xE0
#define OBSERVATION_MASK            0xA0
#define SOFT_RESET_MASK             0xB0
#define INJECT_MASK                 0xC0
#define NULL_MASK                   0x00000000

#define ENABLE                      0x00000001
#define DISABLE                     0x00000000
/*****************************************************************************
***/

/********************************** GPIOs ************************/
XGpio Gpio_ICAP_GRANT                                             ;
Gpio_INJECT_ADDRESS                                               ;
Gpio_INJECT_STROBE                                                ;
Gpio_MONITOR_RX                                                   ;
Gpio_SEM_STATUS                                                   ;
/*****************************************************************/

/******************************** DEVICE CONFIG ********************/
XDcfg        deviceConfig                                         ;
XDcfg_Config *Ptr_deviceConfig                                    ;
/*****************************************************************************
***/

/*************************** CHANNELS AND PORTS ********************/
#define GPIO_PORTS_OUTPUT 0x00000000 // PORTS AS OUTPUT
#define GPIO_PORTS_INPUT  0xFFFFFFFF // PORTS AS INPUT

#define CHANNEL_1         1
#define CHANNEL_2         2

/************************** FUNCTIONS DECLARATION ******************/
void hardware_init           (void)                               ;
void delay                   (uint32_t miliseconds)               ;
void setErrorInjectionCmdAddr(XGpio address, uint8_t channel, uint32_mask)
;
void device_config_init      (void)                               ;
void XDcfg_EnableICAP        (XDcfg *InstancePtr)                 ;
void enable_icap_grant       (void)                               ;
void disable_icap_grant      (void)                               ;
void assert_inject_strobe    (void)                               ;
void ENTER_IDLE_STATE(void)                                       ;
void ENTER_OBSERVATION_STATE(void)                                ;
void print_SEM_STATUS(void)                                       ;
u8 retrieve_SEM_STATUS(void)                                      ;
unsigned int generate_ERROR_INJECTION_ADDRESS(void)               ;
void PERFORM_ERROR_INJECTION(void)                                ;
void PERFORM_SOFT_RESET(void)                                     ;
enum t_states get_CURRENT_STATE(void)                             ;
/*****************************************************************/

/*************************** GLOBAL VARIABLES **********************/
unsigned int inject_address;
unsigned int offset          = 1;
unsigned int errors_detected = 0;
unsigned int lfa_previous    = 0x00000000;
/*****************************************************************/
```

```
/************************* SEM CONTROLLER STATES ******************/
enum t_states{
        INITIALIZATION,
        OBSERVATION,
        CORRECTION,
        IDLE,
        ESSENTIAL,
        CLASSIFICATION,
        UNCORRECTABLE,
        INJECTION
};
/*****************************************************************/

int main(void)
{

        int INJECTION_OBSERVATION = 1; // 1 -> start inyection
        enum t_states state = INITIALIZATION;

        hardware_init();
        device_config_init();
        XDcfg_EnableICAP(&deviceConfig);

        while(1){

                state = get_CURRENT_STATE();

                switch(state){

                case INITIALIZATION:
                        /* Do nothing */
                break;

                case OBSERVATION:
                        /* Enter IDLE state to perform injections */
                        ENTER_IDLE_STATE();
                        /****************************************/
                break;

                case IDLE:
                if(INJECTION_OBSERVATION){// 1 -> Inject error
                /* Error injection here */
                PERFORM_ERROR_INJECTION();
                /***********************/
                }else{// 0 -> Observe
                /* Go to observation state */
                ENTER_OBSERVATION_STATE();
                /***********************/

                                                        }
                INJECTION_OBSERVATION =! INJECTION_OBSERVATION;
                break;

                case INJECTION:
                /* Do nothing */
                break;

                case ESSENTIAL:
                xil_printf(     "Error esencial "
```

```
                                "en la direccion de "
                                "memoria 0x%010X\n"
                                "*****************************\n"
                                "*Total errores detectados*\n"
                                "*                    %d"
                                "                     *\n"
                                "*****************************\n",
                                inject_address, errors_detected);

                        /* Soft Reset */
                        ENTER_IDLE_STATE();
                        /**************/
                        break;

                        case CLASSIFICATION:
                        xil_printf("status: CLASSIFICATION\n");
                        break;

                        case CORRECTION:
                        xil_printf("status: CORRECTION\n");
                        errors_detected++;
                        break;

                        case UNCORRECTABLE:
                        xil_printf("status: UNCORRECTABLE\n");
                        break;
                        }
                }
                return 0;
}

enum t_states get_CURRENT_STATE(void){

        u8 SEM_status = retrieve_SEM_STATUS();

        if(SEM_status == STATUS_IDLE_MASK){
                return IDLE;
        }
        if(SEM_status == STATUS_INITIALIZATION_MASK){
                return INITIALIZATION;
        }
        if(SEM_status == STATUS_OBSERVATION_MASK){
                return OBSERVATION;
        }
        if(SEM_status == STATUS_INJECTION_MASK){
                return INJECTION;
        }
        if(SEM_status == STATUS_CORRECTION_MASK){
                return CORRECTION;
        }
        if(SEM_status == STATUS_CLASSIFICATION_MASK){
                return CLASSIFICATION;
        }
        if(SEM_status == (STATUS_ESSENTIAL_MASK |
                          STATUS_UNCORRECTABLE_MASK))
        {
                return UNCORRECTABLE;
        }
```

```
        if(SEM_status == (STATUS_ESSENTIAL_MASK |
                        STATUS_OBSERVATION_MASK))
        {
                return ESSENTIAL;
        }

}

/* It takes LFA generated from EBC FILE */
unsigned int generate_ERROR_INJECTION_ADDRESS(void){

        int valid_frame = 0;

        while(!valid_frame){
                /* Me quedo con la linear frame address */
                unsigned int lfa = LFA[offset] >> 12;
                xil_printf("lfa: 0x%010X --- offset: %d\n", LFA[offset]);
                /* No se puede inyectar fallos en frames adyacentes */
                if((lfa - lfa_previous) >= 2){
                        valid_frame = 1;
                        lfa_previous = lfa;
                        inject_address = LFA[offset];
                }
                offset+=1;
        }
        xil_printf("Inyeccion en: 0x%010X --- offset: %d\n",
inject_address, offset-1);
        return inject_address;

}


void PERFORM_ERROR_INJECTION(void){

        u32 ERROR_ADDRESS = generate_ERROR_INJECTION_ADDRESS();

        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_2,
INJECT_MASK);
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_1,
ERROR_ADDRESS);
        assert_inject_strobe();
}

void PERFORM_SOFT_RESET(void){
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_2,
SOFT_RESET_MASK);
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_1,
NULL_MASK);
        assert_inject_strobe();
}


u8 retrieve_SEM_STATUS(void){
        return XGpio_DiscreteRead(&Gpio_SEM_STATUS, 1);
}

void print_SEM_STATUS(void){
        u8 SEM_status = retrieve_SEM_STATUS();
        xil_printf("Current SEM STATUS: ");
```

```
        for (int i = 7; i >= 0; i--) {
                xil_printf("%d", (SEM_status >> i) & 1);
        }
        xil_printf("\n");
}

void ENTER_IDLE_STATE(void){
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_2,
IDLE_MASK);
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_1,
NULL_MASK);
        assert_inject_strobe();
}

void ENTER_OBSERVATION_STATE(void){
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_2,
OBSERVATION_MASK);
        setErrorInjectionCmdAddr(Gpio_INJECT_ADDRESS, CHANNEL_1,
NULL_MASK);
        assert_inject_strobe();
}

void enable_icap_grant(void){
        XGpio_DiscreteWrite(&Gpio_ICAP_GRANT, CHANNEL_1, 0x01);
}

void disable_icap_grant(void){
        XGpio_DiscreteWrite(&Gpio_ICAP_GRANT, CHANNEL_1, 0x00);
}

void assert_inject_strobe(void){
        XGpio_DiscreteWrite(&Gpio_INJECT_STROBE, CHANNEL_1, 0x01);
        XGpio_DiscreteWrite(&Gpio_INJECT_STROBE, CHANNEL_1, 0x00);
}

void setErrorInjectionCmdAddr(XGpio address, uint8_t channel, uint32_t
mask){
        XGpio_DiscreteWrite(&address, channel, mask);
}

void delay(uint32_t miliseconds){
        uint32_t counter = 0;
        uint32_t cycles   = miliseconds * 10000;
        while(counter < cycles){
                counter++;
        }
}


void device_config_init(void){

        int Status;

        Ptr_deviceConfig = XDcfg_LookupConfig(DEVICE_CONFIG_ID);
        Status = XDcfg_CfgInitialize(&deviceConfig , Ptr_deviceConfig,
Ptr_deviceConfig->BaseAddr);
        if (Status != XST_SUCCESS) {
                xil_printf("Device Configuration Initialization
Failed\r\n");
```

```
                return XST_FAILURE;
        }

}


void XDcfg_EnableICAP(XDcfg *InstancePtr){

        delay(500);

        uint32_t RegisterHandler;

        Xil_AssertVoid(InstancePtr != NULL);
        Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);

        /* Pointing & Reading from Device Configuration Base Address */
        RegisterHandler = XDcfg_ReadReg(InstancePtr->Config.BaseAddr,
XDCFG_CTRL_OFFSET);

        /* Setting the PCAP_MODE (bit 26) in the Multiplexer below
         * to have Full Access to the PL Config Module */
        XDcfg_WriteReg(InstancePtr->Config.BaseAddr,
                        XDCFG_CTRL_OFFSET, (RegisterHandler |
DEVC_CTRL_PCAP_MODE_MASK));

        /* Clearing the PCAP_PR (bit 27) in the Multiplexer above
         * to Grant Access to ICAP Path */
        XDcfg_WriteReg(InstancePtr->Config.BaseAddr,
                        XDCFG_CTRL_OFFSET, (RegisterHandler &
(~DEVC_CTRL_PCAP_PR_MASK)));

        /* Now it's time to ENABLE the icap_grant signal
         * through GPIO to be able to interact with
         * the SEM Controller
         */
        enable_icap_grant();
        delay(500);

}


void hardware_init(void){
        int Status;

        /* Initialize the GPIO drivers */
        Status = XGpio_Initialize(&Gpio_ICAP_GRANT,
XPAR_GPIO_0_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                xil_printf("Gpio_ICAP_GRANT Initialization Failed\r\n");
                return XST_FAILURE;
        }
        Status = XGpio_Initialize(&Gpio_INJECT_ADDRESS,
XPAR_GPIO_1_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                xil_printf("Gpio_INJECT_ADDRESS Initialization Failed\r\n");
                return XST_FAILURE;
        }
        Status = XGpio_Initialize(&Gpio_INJECT_STROBE,
XPAR_GPIO_2_DEVICE_ID);
```

```
        if (Status != XST_SUCCESS) {
                xil_printf("Gpio_INJECT_STROBE Initialization Failed\r\n");
                return XST_FAILURE;
        }
        Status = XGpio_Initialize(&Gpio_MONITOR_RX,
XPAR_GPIO_3_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                xil_printf("Gpio_MONITOR_RX Initialization Failed\r\n");
                return XST_FAILURE;
        }
        Status = XGpio_Initialize(&Gpio_SEM_STATUS,
XPAR_GPIO_4_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                xil_printf("Gpio_SEM_STATUS Initialization Failed\r\n");
                return XST_FAILURE;
        }
        XGpio_SetDataDirection(&Gpio_ICAP_GRANT,     CHANNEL_1,
GPIO_PORTS_OUTPUT);
        XGpio_SetDataDirection(&Gpio_INJECT_ADDRESS, CHANNEL_1,
GPIO_PORTS_OUTPUT);
        XGpio_SetDataDirection(&Gpio_INJECT_ADDRESS, CHANNEL_2,
GPIO_PORTS_OUTPUT);
        XGpio_SetDataDirection(&Gpio_INJECT_STROBE,  CHANNEL_1,
GPIO_PORTS_OUTPUT);
        XGpio_SetDataDirection(&Gpio_MONITOR_RX,     CHANNEL_1,
GPIO_PORTS_OUTPUT);
        XGpio_SetDataDirection(&Gpio_SEM_STATUS,     CHANNEL_1,
GPIO_PORTS_INPUT );

        XGpio_DiscreteWrite(&Gpio_ICAP_GRANT, CHANNEL_1, 0x00);
        XGpio_DiscreteWrite(&Gpio_INJECT_ADDRESS, CHANNEL_1, 0x00000000);
        XGpio_DiscreteWrite(&Gpio_INJECT_ADDRESS, CHANNEL_2, 0x00);
        XGpio_DiscreteWrite(&Gpio_ICAP_GRANT, CHANNEL_1, 0x00);
        XGpio_DiscreteWrite(&Gpio_ICAP_GRANT, CHANNEL_1, 0x00);
        XGpio_DiscreteWrite(&Gpio_ICAP_GRANT, CHANNEL_1, 0x00);


        /* Maintaining icap_grant signal deasserted until PCAP activity
finishes */
        disable_icap_grant();
```

ANEXO D: Test Bench para el diseño de FFT con Inyector de Errores XOR(VHDL)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real.all;

entity design_1_wrapper_tb is
end;

architecture test of design_1_wrapper_tb is

  component FFT_wrapper is
  port (
    clk     : in std_logic;
    DIRECT_FFT_OUTPUT : out STD_LOGIC_VECTOR ( 31 downto 0 );
    DISTORDED_FFT_OUTPUT    : out std_logic_vector (31 downto 0); --64 for
1024 samples, 32 for 512
    M_AXIS_DATA_0_tdata : out std_logic_vector(31 downto 0);

    M_AXIS_DATA_0_tlast    : out std_logic;
    m_axis_data_1_tlast    : out std_logic;

    M_AXIS_DATA_0_tvalid   : out std_logic;
    m_axis_data_1_tvalid   : out std_logic;

    M_AXIS_DATA_0_tready : in std_logic;

    PURE_SIN_TONE       : out std_logic_vector(15 downto 0);

    enable  : in std_logic;
    reset_n : in std_logic;
    inject_word : in std_logic_vector(31 downto 0);

    m_axis_data_tuser_0    : out std_logic_vector(23 downto 0)
    );
  end component FFT_wrapper;

  -- Generics
  --constant clk_period : time := 5 ns;
  -- Ports
  signal DISTORDED_FFT_OUTPUT    : std_logic_vector (31 downto 0);
  signal RECOVERED_DISTORDED_TONE    : std_logic_vector (31 downto 0);
  signal tlast_direct           : std_logic;
  signal tlast_inv              : std_logic;
  signal tready                 : std_logic:='1';
  signal tvalid_direct          : std_logic;
  signal tvalid_inv             : std_logic;
  signal m_axis_data_tuser_0    : std_logic_vector (23 downto 0);
  signal m_axis_data_1_tuser    : std_logic_vector (23 downto 0);
  signal M_AXIS_DATA_tdata_sine : std_logic_vector(15 downto 0);

  signal real_fft      : unsigned (15 downto 0);
  signal real_fft_inv  : unsigned (15 downto 0);
  signal imag_fft      : unsigned (15 downto 0);
  signal imag_fft_inv  : unsigned (15 downto 0);
```

```
  signal fft_inv       : unsigned (15 downto 0);
  signal real_fft_sq   : unsigned (31 downto 0);
  signal imag_fft_sq   : unsigned (31 downto 0);
  signal fft_sq_mod    : unsigned (31 downto 0);
  signal fft_mod       : real;
  signal fft_mod_un    : unsigned (15 downto 0);

  signal samples       : std_logic_vector(9 downto 0);
  signal sine          : std_logic_vector(15 downto 0);
  signal SEU           : std_logic_vector(31 downto 0) := (others => '0');

  signal clk     : std_logic  := '0';
  signal reset_n : std_logic  := '0';
  signal enable  : std_logic  := '0';

begin

  design_1_wrapper_inst : FFT_wrapper
    port map (
      DISTORDED_FFT_OUTPUT  => DISTORDED_FFT_OUTPUT,
      M_AXIS_DATA_0_tdata => RECOVERED_DISTORDED_TONE,
      M_AXIS_DATA_0_tlast  => tlast_direct,
      m_axis_data_1_tlast => tlast_inv,
      m_axis_data_0_tready => tready,
      m_axis_data_0_tvalid => tvalid_direct,
      m_axis_data_1_tvalid => tvalid_inv,
      m_axis_data_tuser_0  => m_axis_data_tuser_0,
      PURE_SIN_TONE =>     sine,
      clk => clk,
      enable => enable,
      reset_n => reset_n,
      inject_word => SEU
    );

    one_tic_reset : process
    begin
        enable <= '1';
        reset_n <= '0';
        wait for 20 ns;
        reset_n <= '1';
        wait;
    end process;

    clk <= not clk after 5 ns;

    tready <= '1';
    --
    --DIRECT FFT
    imag_fft <= unsigned(DISTORDED_FFT_OUTPUT(31 downto 16));
    real_fft <= unsigned(DISTORDED_FFT_OUTPUT(15 downto 0));
    --
    imag_fft_sq <= unsigned(signed(DISTORDED_FFT_OUTPUT(31 downto 16)) *
signed(DISTORDED_FFT_OUTPUT(31 downto 16)));
    real_fft_sq <= unsigned(signed(DISTORDED_FFT_OUTPUT(15 downto 0)) *
signed(DISTORDED_FFT_OUTPUT(15 downto 0)));
    --
    fft_sq_mod <= (real_fft_sq + imag_fft_sq);
    fft_mod <= (sqrt(real(to_integer(fft_sq_mod))));
    fft_mod_un <= to_unsigned(natural (fft_mod), 16);
```

```
    --
    samples <= m_axis_data_tuser_0 (9 downto 0);


   --INVERSE FFT
    imag_fft_inv <= unsigned(RECOVERED_DISTORDED_TONE(31 downto 16));
    real_fft_inv <= unsigned(RECOVERED_DISTORDED_TONE(15 downto 0));
    fft_inv <= imag_fft_inv + real_fft_inv;

   Inject_SEU : process
   begin
   SEU <= "00000000000000000000000000000000";
   wait for 34 us;
   report"INJECT SEU------------";
   SEU <= "00000000000000001000000000000000";
   --SEU <= "00000000000000000000000000000000";
   wait for 10 ns;
   SEU <= "00000000000000000000000000000000";
   wait;
   end process;

end;
```

ANEXO E: Inyector de Fallos XOR (VHDL)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Fault_Injector_XOR is
Port (freq_domain_data: in std_logic_vector(31 downto 0);
      error_injection_position: in std_logic_vector(31 downto 0);
      altered_data_output: out std_logic_vector(31 downto 0)
       );
end Fault_Injector_XOR;

architecture rtl of Fault_Injector_XOR is

begin

altered_data_output <= freq_domain_data XOR error_injection_position
```