



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

**Desarrollo de una herramienta de inyección de
fallas para microcontroladores de a bordo
de sistemas espaciales**

TESIS

Que para obtener el título de
Ingeniero Aeroespacial

P R E S E N T A

Brandon Martínez Piña

DIRECTOR DE TESIS

M. I. Christo Aldair Lara Tenorio



Ciudad Universitaria, Cd. Mx., 2026



**PROTESTA UNIVERSITARIA DE INTEGRIDAD Y
HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)**



De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado DESARROLLO DE UNA HERRAMIENTA DE INYECCION DE FALLAS PARA MICROCONTROLADORES DE A BORDO DE SISTEMAS ESPACIALES que presenté para obtener el título de INGENIERO AERROESPACIAL es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

BRANDON MARTINEZ PIÑA
Número de cuenta: 318028569

AGRADECIMIENTOS

Quiero expresar mi más profundo agradecimiento a mi director de tesis, el M.I. Christo Aldair Lara Tenorio, por su guía, apoyo y todas las palabras de aliento que me dieron fuerza en los momentos más difíciles de este proyecto. Sus consejos y paciencia hicieron que este camino fuera mucho más llevadero y enriquecedor, y su confianza en mí me motivó a seguir adelante incluso en los momentos más complicados.

También agradezco a los profesores y compañeros del LIESE por compartir sus conocimientos y acompañarme en esta aventura que, aunque desafiante, ha sido sumamente gratificante, llena de aprendizaje y momentos que siempre recordaré.

A mi familia, por ser mi pilar fundamental; a mi madre, Fabiola, por su amor incondicional y apoyo constante; a mis hermanos, Ricardo y Tania; a mis abuelos, Ana y David; y a mi tío Salvador, quienes también fueron parte importante de este logro. Gracias a todos por estar siempre presentes, por acompañarme en cada paso y por brindarme la fuerza necesaria para seguir adelante, haciendo que cada desafío fuera más fácil de superar.

Y, por supuesto, a mis amigos César, Erik, Alejandro, Cristian, Rodrigo, Emmanuel y Rogelio, así como a muchos más que me han acompañado a lo largo de esta etapa. Gracias por estar en los momentos importantes, por el ánimo, las risas y por hacer que esta experiencia fuera mucho más especial.

No soy muy bueno con las palabras, pero quiero que todas las personas a quienes he mencionado sepan que mi gratitud viene completamente del corazón y que valoro profundamente cada gesto, cada apoyo y cada momento que hemos compartido juntos.

RESUMEN

Los microcontroladores empleados en sistemas espaciales se encuentran expuestos a los efectos de la radiación, la cual puede producir alteraciones en el contenido de la memoria y de los registros internos. Para evaluar el comportamiento de los microcontroladores ante estos efectos, una de las alternativas consiste en realizar ensayos de radiación que permitan reproducir condiciones similares a las del entorno espacial. Sin embargo, este tipo de ensayos requiere infraestructura especializada, lo que representa altos costos de implementación. Como alternativa a este tipo de pruebas, en este trabajo se propone el desarrollo de una herramienta de inyección de fallas que emplea técnicas de depuración en chip, orientada a microcontroladores basados en la arquitectura ARM Cortex-M, con el propósito de simular de manera controlada y reproducible las alteraciones asociadas a los *upset* de evento único (SEU, por su siglas en inglés).

La herramienta presenta una arquitectura modular que permite configurar y ejecutar campañas de inyección de fallas, modificando de manera controlada el contenido de la memoria y de los registros de configuración de periféricos, así como analizar de forma automática los resultados. Asimismo, integra una interfaz gráfica de usuario e incorpora tres modos de operación: manual, automático y mediante archivos CSV.

La evaluación experimental se realizó mediante campañas de inyección de fallas en memoria RAM, memoria *flash* simulada en RAM y registros de configuración de periféricos. En la RAM, el 56.2 % de las fallas fueron clasificadas como silenciosas y el 35.8 % como propagadas; en la memoria *flash* simulada en RAM, el 48.0 % fueron propagadas y el 43.6 % silenciosas; mientras que en los registros de configuración de periféricos, el 52.4 % fueron propagadas y el 30.0 % silenciosas. Estos resultados muestran una mayor proporción de fallas silenciosas en RAM y una mayor propagación de las alteraciones cuando se afectan regiones asociadas con código ejecutable o registros internos que intervienen directamente en el funcionamiento del sistema. Además, la repetición de las campañas mediante archivos CSV produjo resultados idénticos a los experimentos originales, demostrando la reproducibilidad de la herramienta.

En conjunto, los resultados demuestran que la herramienta de inyección de fallas desarrollada proporciona un entorno controlado y reproducible para el estudio de la vulnerabilidad de microcontroladores ARM Cortex-M ante *upset*. Asimismo, su funcionamiento en los sistemas operativos *Windows* y *macOS*, así como su evaluación en un microcontrolador basado en ARM Cortex-M3, evidencian sus capacidades de portabilidad.

<i>Agradecimientos</i>	I
<i>Resumen</i>	II
<i>Índice general</i>	III
<i>Índice de figuras</i>	VI
<i>Índice de tablas</i>	VIII
<i>Siglas</i>	IX

Sección I. Planteamiento del proyecto 1

1 INTRODUCCIÓN	2
1.1 Presentación del problema	3
1.2 Hipótesis	3
1.3 Objetivo	3
1.3.1 Objetivos específicos	3
1.4 Alcances	3
1.5 Justificación	3
2 METODOLOGÍA	5

Sección II. Investigación preliminar 7

3 SATÉLITES ARTIFICIALES	8
3.1 Subsistemas que integran los satélites	8
3.2 Clasificación de los satélites artificiales	10
3.2.1 Según su área de aplicación	10
3.2.2 Según su masa	11
3.3 Estándar <i>CubeSat</i>	12
4 MEDIO AMBIENTE ESPACIAL	14
4.1 Vacío	14
4.2 Entorno neutro	15
4.3 Plasma	16
4.4 Micrometeoritos y basura espacial	17
5 RADIACIÓN ESPACIAL	18
5.1 Fuentes y componentes del entorno de radiación espacial	19
5.1.1 Partículas energéticas solares (SEP)	19
5.1.2 Rayos cósmicos galácticos (GCR)	20
5.1.3 Cinturones de Van Allen	20
5.2 Efectos de la radiación espacial	21
5.2.1 Efectos acumulativos	21
5.2.2 Efectos de evento único (SEE)	22
5.2.3 <i>Soft errors</i>	23
5.2.4 <i>Hard errors</i>	24
5.3 Ensayos de radiación	25
6 INYECCIÓN DE FALLAS (FI)	26
6.1 Clasificación de las herramientas de inyección de fallas	27
6.1.1 Inyección de fallas por <i>hardware</i>	27
6.1.2 Inyección de fallas por <i>software</i>	28

6.1.3	Inyección de fallas por simulación	29
6.1.4	Inyección de fallas híbrida	29
6.1.5	Inyección de fallas por depuración	30
6.2	Confiabilidad	32
6.2.1	Factores que afectan la confiabilidad: falla, error y fallo	32
6.3	Tolerancia a fallas	34
7	UNIDAD DE MICROCONTROLADOR (MCU)	35
7.1	Arquitectura de un microcontrolador	35
7.1.1	Unidad central de procesamiento (CPU)	36
7.1.2	Unidad de memoria	36
7.1.3	Interfaz de entrada/salida (E/S)	37
7.2	Arquitectura ARM	37
7.2.1	Perfiles de la arquitectura ARM	37
7.2.2	Modelo de programación de Cortex-M	38
7.2.3	Organización de la memoria en Cortex-M	39
7.2.4	Excepciones e interrupciones en Cortex-M	40
7.2.5	Infraestructura de depuración en Cortex-M	41
7.2.6	Archivos para la descripción y análisis del sistema	42
8	ESTADO DEL ARTE DE LAS HERRAMIENTAS DE INYECCIÓN DE FALLAS	44
8.1	Herramientas de inyección de fallas por <i>software</i>	44
8.1.1	Inyección de fallas genérica orientada a objetos (GOOFI)	44
8.1.2	Herramienta de inyección de fallas en microprocesadores (MiFIT)	45
8.1.3	Banco de pruebas para estudios en ambientes hostiles sobre circuitos integrados (THESIC)	47
8.1.4	Inyección automática en tiempo real de fallos y errores (FERRARI)	48
8.1.5	Evaluador de tolerancia a fallas y desempeño en red (NFTAPE)	49
8.2	Herramientas de inyección de fallas por simulación	50
8.2.1	Evaluación de la confiabilidad basada en VHDL mediante inyección eficiente de fallas(VERIFY)	50
8.2.2	Herramienta de simulación de inyección de fallas-errores multinivel (MEFISTO)	51
8.2.3	FOCUS	52
8.3	Herramientas de inyección de fallas por depuración	53
8.3.1	Depuración en chip-inyección de fallas (OCD-FI)	53
8.3.2	Depuración en chip (OCD) empleando GDB	54
8.3.3	Depuración en chip basado en JTAG	56
8.4	Análisis de las herramientas de inyección de fallas	57
8.5	Conclusiones del estado del arte	58
	Sección III. Diseño	59
9	DISEÑO CONCEPTUAL	60
9.1	Requisitos del sistema	60
9.2	Descripción del sistema	61
10	DISEÑO A NIVEL SISTEMA	62
10.1	Arquitectura a nivel sistema	62
10.2	Arquitectura de alto nivel de la herramienta de inyección de fallas (FI)	63
11	DISEÑO DE DETALLE E IMPLEMENTACIÓN	66
11.1	Selección del microcontrolador y herramientas de desarrollo	66
11.2	Implementación de la herramienta de inyección de fallas	67

11.2.1 Implementación de la interfaz gráfica de usuario (GUI)	68
11.2.2 Submódulo de preparación del entorno de <i>software</i>	74
11.2.3 Submódulo de análisis de archivos ELF	75
11.2.4 Submódulo de análisis de archivos SVD	76
11.2.5 Submódulo de análisis de RAM y <i>flash</i>	77
11.2.6 Submódulo generador de lista de fallas	79
11.2.7 Submódulo generador de lista de fallas manual	80
11.2.8 Submódulo de detección de bucles	82
11.2.9 Submódulo de gestión de sesión con la MCU	83
11.2.10 Submódulo de gestión de sesión con la MCU y programación en RAM	84
11.2.11 Submódulo de ejecución de campañas de inyección de fallas	85
11.2.12 Submódulo de detección de última campaña	87
11.2.13 Submódulo de ejecución de referencia	88
11.2.14 Submódulo de análisis de resultados	90
11.2.15 Submódulo de visualización y exploración de campañas	92
11.2.16 Submódulo de ejecución integral	93
 Sección IV. Resultados y conclusiones	96
12 RESULTADOS	97
12.1 Entorno experimental	97
12.1.1 <i>Software</i> bajo prueba	97
12.1.2 Configuración de las campañas de FI	97
12.1.3 Validación de la herramienta en una MCU Cortex-M3	103
12.2 Análisis de resultados	103
 <i>Conclusiones</i>	105
 <i>Trabajo a futuro</i>	106
 <i>Referencias</i>	107

Figura 3.1.	Primer satélite artificial <i>Sputnik 1</i>	8
Figura 3.2.	Satélite artificial	8
Figura 3.3.	Satélite de comunicaciones	11
Figura 3.4.	<i>CubeSat</i>	12
Figura 3.5.	Tamaños estandarizados de <i>CubeSat</i>	13
Figura 4.1.	Densidad vs altitud	15
Figura 4.2.	Arrastre atmosférico	15
Figura 4.3.	Resplandor ocasionado por el oxígeno atómico en una nave espacial	16
Figura 4.4.	Propiedades del plasma espacial natural en diferentes regiones del entorno terrestre	16
Figura 4.5.	Basura espacial.	17
Figura 5.1.	Fondo cósmico de microondas.	18
Figura 5.2.	Mecanismos de aceleración de SEP.	19
Figura 5.3.	Interacción del campo magnético terrestre con partículas energéticas solares y rayos cósmicos galácticos.	20
Figura 5.4.	Anillos de Van Allen	21
Figura 5.5.	Anomalía del Atlántico Sur	21
Figura 5.6.	Defecto de Frenkel	22
Figura 5.7.	<i>Upset</i> de un solo <i>bit</i> frente a <i>upset</i> de múltiples <i>bits</i>	24
Figura 6.1.	Entorno de una herramienta de inyección de fallas.	27
Figura 6.2.	Cadena de falla, error y fallo.	33
Figura 7.1.	Arquitectura funcional de un microcontrolador	35
Figura 7.2.	Estructura interna simplificada de un microcontrolador	36
Figura 7.3.	Arquitectura de depuración basada en CMSIS-DAP y <i>CoreSight</i>	42
Figura 8.1.	Arquitectura MiFIT	46
Figura 8.2.	Sistema THESIC dentro de la cámara de vacío disponible en instalaciones del L.B.L (Berkeley California)	48
Figura 8.3.	Arquitectura NFTAPE	49
Figura 8.4.	Infraestructura OCD	54
Figura 8.5.	Arquitectura OCD-GDB	55
Figura 8.6.	Arquitectura OCD-JTAG	56
Figura 9.1.	Diagrama conceptual	61
Figura 10.1.	Diagrama del sistema.	62
Figura 10.2.	Arquitectura de alto nivel de abstracción de la herramientas de inyección de fallas.	64
Figura 11.1.	Tarjeta de desarrollo STM32F4-Discovery.	66
Figura 11.2.	Diseño de detalle de la herramienta de inyección de fallas.	67
Figura 11.3.	Ventana principal de la GUI.	68
Figura 11.4.	Ventana de configuración.	69
Figura 11.5.	Ventana de preparación.	70
Figura 11.6.	Ventana del modo automático.	71
Figura 11.7.	Ventana del modo manual.	72
Figura 11.8.	Ventana del modo CSV.	73
Figura 11.9.	Diagrama de flujo del submódulo de configuración.	74
Figura 11.10.	Diagrama de flujo del módulo de análisis de archivos ELF.	75

Figura 11.11.	Diagrama de flujo del submódulo de análisis de archivos SVD.	77
Figura 11.12.	Diagrama de flujo del módulo de análisis de RAM y <i>flash</i>	78
Figura 11.13.	Diagrama de flujo del submódulo generador de lista de fallas.	80
Figura 11.14.	Diagrama de flujo del submódulo generador de lista de fallas manual.	81
Figura 11.15.	Diagrama de flujo del submódulo de detección de bucles.	82
Figura 11.16.	Diagrama de flujo del submódulo gestión de sesión con la MCU.	84
Figura 11.17.	Diagrama de flujo del módulo de gestión de sesión con la MCU y programación en RAM.	85
Figura 11.18.	Diagrama de flujo del submódulo de ejecución de campañas de inyección de fallas.	87
Figura 11.19.	Diagrama de flujo del submódulo de detección de última campaña.	88
Figura 11.20.	Diagrama de flujo del submódulo de ejecución de referencia.	89
Figura 11.21.	Diagrama de flujo del submódulo de análisis de resultados.	92
Figura 11.22.	Diagrama de flujo del submódulo de visualización y exploración de campañas.	93
Figura 11.23.	Diagrama de flujo del submódulo de ejecución integral	95
Figura 12.1.	Resultados de la campaña 1.	99
Figura 12.2.	Resultados de la campaña 2.	100
Figura 12.3.	Resultados de la campaña 3.	101
Figura 12.4.	Resultados de la campaña 4.	102

Tabla 3.1.	Clasificación de los satélites por masa y costo.	12
Tabla 4.1.	Efectos del medio ambiente espacial	14
Tabla 6.1.	Comparativa de herramientas de depuración.	32
Tabla 7.1.	Principales registros del modelo de programación Cortex-M	39
Tabla 7.2.	Organización general del espacio de direcciones en Cortex-M	39
Tabla 8.1.	Comparativa entre las diferentes herramientas de inyección de fallas.	57
Tabla 9.1.	Requisitos de la herramienta de inyección de fallas.	60
Tabla 11.1.	MCU disponibles en el listado predefinido de la herramienta de FI.	70
Tabla 11.2.	Estados finales y códigos de estado del proceso de inyección de fallas.	91
Tabla 12.1.	Resumen de las campañas de inyección de fallas realizadas.	98
Tabla 12.2.	Resultados en las campañas de FI sobre una MCU Cortex-M3.	103

SIGLAS

ADC	<i>Analog-to-Digital Converter</i>	ESA	<i>European Space Agency</i>
ADCS	<i>Attitude Determination and Control System</i>	FPB	<i>Flash Patch and Breakpoint</i>
ALU	<i>Arithmetic Logic Unit</i>	FERARRI	<i>Fault and Error Automatic Real-time Injector</i>
API	<i>Application Programming Interface</i>	FI	<i>Fault Injection</i>
C&DH	<i>Command and Data Handling</i>	FIM	<i>Fault Injection Manager</i>
CEU	<i>Code Emulated Upsets</i>	FIS	<i>Fault Injection Signals</i>
CMB	<i>Cosmic Microwave Background</i>	FIST	<i>Fault Injection System for Study of Transient Fault Effect</i>
CME	<i>Coronal Mass Ejection</i>	G&N	<i>Guidance and Navigation</i>
CRC	<i>Cyclic Redundancy Check</i>	GCR	<i>Galactic Cosmic Rays</i>
COTS	<i>Commercial Off-The-Shelf</i>	GDB	<i>GNU Debugger</i>
CPU	<i>Central Processing Unit</i>	GEO	<i>Geostationary Equatorial Orbit</i>
CSV	<i>Comma-Separated Values</i>	GOOFI	<i>Generic Object Oriented Fault Injection</i>
CU	<i>Controller Unit</i>	GPS	<i>Global Positioning System</i>
DAP	<i>Debug Access Port</i>	GUI	<i>Graphical User Interface</i>
DAC	<i>Digital-to-Analog Converter</i>	IDE	<i>Integrated Development Environment</i>
DD	<i>Displacement Damage</i>	IP	<i>Intellectual Property</i>
DHCSR	<i>Debug Halting Control and Status Register</i>	ISS	<i>International Space Station</i>
DI	<i>Debugging Interface</i>	JTAG	<i>Joint Test Action Group</i>
DSP	<i>Digital Signal Processing</i>	LAN	<i>Local Area Network</i>
DUT	<i>Device Under Test</i>	LWFI	<i>Lightweight Fault Injectors</i>
DWT	<i>Data Watchpoint and Trace</i>	LR	<i>Link Register</i>
EEPROM	<i>Electrically Erasable Programmable Read-Only Memory</i>	LEO	<i>Low Earth Orbit</i>
ELF	<i>Executable and Linkable Format</i>	LET	<i>Linear Energy Transfer</i>
EPROM	<i>Erasable Programmable Read-Only Memory</i>	MBU	<i>Multiple-Bit Upset</i>
EPS	<i>Electrical Power Subsystem</i>	MCU	<i>Microcontroller Unit</i>
E/S	<i>Entrada/Salida</i>	MEFISTO	<i>Multi-level Error/Fault Injection Simulation Tool</i>
		MEM-AP	<i>Memory Access Port</i>
		MiFIT	<i>Microprocessors Fault Injection Tool</i>

MQM	<i>Message Queuing and Management</i>	SEP	<i>Solar Energetic Particles</i>
MMI	<i>Memory Mapped Interface</i>	SET	<i>Single Event Transient</i>
MSP	<i>Main Stack Pointer</i>	SEU	<i>Single Event Upset</i>
MTTF	<i>Mean Time To Failure</i>	SFR	<i>Special Function Registers</i>
NASA	<i>National Aeronautics and Space Administration</i>	SP	<i>Stack Pointer</i>
NFTAPE	<i>Networked Fault Tolerance And Performance Evaluator</i>	SRAM	<i>Static Random-Access Memory</i>
NVIC	<i>Nested Vectored Interrupt Controller</i>	SVD	<i>System View Description</i>
OBC	<i>On-Board Computer</i>	SWD	<i>Serial Wire Debug</i>
OCD	<i>On-Chip Debug</i>	SWIFI	<i>Software Implemented Fault Injection</i>
PC	<i>Personal Computer</i>	TAP	<i>Test Access Port</i>
PROM	<i>Programmable Read-Only Memory</i>	TT&C	<i>Telemetry, Tracking and Command</i>
PSP	<i>Process Stack Pointer</i>	TCS	<i>Thermal Control Subsystem</i>
PWM	<i>Pulse Width Modulation</i>	TID	<i>Total Ionizing Dose</i>
RAM	<i>Random Access Memory</i>	THESIC	<i>Testbed for Harsh Environment Studies on Integrated Circuits</i>
RCP	<i>Remote Procedure Call</i>	UnACE	<i>Unnecessary for Architecturally Correct Execution</i>
RCT	<i>Run Control & Trace</i>	UNAM	Universidad Nacional Autónoma de México
RISC	<i>Reduced Instruction Set Computer</i>	URSS	Unión de Repúblicas Socialistas Soviéticas
ROM	<i>Read-Only Memory</i>	USB	<i>Universal Serial Bus</i>
RWM	<i>Read & Write Access</i>	VHDL	<i>Very High Speed Integrated Circuits Hardware Description Language</i>
SAA	<i>South Atlantic Anomaly</i>	VERIFY	<i>VHDL-based Evaluation of Reliability by Injecting Faults efficiently</i>
SCIFI	<i>Scan-Chain Implemented Fault Injection</i>	VLSI	<i>Very Large Scale Integration</i>
SDC	<i>Silent Data Corruption</i>	VTOR	<i>Vector Table Offset Register</i>
SEE	<i>Single Event Effect</i>		
SEFI	<i>Single Event Functional Interrupt</i>		
SEL	<i>Single Event Latchup</i>		

SECCIÓN I
PLANTEAMIENTO DEL PROYECTO

1. INTRODUCCIÓN

En la actualidad, el uso de componentes comerciales disponibles (COTS, por sus siglas en inglés) en sistemas espaciales se ha vuelto cada vez más común debido a su bajo costo y fácil acceso. De acuerdo con la NASA, los COTS son “componentes diseñados para aplicaciones comerciales, cuyas características de rendimiento, configuración y confiabilidad son establecidas y controladas por el fabricante, sin requisitos adicionales impuestos por el usuario” [1]. Estos componentes pueden proporcionar capacidades funcionales suficientes para desempeñar determinadas tareas dentro de una misión espacial, por lo que representan una alternativa frente a componentes desarrollados específicamente para aplicaciones espaciales. Sin embargo, no están diseñados para operar bajo las condiciones particulares del entorno espacial, en donde factores como la radiación, las variaciones extremas de temperatura y el vacío pueden afectar su desempeño y confiabilidad.

Dentro de los factores que amenazan la operación de los sistemas electrónicos y, en particular, el correcto funcionamiento de los satélites, la radiación espacial constituye una de las principales preocupaciones, ya que puede inducir alteraciones temporales o permanentes en los dispositivos electrónicos. Entre los fenómenos asociados a la radiación, los efectos de evento único (SEE, por sus siglas en inglés) resultan especialmente críticos, y los *upsets* de evento único (SEU, por sus siglas en inglés) representan una de sus manifestaciones más relevantes, al provocar cambios no intencionados en la información almacenada en registros o memorias, lo que puede derivar en errores o en el mal funcionamiento del sistema.

Para estudiar el comportamiento de los dispositivos electrónicos ante este tipo de entorno, se pueden realizar ensayos en los que los componentes son expuestos a fuentes radiactivas bajo condiciones controladas. Si bien estos ensayos permiten evaluar directamente los efectos producidos, requieren infraestructura especializada e instrumentación específica, lo que incrementa su complejidad y costo.

Ante estas limitaciones, la inyección de fallas (FI, por sus siglas en inglés) se presenta como una estrategia para simular de manera controlada los efectos de la radiación en los sistemas electrónicos, permitiendo validar mecanismos de detección, mitigación y recuperación de errores, así como generar entornos controlados que faciliten el análisis del comportamiento de las unidades de microcontrolador (MCU, por sus siglas en inglés) de a bordo de misiones espaciales ante perturbaciones inducidas.

Con base en lo anterior, el presente trabajo propone el desarrollo de una herramienta de FI implementada mediante técnicas de depuración en chip (OCD, por sus siglas en inglés), adaptable a MCU ARM Cortex-M y orientada a simular efectos *upset*, *stuck-at-0* y *stuck-at-1*, constituyendo así una alternativa para la experimentación y el análisis de fallas en sistemas electrónicos.

En la *Sección I. Planteamiento del proyecto* se presentan los elementos centrales del trabajo, incluyendo la descripción del problema, la hipótesis formulada, el objetivo general y los objetivos específicos, así como los alcances, la justificación y la metodología de diseño propuesta.

En la *Sección II. Investigación preliminar* se presentan los fundamentos teóricos necesarios para comprender el desarrollo del trabajo, incluyendo aspectos relacionados con el medio ambiente espacial, la radiación, la FI y la arquitectura de las MCU. Asimismo, se analiza el estado del arte de las principales herramientas de FI, destacando sus características, ventajas y limitaciones.

En la *Sección III. Diseño* se detalla el proceso de construcción de la herramienta de FI propuesta, abarcando el diseño conceptual, el diseño a nivel sistema y el diseño de detalle sobre una MCU representativa.

Finalmente, en la *Sección IV. Resultados y conclusiones* se presentan y analizan los resultados, evaluando la efectividad de la herramienta desarrollada, así como las conclusiones generales derivadas del proyecto.

1.1 *Presentación del problema*

Las MCU son componentes fundamentales de los sistemas electrónicos empleados en misiones espaciales, por lo que su susceptibilidad a la radiación espacial constituye un aspecto relevante para la confiabilidad del sistema. Entre los efectos que la radiación puede producir se encuentran los SEE, siendo los SEU responsables de alteraciones en registros y memorias que pueden generar errores en el funcionamiento del sistema y, en casos extremos, comprometen la misión.

Si bien los ensayos de radiación permiten someter a las MCU a condiciones de exposición similares a las presentes en el entorno espacial, su implementación requiere infraestructura especializada y conlleva altos costos operativos. Esto dificulta que equipos con recursos limitados puedan realizar evaluaciones prácticas y obtener datos experimentales sobre el comportamiento de las MCU ante perturbaciones inducidas por radiación.

En este contexto, surge la necesidad de desarrollar una herramienta de FI accesible y adaptable a MCU, capaz de simular de manera controlada efectos *upset* en registros de configuración de periféricos y memoria, facilitando la evaluación precisa del comportamiento de los sistemas electrónicos de misiones espaciales ante fallas.

1.2 *Hipótesis*

La implementación de una herramienta de FI mediante técnicas OCD, adaptable a MCU y orientada a simular efectos SEU, permite reproducir y estudiar de manera controlada los efectos de la radiación sobre sistemas embebidos empleados en misiones espaciales, ofreciendo así una alternativa accesible que no requiere infraestructura especializada.

1.3 *Objetivo*

Desarrollar una herramienta de inyección de fallas que emplea la interfaz de depuración de un microcontrolador para simular la ocurrencia de efectos *upset*, con el propósito de evaluar los nodos sensibles de un sistema que estará expuesto a la radiación espacial.

1.3.1 *Objetivos específicos*

- Desarrollar una herramienta de FI empleando técnicas de OCD, capaz de simular efectos SEU en la memoria y los registros de configuración de periféricos de una MCU.
- Ejecutar campañas controladas de FI para observar el comportamiento del sistema y validar la efectividad de la herramienta.
- Diseñar una interfaz gráfica de usuario que permita ejecutar campañas automatizadas y recopilar los resultados.

1.4 *Alcances*

El presente trabajo contempla el desarrollo de una herramienta de FI implementada mediante técnicas de OCD para MCU, cuyo alcance se centra en la simulación de efectos SEU a través de la alteración controlada del contenido de memoria y los registros de configuración de periféricos, aplicada en una MCU representativa de las arquitecturas ARM Cortex-M, con el fin de validar su desempeño frente a la ocurrencia de fallas a bordo de nanosatélites tipo *CubeSat*.

1.5 *Justificación*

La herramienta de FI mediante OCD permite reproducir de manera controlada los efectos de la radiación en una MCU, ofreciendo una alternativa práctica frente a métodos costosos o dependientes de infraestructura especializada,

al mismo tiempo que facilita la experimentación directa en sistemas embebidos y contribuye al fortalecimiento de proyectos espaciales, promoviendo soluciones más robustas en entornos con recursos limitados.

2. METODOLOGÍA

La metodología corresponde a la forma en que se llevó a cabo el desarrollo del proyecto, definiendo el orden, los criterios y las actividades necesarias para alcanzar los objetivos planteados. En este capítulo se presenta la metodología que guió el desarrollo de la herramienta de FI, estructurada en cinco etapas fundamentales que abarcan desde la definición del alcance y los requisitos, hasta la documentación final y las propuestas de trabajo a futuro.

Cada etapa integra actividades específicas que permiten avanzar de manera sistemática en el diseño, implementación y validación de la herramienta de FI, asegurando que el desarrollo no solo cumpla con los requisitos definidos, sino que también permita verificar que la herramienta es capaz de simular efectos tipo *upset* utilizando técnicas de OCD.

Etapas 1: Planteamiento del proyecto

- 1. Identificación de la necesidad.** Se define el objetivo general del proyecto, así como la justificación para el desarrollo de la herramienta de FI.
- 2. Investigación preliminar.** Se realiza un estudio detallado de la literatura para documentar el marco teórico del proyecto, así como un análisis de las herramientas y técnicas existentes para comprender el estado del arte en FI.
- 3. Definición de problemática, objetivos y alcances.** Se realiza la presentación del problema, los objetivos específicos que guían el cumplimiento del objetivo general y el alcance que delimita el proyecto.

Etapas 2: Diseño de la herramienta

- 4. Definición de los requisitos del sistema.** Se establecen los requisitos funcionales y no funcionales de la herramienta de FI, incluyendo la capacidad de modificar el contenido de la memoria y los registros internos, compatibilidad con distintas MCU y los criterios relacionados con la configuración, ejecución y control de campañas de FI.
- 5. Diseño conceptual.** Se define la estructura lógica de la herramienta de FI, identificando sus componentes principales y describiendo cómo se relacionan entre sí.
- 6. Diseño a nivel sistema.** Se presenta la arquitectura general de la herramienta de FI desde el más alto nivel de abstracción, identificando sus principales módulos e interacciones.
- 7. Diseño de detalle.** Se profundiza en la organización interna de cada módulo, mostrando la composición general de la herramienta de FI.

Etapas 3: Implementación

- 8. Desarrollo de la herramienta.** Se implementa el inyector de fallas, garantizando la correcta modificación del contenido de la memoria y los registros internos para simular efectos *upset*.
- 9. Verificación inicial.** Se realiza una verificación preliminar del funcionamiento de la herramienta en un entorno controlado, asegurando que las alteraciones del contenido de la memoria y los registros internos se apliquen correctamente.
- 10. Ajustes y correcciones.** Con base en los resultados de la verificación, se realizan modificaciones y mejoras para optimizar la precisión y confiabilidad de la herramienta FI.

Etapas 4: Evaluación y pruebas

- 11. Pruebas controladas.** Se ejecutan pruebas controladas para evaluar la capacidad de la herramienta para simular efectos SEU en ubicaciones específicas de memoria y registros.
- 12. Análisis de resultados.** Se recopilan y analizan los datos obtenidos, observando el comportamiento del sistema objetivo ante las fallas inyectadas y evaluando la precisión de la simulación.
- 13. Ajustes y optimización.** Con base en el análisis de resultados, se realizan modificaciones para mejorar la precisión y eficiencia de la herramienta, optimizando los algoritmos de inyección y los mecanismos de configuración y ejecución de las campañas de FI.

Etapas 5: Documentación y conclusiones

- 14. Documentación técnica.** Se elabora la documentación completa de la herramienta, incluyendo su arquitectura, diseño conceptual, diseño a nivel sistema y diseño de detalle.
- 15. Evaluación final.** Se presentan las conclusiones sobre la efectividad de la herramienta, identificando logros alcanzados y limitaciones encontradas.
- 16. Líneas futuras de investigación.** Se proponen mejoras y líneas de desarrollo futuras, tales como la incorporación de nuevos modelos de falla, la adaptación de la herramienta de FI a otras MCU y la automatización avanzada de campañas de inyección.

SECCIÓN II
INVESTIGACIÓN PRELIMINAR

3. SATÉLITES ARTIFICIALES

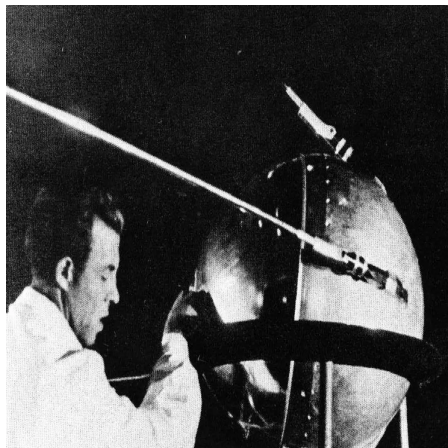


Figura 3.1. Primer satélite artificial *Sputnik 1* [2].

En el ámbito espacial, un satélite se define como cualquier cuerpo que orbita alrededor de otro de mayor masa, y aunque comúnmente este término se asocia con la Luna por ser el satélite natural de la Tierra, también existen los satélites artificiales, que son dispositivos creados por el ser humano con diversos fines. Un ejemplo destacado es la Estación Espacial Internacional (ISS, por sus siglas en inglés), la cual funciona como un laboratorio en órbita en el que los astronautas realizan investigaciones científicas.

Desde una perspectiva histórica, el primer satélite artificial desarrollado por el ser humano fue el *Sputnik 1*, lanzado por la Unión Soviética el 4 de octubre de 1957. Este consistía en una esfera de aluminio de aproximadamente 58 cm de diámetro y 83.6 kg de masa, equipada con cuatro antenas y dos transmisores que emitían señales de radio a intervalos regulares de tiempo. Su lanzamiento marcó el inicio de la era espacial y demostró la capacidad tecnológica de la Unión Soviética, lo que provocó una fuerte reacción en Estados Unidos y derivó en la creación, en 1958, de la Administración Nacional de Aeronáutica y del Espacio (NASA, por sus siglas en inglés).

A partir de este momento, se impulsó una rápida evolución en el desarrollo de satélites con distintos propósitos, entre los cuales destacó el *Telstar 1*, primer satélite de comunicaciones capaz de transmitir señales de televisión a larga distancia, lo que revolucionó la forma en que el mundo se mantenía conectado. Estos avances marcaron el inicio de una intensa competencia tecnológica que culminó en hitos históricos, como el envío del primer ser humano al espacio y la llegada del hombre a la Luna.

3.1 Subsistemas que integran los satélites [3]

El correcto funcionamiento de un satélite depende de la integración coordinada de varios subsistemas, cada uno con funciones específicas que aseguran una operación continua y confiable de la misión. Estos subsistemas conforman la arquitectura fundamental de los satélites artificiales, permitiendo desde la generación y distribución de energía hasta el control de actitud y órbita, la gestión de comunicaciones y el procesamiento de información.

En conjunto, los subsistemas integran un sistema interdependiente en donde el mal funcionamiento de uno puede comprometer el rendimiento general del satélite o incluso la continuidad de la misión, por lo que el diseño, la selección de componentes y la implementación de estrategias de tolerancia a fallas se vuelven aspectos críticos en el desarrollo de plataformas espaciales, especialmente cuando se emplean componentes COTS.

El análisis detallado de cada subsistema permite comprender su función dentro del sistema satelital, su interacción con los demás y las consideraciones técnicas necesarias para asegurar su funcionamiento en el entorno hostil del espacio.



Figura 3.2. Satélite artificial [4].

Subsistema de determinación y control de actitud (ADCS)

El subsistema de determinación y control de actitud (ADCS, por sus siglas en inglés) se encarga de estabilizar el satélite y orientarlo según las necesidades de la misión, incluso frente a perturbaciones externas que puedan afectarlo, utilizando una combinación de sensores para determinar su orientación y actuadores que aplican los pares necesarios para corregir desviaciones y mantener la orientación deseada.

Subsistema de telemetría, seguimiento y comando (TT&C)

El subsistema de telemetría, seguimiento y comando (TT&C, por sus siglas en inglés), también conocido como subsistema de comunicaciones, constituye el enlace principal entre el satélite y los sistemas en Tierra. Entre sus funciones se encuentran la transmisión hacia el centro de control de los datos de la misión y de información sobre el estado del satélite, así como la recepción de comandos enviados desde Tierra para supervisar y controlar las operaciones del satélite.

Subsistema de potencia eléctrica (EPS)

El subsistema de potencia eléctrica (EPS, por sus siglas en inglés) se encarga de generar, almacenar, distribuir y regular la energía requerida por los diferentes subsistemas del satélite, garantizando un suministro continuo y confiable durante toda la misión, incluso ante condiciones adversas o eclipses prolongados. Asimismo, integra elementos como paneles solares, baterías recargables, controladores de carga y unidades de distribución de potencia, de modo que su desempeño influye directamente en la supervivencia y operatividad del satélite.

Subsistema de control térmico (TCS)

El subsistema de control térmico (TCS, por sus siglas en inglés) tiene como objetivo mantener los componentes del satélite y la carga útil dentro de los márgenes de temperatura establecidos para su correcto funcionamiento. Para ello, emplea estrategias pasivas, entre ellas recubrimientos térmicos, aislamientos multicapa y radiadores, así como estrategias activas, que incluyen calentadores, refrigeradores o sistemas termoelectrónicos. Estas estrategias permiten mantener el desempeño y la supervivencia de los equipos a lo largo de las diferentes fases de la misión, desde el lanzamiento hasta la operación en órbita.

Subsistema de estructuras y mecanismos

El subsistema de estructuras y mecanismos proporciona el soporte mecánico que integra todos los demás subsistemas del satélite, conecta la nave al vehículo de lanzamiento y permite su separación segura en el momento indicado. Para ello, debe cumplir requisitos de resistencia y rigidez que consideran las cargas experimentadas durante el lanzamiento, la inserción orbital y las operaciones en el espacio. Asimismo, el diseño considera los modos de falla estructural y las oportunidades de optimización de masa, buscando un equilibrio entre robustez, masa y funcionalidad para asegurar la integridad mecánica del sistema a lo largo de toda la misión.

Subsistema de guía y navegación (G&N)

El subsistema de guía y navegación (G&N, por sus siglas en inglés) tiene el objetivo de determinar y controlar la posición y velocidad del satélite en función del tiempo, garantizando que su trayectoria cumpla con los objetivos de la misión. La navegación consiste en estimar con precisión la órbita mediante mediciones y cálculos de posición y velocidad, mientras que la guía o control orbital ejecuta las maniobras necesarias para mantener o modificar dicha órbita según los objetivos operativos. Este subsistema trabaja de manera complementaria con el ADCS: el ADCS controla la orientación respecto al centro de masa, mientras que el G&N gestiona la posición en el espacio, de modo que ambos colaboran en la ejecución de maniobras precisas y en el posicionamiento del satélite durante todas las fases de la misión.

Computadora de a bordo (OBC)

La computadora de a bordo (OBC, por sus siglas en inglés) actúa como el cerebro del satélite, coordinando y controlando el funcionamiento de los diferentes subsistemas, ejecutando las rutinas de control, procesando los comandos enviados desde Tierra y asegurando que la misión se desarrolle conforme a lo programado. Asimismo, se encarga de funciones de comando y manejo de datos, incluyendo la recepción, validación y distribución de comandos hacia los demás subsistemas, así como la recopilación, procesamiento y gestión de los datos de telemetría y de la misión.

La OBC suele estar basada en MCU o procesadores de alta confiabilidad, seleccionados por su tolerancia a la radiación y su capacidad de operar en entornos espaciales hostiles. No obstante, en misiones con recursos limitados es común emplear componentes COTS, implementando en estos casos arquitecturas redundantes y estrategias de mitigación de fallas para garantizar la continuidad operativa del satélite ante posibles errores y fallos en los dispositivos electrónicos.

Carga útil

La carga útil constituye el elemento central de la misión, ya que define el propósito del satélite y los resultados que se espera obtener, es decir, representa el conjunto de instrumentos, sensores o equipos diseñados específicamente para cumplir los objetivos científicos, tecnológicos o de comunicación de la misión. Dependiendo del tipo de satélite, la carga útil puede incluir:

- Cámaras ópticas o multispectrales, empleadas en satélites de observación terrestre.
- Transpondedores o antenas, utilizadas en satélites de comunicaciones.
- Sensores de radiación, detectores de partículas o magnetómetros, incluidos en misiones científicas.
- Módulos experimentales, comunes en *CubeSat* o misiones tecnológicas.

El diseño de la carga útil debe integrarse cuidadosamente con los demás subsistemas, considerando factores como el consumo de potencia, la transferencia térmica, las interferencias electromagnéticas, la orientación requerida y la capacidad de transmisión de datos. Su correcto funcionamiento depende directamente de la OBC, que gestiona las rutinas de adquisición, almacenamiento y envío de información hacia la estación terrena.

En el caso de los *CubeSat*, la carga útil suele incluir módulos de experimentación tecnológica como sensores de radiación, MCU de alta confiabilidad o sistemas de prueba de tolerancia a fallas, representando este tipo de misiones una plataforma ideal para validar nuevas arquitecturas o estrategias de mitigación frente a los efectos de la radiación espacial, aspecto particularmente relevante en el desarrollo de herramientas de FI.

3.2 Clasificación de los satélites artificiales

A lo largo del tiempo, los satélites artificiales han experimentado una notable diversificación en sus funciones y características, con el fin de cumplir una amplia variedad de tareas fundamentales en distintas áreas. De acuerdo con [5], se pueden identificar cuatro principales áreas de aplicación: comunicaciones, teledetección, navegación y ciencia y exploración.

Sin embargo, esta no es la única forma de clasificar los satélites, ya que también pueden agruparse de acuerdo con otros criterios, como su masa, la órbita en la que operan o su diseño estructural. A continuación, se presentan dos clasificaciones relevantes: según su área de aplicación y según su masa.

3.2.1 Según su área de aplicación

Una de las formas más comunes de clasificar los satélites es según su área de aplicación, ya que cada tipo se diseña para cumplir objetivos específicos que responden a necesidades científicas, comerciales, militares o de servicios globales.

Satélites de comunicaciones

Los satélites de comunicaciones pueden operar en diferentes tipos de órbita, entre las que se encuentran la órbita geoestacionaria (GEO, por sus siglas en inglés) y la órbita terrestre baja (LEO, por sus siglas en inglés). Estos satélites proporcionan servicios de comunicación como la transmisión de señales de radio, televisión, telefonía e internet. En algunos casos, un único satélite puede proporcionar cobertura sobre una determinada región, mientras que las constelaciones de satélites permiten ampliar la cobertura mediante múltiples satélites distribuidos en una misma órbita o en diferentes planos orbitales.



Figura 3.3. Satélite de comunicaciones [6].

De acuerdo con [7], estos satélites actúan como repetidores: reciben señales de radiofrecuencia enviadas desde una estación terrestre, las amplifican y las retransmiten de vuelta a la Tierra con una frecuencia distinta. Este proceso es posible gracias a un dispositivo conocido como transpondedor.

Satélites de teledetección

Los satélites de teledetección se diseñan para obtener y recopilar información sobre la superficie terrestre. Están equipados con sensores de alta precisión capaces de captar emisiones electromagnéticas en diversas bandas espectrales, lo que permite registrar imágenes y datos sobre múltiples características del entorno y fenómenos naturales, como la erosión, los incendios forestales o la contaminación ambiental.

Satélites de navegación

Este tipo de satélites surgió con fines militares, al permitir la guía precisa de misiles, submarinos o tropas. Con el tiempo, su uso se extendió al ámbito civil, permitiendo determinar la ubicación en cualquier lugar del planeta mediante señales de radio transmitidas por los satélites y captadas por los receptores, que calculan la posición exacta a partir de esta información.

Un ejemplo es el sistema de posicionamiento global (GPS, por sus siglas en inglés), el cual utiliza una constelación de satélites para proporcionar información de ubicación precisa en la superficie terrestre, resultando de gran utilidad en sectores como el transporte, la geolocalización y la navegación personal.

Satélites de ciencia y exploración

Los satélites de ciencia y exploración tienen como objetivo principal ampliar nuestro conocimiento del universo, el sistema solar y la Tierra. A diferencia de los satélites comerciales o militares, estos se diseñan específicamente para llevar a cabo investigaciones científicas, observaciones detalladas y misiones de exploración planetaria.

Estos satélites no solo han permitido responder muchas preguntas científicas, sino que también han generado nuevas interrogantes que impulsan la exploración futura, contribuyendo a que la humanidad comprenda mejor su lugar en el universo y sentando las bases para próximas misiones espaciales.

3.2.2 Según su masa

Como se mencionó, la evolución tecnológica ha permitido que los satélites se diversifiquen en distintas áreas de aplicación. No obstante, este avance ha estado acompañado de un incremento en su tamaño y masa, generando desafíos importantes al requerir infraestructura especializada y grandes inversiones. Por ello, durante mucho tiempo, su desarrollo estuvo limitado a un reducido número de naciones con alto poder económico, tecnológico, militar y cultural.

Gracias a los avances en tecnología de integración a muy gran escala (VLSI, por sus siglas en inglés) y al uso de componentes COTS, fue posible desarrollar una nueva clasificación de satélites: los satélites pequeños, que requieren menos recursos, menor tiempo de desarrollo y presentan menor complejidad de construcción, lo que los hace más accesibles para empresas, organizaciones e incluso países con recursos limitados. En la tabla 3.1 se presenta una clasificación de los satélites según su masa y el costo aproximado de fabricación.

Tabla 3.1. Clasificación de los satélites por masa y costo (adaptado de [8]).

Clase	Masa [kg]	Costo [MUSD]
Satélite convencional grande	> 1000	> 135.31
Satélite convencional pequeño	500–1000	33.83–135.31
Minisatélite	100–500	9.47–13.31
Microsatélite	10–100	1.35–9.47
Nanosatélite	1–10	0.14–1.35
Picosatélite	< 1	< 0.14

3.3 Estándar CubeSat

Los *CubeSat* son nanosatélites con un diseño estandarizado que surgieron como herramienta educativa para universidades. Cuentan con un formato modular y compacto basado inicialmente en unidades (U) de $10 \times 10 \times 10$ cm, permitiendo desarrollar, lanzar y operar un satélite con costos y tiempos de desarrollo significativamente menores en comparación con los satélites convencionales.

Con el tiempo, el concepto se ha expandido para incluir múltiples configuraciones, como 2U, 3U y 6U, como se muestra en la figura 3.5. Actualmente, es común ensamblar múltiples unidades para formar satélites de mayor tamaño, lo que brinda mayor flexibilidad en sus aplicaciones, y aunque suelen clasificarse como nanosatélites, aquellos que superan el formato de 6U se consideran microsatélites, empleándose en misiones cada vez más complejas.

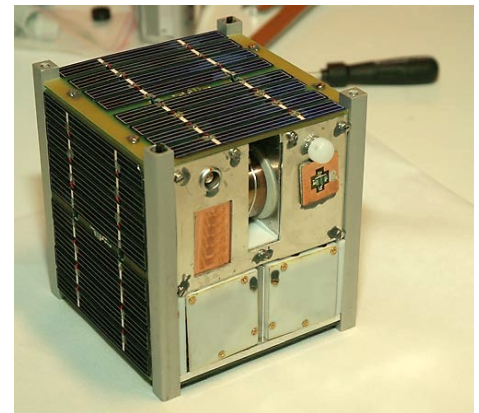


Figura 3.4. *CubeSat* [9].

Los *CubeSat* han democratizado el acceso al espacio, consolidándose como plataformas versátiles para la investigación, la demostración tecnológica y la formación de constelaciones satelitales a costos alcanzables, fomentando que universidades, empresas y agencias espaciales de menor escala participen activamente en misiones y experimentos que antes estaban al alcance solo de grandes programas espaciales.

De acuerdo con [8], el crecimiento global de los *CubeSat* ha sido impulsado por cinco factores clave:

- **Accesibilidad:** Diseñados originalmente como herramienta educativa, los *CubeSat* permitieron a universidades y pequeños laboratorios afrontar los costos y tiempos de desarrollo de satélites gracias a componentes asequibles y diseños simplificados.
- **Estandarización:** Las especificaciones estructurales, los mecanismos de despliegue y la posibilidad de apilar unidades favorecieron que ciertas soluciones de ingeniería se consolidaran como estándares de la industria.
- **Emprendimiento:** Su baja inversión inicial y el potencial de retornos significativos convirtieron a los *CubeSat* en plataformas atractivas para desarrolladores, empresas y clientes diversos.
- **Tecnología:** Los avances en miniaturización, autonomía distribuida y desempeño de sistemas han incrementado el número de componentes COTS con herencia de vuelo, facilitando el desarrollo de misiones más complejas al permitir integrar mayores capacidades de procesamiento y funcionalidad en plataformas de menor volumen y costo.

- **Comunidad:** La experiencia, tecnología e infraestructura compartidas han promovido la aparición de empresas y servicios especializados, ampliando las oportunidades de acceso al espacio incluso para actores con recursos limitados.

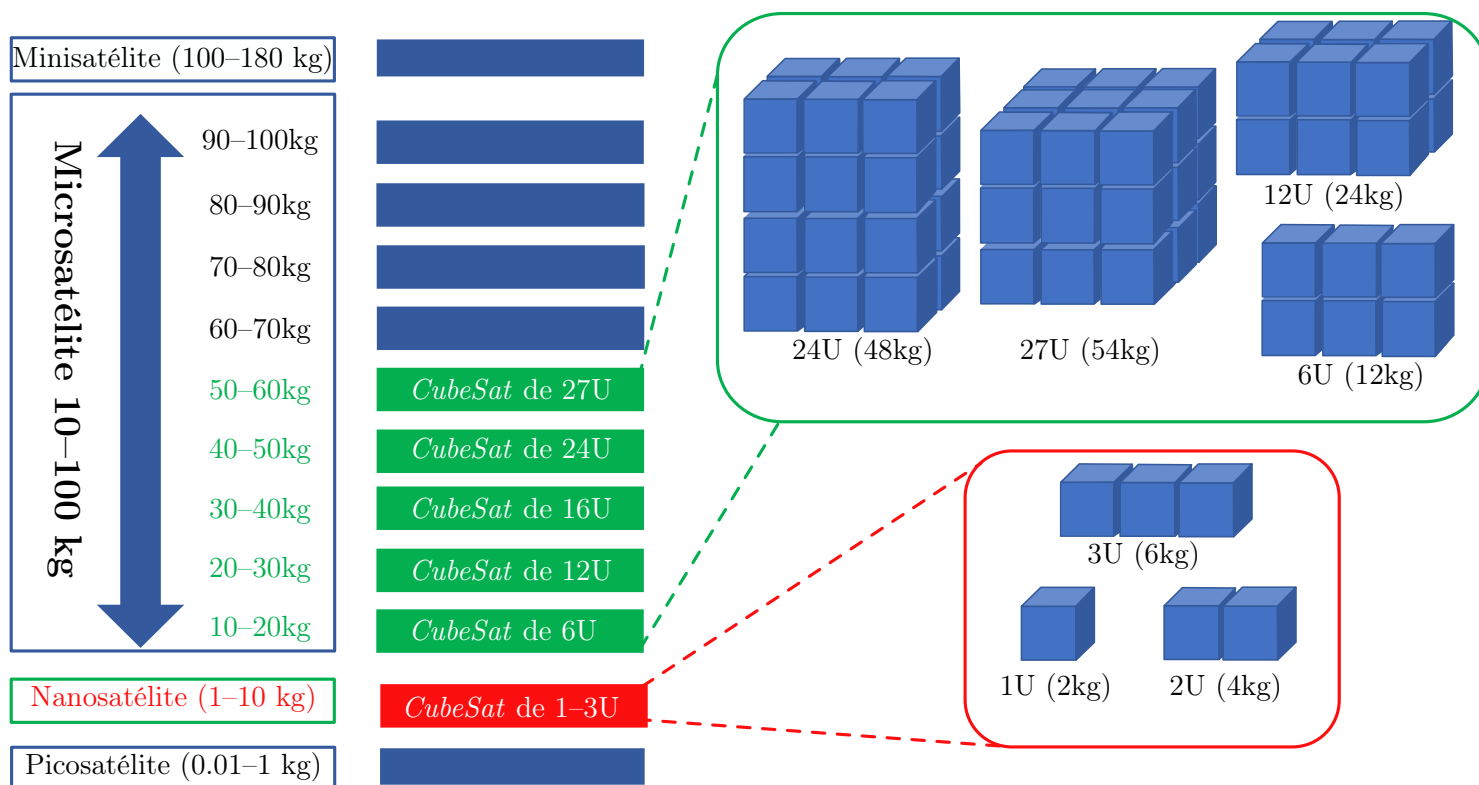


Figura 3.5. Tamaños estandarizados de *CubeSat* (adaptado de [10]).

Aplicaciones de los *CubeSat* [8]

Los *CubeSat* se han convertido en plataformas versátiles capaces de cumplir diversos objetivos de misión:

- **Instrumentos espaciales alternativos:** Amplían y complementan funciones tradicionales de satélites, como telecomunicaciones y observación terrestre.
- **Laboratorios y experimentos espaciales:** Permiten que disciplinas como ciencias de la vida, ciencia de materiales y radiología realicen experimentos directamente en el espacio.
- **Cargas útiles fuera de la órbita terrestre:** Facilitan mediciones adicionales o el soporte a satélites principales, así como la colocación de instrumentos en órbita.
- **Megaconstelaciones a precios alcanzables:** La estandarización permite la producción en masa, mejorando la cobertura, la redundancia y el tiempo de revisita de constelaciones de satélites.

4. MEDIO AMBIENTE ESPACIAL

Con el aumento de la presencia humana en el espacio, a través de satélites artificiales, sondas, estaciones orbitales y misiones tripuladas, resulta esencial comprender los factores que afectan la operación y la supervivencia de estos sistemas en dicho entorno.

El entorno espacial presenta condiciones muy diferentes a las de la superficie terrestre, en las cuales las naves y los equipos en órbita deben soportar un medio ambiente hostil, caracterizado por el vacío, el plasma, la radiación de alta energía, los micrometeoritos y los desechos espaciales. Estos factores pueden generar efectos severos, como se muestra en la tabla 4.1, por lo que es indispensable considerarlos desde las etapas de diseño, desarrollo y operación, con el fin de garantizar un funcionamiento confiable y duradero de los sistemas espaciales [11].

La comprensión detallada del medio ambiente espacial permite no solo optimizar el diseño y la durabilidad de los sistemas espaciales, sino también definir estrategias de diseño tolerante a fallas que incrementen la seguridad y la confiabilidad de las misiones.

Tabla 4.1. Efectos del medio ambiente espacial (adaptado de [11]).

Entorno	Efectos
Vacío	- Diferencias de presión. - Degradación por radiación ultravioleta solar. - Contaminación.
Neutro	- Efectos mecánicos: Arrastre aerodinámico, desgaste físico (<i>sputtering</i>). - Efectos químicos: Ataque por oxígeno atómico, brillo de la nave espacial.
Plasma	- Carga de la nave espacial: Cambio en el potencial eléctrico. - Descarga electrostática: Ruptura dieléctrica, descarga electrostática. - Desgaste físico aumentado. - Atracción de contaminantes.
Radiación	- Efectos por dosis total: Degradación de la electrónica, riesgo para la seguridad de la tripulación. - Efectos de evento único: <i>Upsets, latchup, burnout</i>.
Micrometeoritos/ basura espacial	- Daños por impactos a hipervelocidad.

4.1 Vacío

El entorno de vacío se define por la ausencia de atmósfera, resultado de la disminución progresiva de la densidad del aire con la altitud, como se muestra en la figura 4.1, en donde la presión atmosférica alcanza valores tan bajos que se considera un vacío. Esta condición representa un desafío para los sistemas espaciales, ya que algunos materiales contienen pequeñas burbujas de gas atrapadas bajo condiciones terrestres, las cuales se liberan gradualmente al exponerse al vacío en un proceso conocido como desgasificación, considerado una de las principales fuentes de contaminación molecular en sistemas espaciales. Los compuestos liberados durante este proceso pueden depositarse sobre superficies sensibles y formar delgadas películas contaminantes que afectan el funcionamiento de dispositivos como sensores ópticos y componentes electrónicos [11].

Además, el vacío genera otros fenómenos físicos que no se presentan en la Tierra, como la soldadura en frío, la cual ocurre cuando dos superficies metálicas entran en contacto directo en ausencia de aire, provocando que se adhieran entre sí como si estuvieran soldadas [5]. Asimismo, la ausencia de un fluido impide la transferencia de calor por convección, por lo que la disipación térmica en el espacio depende principalmente de mecanismos como la conducción y la radiación [13].

Finalmente, la ausencia de atmósfera implica que la radiación ultravioleta no se absorbe ni se bloquea, dejando los sistemas espaciales directamente expuestos y afectando las propiedades físicas y químicas de los materiales. Esto evidencia la necesidad de seleccionar y diseñar cuidadosamente los componentes y materiales empleados en satélites y otras plataformas espaciales para garantizar su correcto funcionamiento.

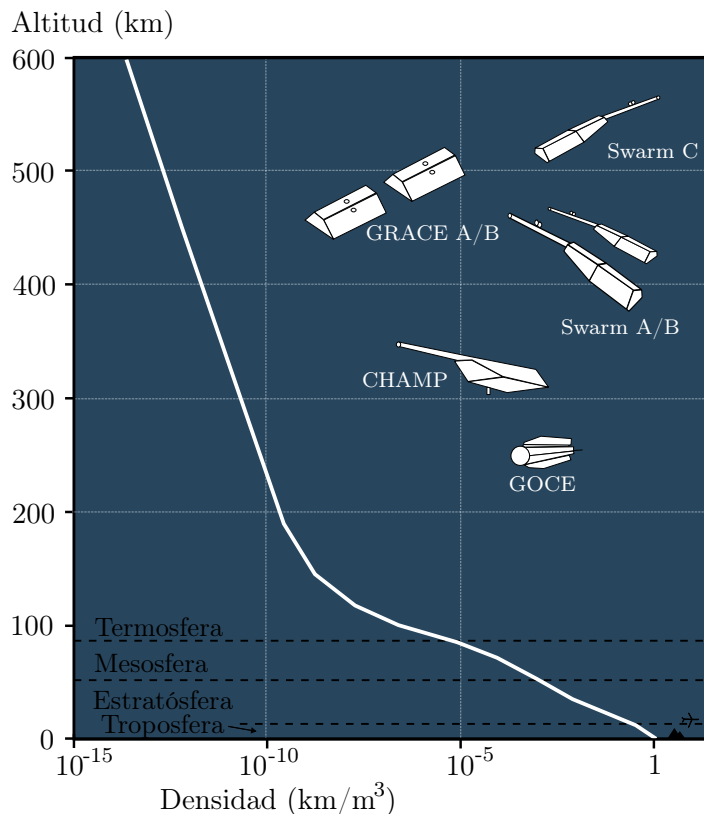


Figura 4.1. Densidad vs altitud (adaptado de [12]).

4.2 Entorno neutro

El entorno neutro se refiere a la atmósfera neutra presente en las altitudes correspondientes a las órbitas LEO, cuyo estudio resulta fundamental dentro del análisis del medio ambiente espacial, ya que puede afectar a los satélites tanto por efectos mecánicos como químicos.

Antes de detallar estos efectos, es importante comprender el comportamiento de la atmósfera. Como se mencionó, al aumentar la altitud, tanto la presión como la densidad atmosférica disminuyen. Sin embargo, aunque la densidad sea baja en las regiones superiores, sus efectos no son despreciables, pues aún pueden generar resistencia sobre los vehículos espaciales. Esta resistencia, conocida como arrastre atmosférico, depende de la densidad del aire, la velocidad relativa (definida como la velocidad de la nave con respecto al movimiento de las partículas atmosféricas), la forma, el tamaño y la orientación de la nave con respecto al flujo atmosférico.

La figura 4.2 muestra cómo el arrastre atmosférico influye en la órbita de un satélite; aunque la densidad atmosférica en LEO es baja, las partículas neutras generan suficiente fricción para alterar su trayectoria con el tiempo. Este fenómeno se evidencia en la curva del satélite afectado por arrastre, que refleja la pérdida gradual de altitud y energía orbital, limitando la vida útil de la misión y requiriendo maniobras periódicas de corrección.

Además del arrastre, la interacción de la atmósfera con la nave a altas velocidades orbitales provoca un efecto conocido como *sputtering*, en el cual átomos y moléculas

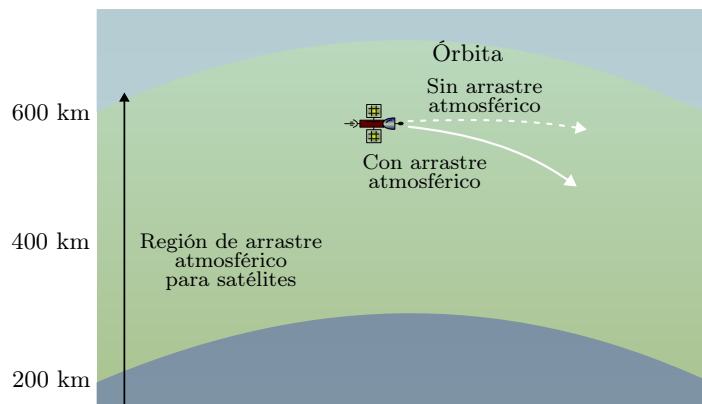


Figura 4.2. Arrastre atmosférico (adaptado de [14]).

neutras transfieren suficiente energía cinética para expulsar átomos de la superficie del material, causando una erosión progresiva de las superficies y componentes estructurales.

Por otra parte, en las capas superiores de la atmósfera, las moléculas de oxígeno son escasas debido a la acción de la radiación solar y de partículas cargadas, que las disocian y generan oxígeno atómico, altamente reactivo, capaz de degradar las superficies de los materiales expuestos, debilitando componentes estructurales, alterando propiedades térmicas y afectando sensores sensibles.

Asimismo, otro efecto derivado de la interacción del oxígeno atómico con las superficies de las naves es la luminiscencia, originada por reacciones químicas entre este elemento y ciertos materiales, como se muestra en la figura 4.3. Este fenómeno puede interferir con los instrumentos ópticos a bordo y reducir la calidad de las observaciones científicas [11].

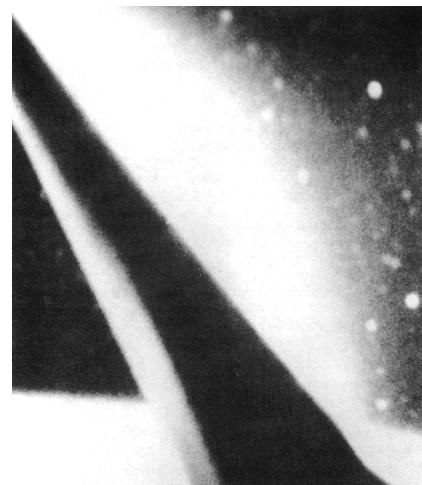


Figura 4.3. Resplandor ocasionado por el oxígeno atómico en una nave espacial [11].

4.3 Plasma

El plasma corresponde a un gas ionizado compuesto por iones y electrones libres, formando un conjunto de partículas cargadas eléctricamente capaces de moverse de manera independiente. Este estado se encuentra ampliamente distribuido en el entorno espacial y sus propiedades, como la energía y la densidad de las partículas, pueden variar considerablemente dependiendo de la altitud, la latitud, la intensidad del campo magnético y la actividad solar [15].

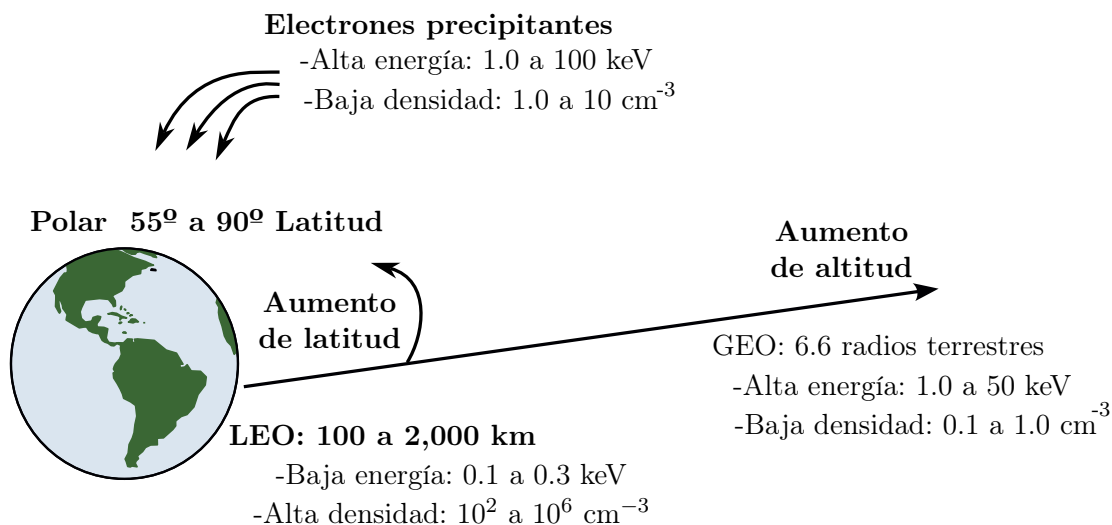


Figura 4.4. Propiedades del plasma espacial natural en diferentes regiones del entorno terrestre (adaptado de [15]).

Como se muestra en la figura 4.4, las características del plasma varían de acuerdo con la región del entorno terrestre. En LEO, predominan partículas de baja energía y alta densidad, mientras que a mayores altitudes, como en GEO, la densidad disminuye y la energía de las partículas aumenta. Asimismo, en las regiones polares pueden presentarse electrones precipitantes de alta energía. Estas variaciones determinan la forma en que el plasma interactúa con los sistemas espaciales [15].

Para las naves, el plasma representa un desafío importante, ya que al encontrarse en un entorno cargado eléctricamente, diferentes partes de la nave (en particular aquellas con distinta conductividad, como las superficies metálicas conductoras y los materiales dieléctricos aislantes) pueden acumular cargas a potenciales distintos. Esta acumulación puede generar descargas electrostáticas capaces de afectar sistemas electrónicos sensibles, provocar fallos en instrumentos e incluso dañar permanentemente algunos subsistemas [11]. Además, las condiciones de plasma pueden interferir con las comunicaciones y el control de actitud, al distorsionar las señales electromagnéticas o generar acumulación de cargas sobre sensores y actuadores.

4.4 *Micrometeoritos y basura espacial*

El espacio no está compuesto únicamente por cuerpos celestes; en las últimas décadas, el ser humano ha colocado numerosos satélites en órbita, lo que ha generado un problema creciente: la acumulación de basura espacial.

De acuerdo con [16], la basura espacial se define como cualquier objeto fabricado por el ser humano que ha perdido su funcionalidad o utilidad, incluyendo satélites fuera de operación, etapas de propulsión desechadas, fragmentos resultantes de colisiones en órbita y otros desechos generados durante misiones. Además, es importante considerar el polvo natural proveniente de la desintegración de cometas, asteroides y meteoritos. Estas partículas, conocidas como micrometeoritos, representan otro factor de riesgo en el entorno espacial.

Tanto los micrometeoritos como los desechos artificiales representan riesgos importantes para las misiones espaciales, ya que incluso impactos de objetos muy pequeños pueden dañar severamente la superficie de naves y satélites. Además de los impactos directos, estos pueden generar la erosión progresiva de los materiales, la liberación de partículas secundarias y la contaminación de sensores o superficies ópticas críticas para el funcionamiento de los sistemas espaciales.



Figura 4.5. Basura espacial [17].

5. RADIACIÓN ESPACIAL

Se entiende por radiación a cualquier forma de energía que se propaga a través del espacio o de un material, ya sea en forma de ondas electromagnéticas o partículas energéticas, y puede originarse de manera natural, como los rayos cósmicos galácticos (GCR, por sus siglas en inglés), o de forma artificial, como en los aceleradores de partículas.

La radiación se clasifica en dos tipos principales: ionizante (alta energía) y no ionizante (baja energía). La radiación ionizante posee suficiente energía para liberar electrones de los enlaces atómicos y generar átomos cargados positivamente (iones), mientras que la radiación no ionizante no cuenta con la energía necesaria para producir este proceso en los materiales que atraviesa.

De acuerdo con el Instituto Nacional de Investigaciones Nucleares [18], en el espacio está presente la radiación cósmica de fondo, una radiación electromagnética que llena el universo y constituye el remanente del *Big Bang*. En los primeros instantes del universo, las condiciones eran tan calientes y densas que los electrones y protones existían en forma de un plasma opaco, el cual impedía el libre desplazamiento de la luz; al expandirse y enfriarse, los electrones se combinaron con los protones para formar átomos neutros, permitiendo que la radiación viajara libremente. Este fenómeno es conocido actualmente como radiación cósmica de fondo de microondas (CMB, por sus siglas en inglés), descubierta en 1965 por Arno Penzias y Robert Wilson, y es detectable de manera uniforme en todas las direcciones del espacio.

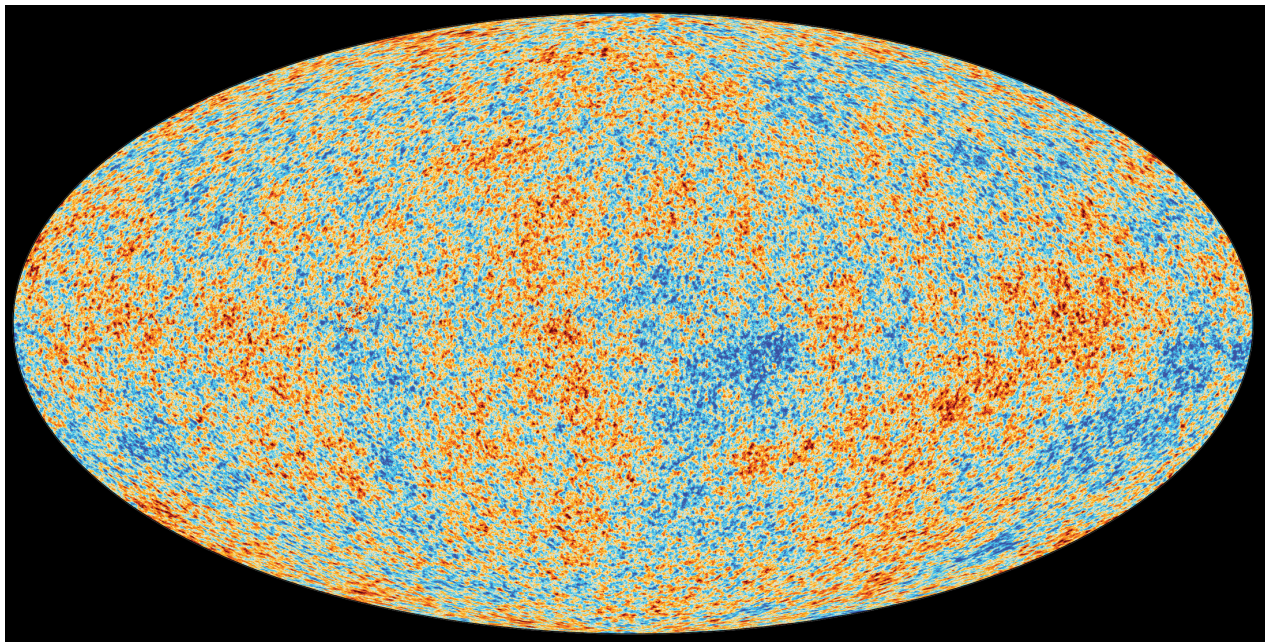


Figura 5.1. Fondo cósmico de microondas [19].

Por otra parte, la radiación cósmica también incluye partículas energéticas provenientes del espacio exterior, emitidas por el Sol, otras estrellas, supernovas y diversas fuentes astrofísicas, las cuales pueden clasificarse como de origen primario, cuando se emiten directamente desde una fuente cósmica; o de origen secundario, cuando se generan tras la interacción de las partículas primarias con la materia del espacio o con la atmósfera terrestre.

Dentro de este contexto, la radiación espacial se entiende como un término amplio que engloba las diversas fuentes de radiación ionizante presentes en el espacio. En el entorno cercano a la Tierra, esta se compone principalmente de dos fuentes: las partículas expulsadas durante eventos solares y los GCR (formados por protones e iones pesados de alta energía procedentes del exterior del sistema solar). Al interactuar con el campo magnético terrestre, parte de estas

partículas puede quedar atrapada en regiones conocidas como cinturones de Van Allen. De esta manera, en el entorno cercano a la Tierra pueden distinguirse partículas en tránsito, de origen solar o galáctico, y partículas atrapadas por el campo geomagnético.

5.1 Fuentes y componentes del entorno de radiación espacial

Como se mencionó, en el entorno espacial cercano a la Tierra, la radiación ionizante proviene principalmente de dos fuentes naturales que determinan las condiciones del ambiente radiativo: las partículas energéticas solares (SEP, por sus siglas en inglés) y los GCR. Cada una de estas fuentes presenta características particulares en cuanto a su origen, composición y niveles de energía, las cuales influyen directamente en el nivel de radiación al que están expuestos los sistemas y dispositivos espaciales.

Por lo tanto, comprender la naturaleza y el comportamiento de estas fuentes, así como la presencia y el comportamiento de las partículas atrapadas por el campo geomagnético, resulta fundamental para el diseño de estrategias de mitigación, la planeación de misiones y la confiabilidad de los sistemas electrónicos a bordo. A continuación, se describen las características de las SEP y los GCR, así como la radiación atrapada en los cinturones de Van Allen.

5.1.1 Partículas energéticas solares (SEP)

El Sol, además de emitir radiación electromagnética en un amplio espectro, libera continuamente partículas cargadas que constituyen el viento solar. Su actividad magnética presenta variaciones a lo largo del ciclo solar y puede dar lugar a fenómenos energéticos, como fulguraciones solares y eyecciones de masa coronal (CME, por sus siglas en inglés). Durante estos eventos, algunas partículas pueden ser aceleradas hasta alcanzar energías considerablemente mayores que las asociadas al viento solar, dando lugar a los eventos de SEP. Estas partículas están compuestas principalmente por protones y partículas alfa, además de una menor proporción de iones más pesados, y constituyen una fuente importante de radiación ionizante capaz de afectar a los sistemas electrónicos.

Los SEP se clasifican generalmente en dos categorías, de acuerdo con su mecanismo de aceleración y su duración:

- **Eventos impulsivos:** Se originan durante fenómenos de reconexión magnética en líneas de campo abiertas (*jets*) en la corona solar.
- **Eventos graduales:** Se producen durante ondas de choque impulsadas por las CME y suelen tener una duración mayor que los eventos impulsivos.

Estos mecanismos de aceleración se ilustran en la figura 5.2, en donde se esquematizan las configuraciones magnéticas asociadas a ambos tipos de eventos: en los impulsivos, las partículas son aceleradas a lo largo de líneas de campo abiertas mediante procesos de reconexión magnética, mientras que en los graduales son energizadas por los frentes de choque generados por las CME.

A pesar de la elevada energía que pueden alcanzar algunas SEP, la Tierra dispone de un escudo natural proporcionado por su campo magnético. La interacción de las partículas cargadas con este campo se describe mediante la fuerza de Lorentz, expresada como:

$$\mathbf{F} = q(\mathbf{v} \times \mathbf{B}) \quad (5.1)$$

en donde q es la carga de la partícula, $\mathbf{v}$ su velocidad y $\mathbf{B}$ el campo magnético. Esta fuerza desvía muchas de las partículas energéticas, reduciendo su impacto directo sobre la superficie terrestre. No obstante, la efectividad de esta protección depende de la latitud: en las regiones ecuatoriales, las partículas inciden perpendicularmente en las líneas del campo y son desviadas con mayor eficacia, mientras que en latitudes altas la protección disminuye, ya que las partículas inciden casi paralelas en las líneas de campo.

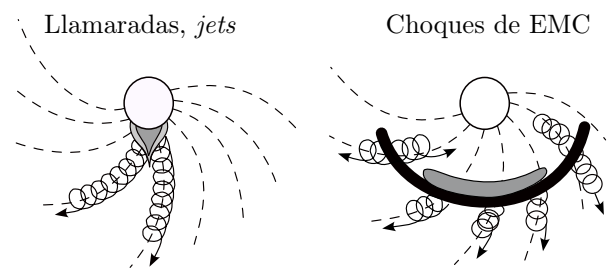


Figura 5.2. Mecanismos de aceleración de SEP (adaptado de [20]).

Este fenómeno explica por qué, aunque las SEP representan un riesgo significativo para las misiones espaciales en órbita y fuera del entorno terrestre, la superficie del planeta se encuentra relativamente protegida frente a la mayor parte de estas partículas. Este efecto se ilustra en la figura 5.3, en donde se muestra cómo el campo magnético terrestre interactúa con el flujo de partículas solares y con los GCR (que se explicarán en la siguiente sección), actuando en conjunto como una barrera natural que desvía gran parte de las partículas cargadas.

5.1.2 Rayos cósmicos galácticos (GCR)

Los GCR son núcleos de iones altamente energéticos que constituyen una fuente relevante de radiación ionizante en el espacio cercano a la Tierra, en donde los protones representan la mayor parte del flujo (alrededor del 83–85 %), seguidos por núcleos de helio (13–14 %) y electrones (aproximadamente 3 %), mientras que los iones pesados constituyen solo cerca del 1 % [11].

Estos rayos se caracterizan por presentar una distribución casi isotrópica, ya que, a escalas galácticas, llegan desde todas las direcciones con una distribución prácticamente uniforme; su flujo es relativamente bajo, de aproximadamente 4 partículas/cm²s, y su energía varía entre 10^8 y 10^{19} eV¹, alcanzando en algunos iones hasta 3×10^{20} eV.

El origen exacto de los GCR aún no se comprende por completo; una fracción significativa se genera dentro de la Vía Láctea, mientras que otra parte tiene un origen extragaláctico. Al aproximarse a la Tierra, sus trayectorias se ven modificadas por la interacción con el campo magnético terrestre y con el medio interplanetario, lo que altera su propagación y reduce la intensidad que finalmente alcanza a la superficie.

La intensidad de los GCR varía a lo largo del ciclo solar, alcanzando su máximo durante los periodos de mínimo solar, en donde el campo magnético del Sol es más débil y permite el ingreso de un mayor número de partículas al entorno terrestre. De esta forma, la actividad solar controla directamente la dosis de radiación proveniente de los GCR recibida por los sistemas electrónicos en misiones espaciales.

5.1.3 Cinturones de Van Allen

Los cinturones de radiación de Van Allen, descubiertos en 1958 por el astrofísico James Van Allen mediante los satélites *Explorer 1* y *Explorer 3*, son dos regiones toroidales formadas por partículas de alta energía atrapadas por el campo magnético terrestre. La estructura general de estos cinturones y su relación con las líneas del campo magnético se muestra en la figura 5.4.

El cinturón interno está compuesto principalmente por protones de alta energía, generados cuando los GCR interactúan con los átomos de la atmósfera superior, mientras que el cinturón externo contiene sobre todo electrones de alta energía y protones de menor energía, cuya intensidad varía notablemente con la actividad solar, especialmente durante tormentas geomagnéticas y eventos asociados a CME.

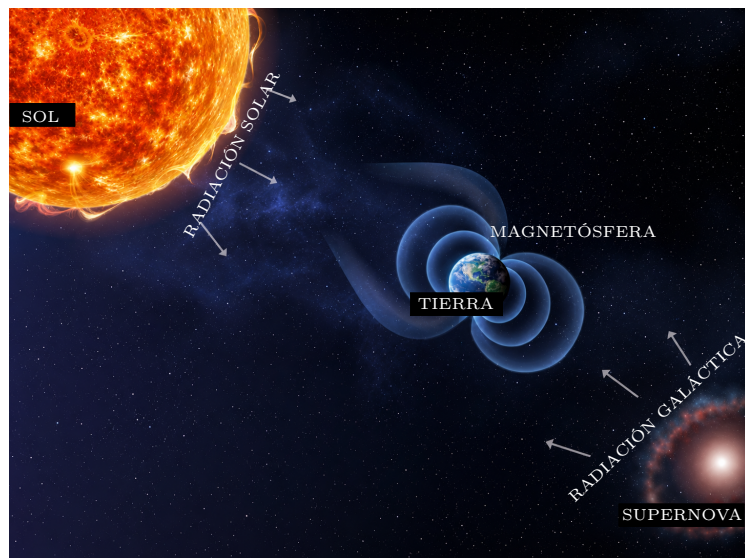


Figura 5.3. Interacción del campo magnético terrestre con partículas energéticas solares y rayos cósmicos galácticos (adaptado de [21]).

¹Unidad de energía utilizada en física de partículas, equivalente a la energía adquirida por un electrón al atravesar una diferencia de potencial de un voltio, en donde $1 \text{ eV} = 1,602 \times 10^{-19} \text{ J}$ [22].

Estos cinturones se extienden aproximadamente entre 1,000 y 60,000 km sobre la superficie terrestre, alcanzando incluso regiones más allá de la órbita baja. Aunque la magnetósfera actúa como un escudo natural que protege la vida y la infraestructura tecnológica en la Tierra, los cinturones de Van Allen representan un entorno altamente hostil para las misiones espaciales, ya que las partículas atrapadas pueden provocar daños acumulativos en los materiales, degradar el rendimiento de los paneles solares y ocasionar fallas asociadas a SEE en componentes electrónicos sensibles.

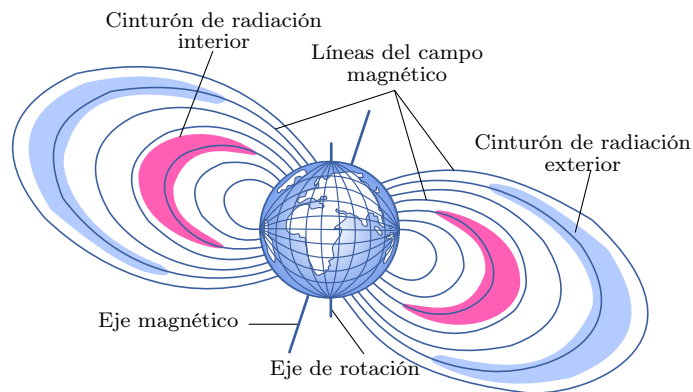


Figura 5.4. Anillos de Van Allen (adaptado de [23]).

Anomalía del Atlántico Sur (SAA)

Sobre Sudamérica y el océano Atlántico Sur se localiza una región inusualmente débil del campo magnético terrestre, conocida como la Anomalía del Atlántico Sur (SAA, por sus siglas en inglés), mostrada en la figura 5.5, en la cual la intensidad del campo es significativamente menor, lo que permite que las partículas energéticas atrapadas en los cinturones de radiación penetren a altitudes más bajas y se acerquen considerablemente a la superficie terrestre.

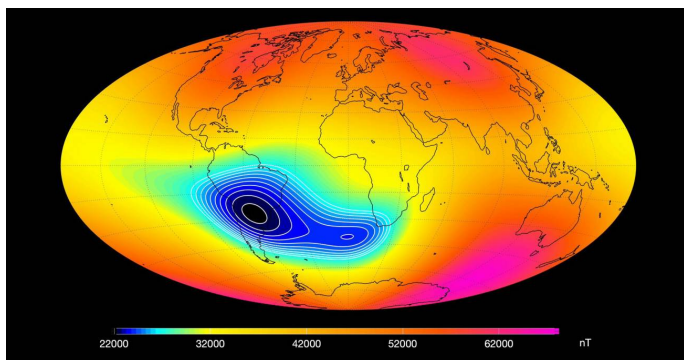


Figura 5.5. Anomalía del Atlántico Sur [24].

Aunque la SAA no genera efectos perceptibles en la vida cotidiana ni representa un riesgo directo para la población en la superficie, su extensión e intensidad han despertado un notable interés científico. Observaciones recientes indican que esta región se desplaza gradualmente hacia el oeste y comienza a dividirse en dos zonas de mínima intensidad, lo que introduce nuevos desafíos para el diseño, la operación y la confiabilidad de las misiones espaciales [24].

El origen de la SAA se asocia principalmente con la inclinación y el desplazamiento relativo entre el eje geomagnético y el eje de rotación terrestre, lo que produce un debilitamiento

local del campo magnético sobre el Atlántico Sur. Esta condición permite que las partículas energéticas atrapadas en el cinturón interno de Van Allen alcancen altitudes cercanas a los 200 km, incrementando significativamente la exposición radiativa y el riesgo para los satélites en LEO [25].

5.2 Efectos de la radiación espacial

La radiación presente en el entorno espacial, tanto en tránsito como atrapada por el campo magnético terrestre, puede generar diversos efectos adversos sobre los materiales y sistemas electrónicos de las naves espaciales. Entre ellos destacan los efectos acumulativos, asociados con la exposición prolongada a la radiación, que provocan una degradación progresiva de los materiales y del desempeño de los componentes electrónicos, así como los SEE, originados por la interacción puntual de una partícula con el nivel de carga suficiente para generar un efecto inmediato en un dispositivo, capaces de inducir errores lógicos, cambios de estado o incluso fallos permanentes en los circuitos integrados.

5.2.1 Efectos acumulativos

Los efectos acumulativos de la radiación se manifiestan principalmente a través de la dosis total ionizante (TID, por sus siglas en inglés) y el daño por desplazamiento (DD, por sus siglas en inglés), las cuales degradan de manera progresiva los materiales y dispositivos electrónicos. A continuación, se describen con detalle cada uno de estos mecanismos.

Daños por desplazamiento (DD)

Los dispositivos semiconductores dependen de una red cristalina bien ordenada que permite la conducción de corriente; cuando esta estructura se altera, las propiedades eléctricas del material cambian y pueden afectar el funcionamiento del dispositivo.

Cuando una partícula energética, como un neutrón, un protón o un ion pesado, colisiona con el núcleo de un átomo del material, el impacto puede expulsarlo de su posición original, generando una vacante en la red cristalina, mientras que el átomo desplazado ocupa una posición intersticial. El par formado por la vacante y el átomo intersticial se conoce como defecto de Frenkel, tal como se muestra en la figura 5.6.

En algunos casos, el átomo intersticial conserva suficiente energía para causar desplazamientos adicionales, produciendo un efecto en cascada que genera múltiples defectos en una pequeña región del material. Posteriormente, las vibraciones térmicas pueden permitir que algunos átomos regresen a sus posiciones originales en un proceso conocido como recocido (*annealing*). Sin embargo, este mecanismo no siempre logra restaurar completamente la red cristalina, por lo que persisten defectos residuales.

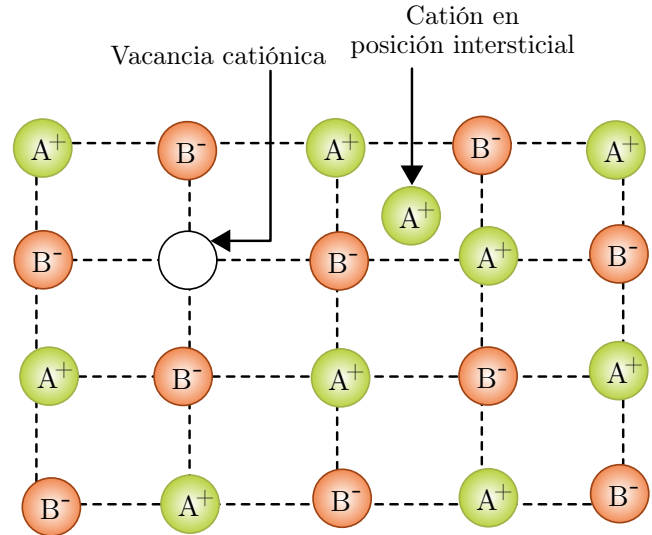


Figura 5.6. Defecto de Frenkel (adaptado de [26]).

Además, cuando neutrones, protones o iones pesados colisionan con un núcleo atómico, parte de la energía puede ser absorbida por el propio núcleo, llevándolo a un estado excitado que, al desexcitarse, emite radiación gamma capaz de inducir ionizaciones adicionales. En otros casos, el núcleo puede capturar la partícula incidente, transformándose en un isótopo radioactivo que decae mediante radiación alfa, beta o gamma, lo que genera nuevos procesos de ionización y desplazamiento, incrementando el daño total en el material.

Los defectos en la estructura cristalina modifican las propiedades eléctricas del semiconductor y, por ende, degradar progresivamente el desempeño del dispositivo. Esto resulta especialmente relevante en dispositivos optoelectrónicos, como celdas solares y fotodetectores, en los cuales el DD puede reducir su eficiencia y alterar su respuesta [27].

Dosis total ionizante (TID)

La cantidad total de energía transferida por la radiación ionizante a un material se denomina dosis total ionizante, la cual depende tanto del tipo y la energía de la radiación incidente como de las propiedades del material. La unidad oficial en el sistema internacional es el *gray* (Gy), definido como 1 J/kg. De forma equivalente, también se emplea la unidad *rad* (*radiation absorbed dose*), definida como 1 rad = 10^{-2} J/kg.

En el contexto de la electrónica espacial, la dosis de radiación suele expresarse en *rad(Si)*, unidad empleada para representar la energía absorbida en silicio, material base de la mayoría de los circuitos integrados. La exposición acumulativa a dosis elevadas puede provocar variaciones en los parámetros eléctricos de los dispositivos semiconductores, tales como la tensión umbral, la corriente de fuga y la ganancia, que con el tiempo puede conducir a la degradación o incluso al fallo permanente del dispositivo.

5.2.2 Efectos de evento único (SEE)

Los SEE son fenómenos estocásticos que surgen de la interacción de una partícula energética con un dispositivo electrónico y pueden clasificarse, de manera general, en *soft errors*, correspondientes a efectos no destructivos, y *hard errors*, asociados con efectos persistentes o potencialmente destructivos. Estos efectos se asocian principalmente con

la incidencia de neutrones, protones o iones pesados que inducen ionización en el material semiconductor, aunque también pueden generarse por electrones o muones mediante reacciones nucleares.

Tradicionalmente, se consideraban una amenaza únicamente los protones de alta energía ($E > 20$ MeV), pero con la tendencia actual hacia tecnologías VLSI, los protones de baja energía ($E < 3$ MeV) también contribuyen de manera significativa en la generación de *soft errors*, de modo que, dependiendo de la energía y del tipo de partícula incidente, pueden originarse mecanismos destructivos o no destructivos.

5.2.3 *Soft errors* [28]

Los *soft errors* no destruyen físicamente el dispositivo, pero sí afectan temporalmente su funcionamiento, por lo que no generan daños permanentes y, en muchos casos, pueden pasar desapercibidos, razón por la cual también se les denomina errores silenciosos. Entre los SEE asociados a este tipo de errores se encuentran los SEU, capaces de alterar el estado lógico de una celda de memoria o de un nodo interno del circuito. Sin embargo, cuando la perturbación afecta una unidad de control o un componente crítico, puede provocar alteraciones detectables en la operación del sistema, permitiendo la aplicación de medidas correctivas para restablecer su funcionamiento normal o evitar una falla más severa.

Upset de evento único (SEU)

Los SEU consisten en alteraciones del estado lógico almacenado en una celda de memoria o en un registro, provocando un cambio de estado sin dañar físicamente el dispositivo. Este fenómeno es común en aplicaciones espaciales y ocurre por la interacción de neutrones o protones con el silicio en circuitos electrónicos.

La susceptibilidad de un circuito ante un SEU depende de su carga crítica (Q_{crit}), la cual representa la mínima cantidad de carga necesaria para alterar la señal de salida. Esta carga está relacionada con la capacitancia nodal (C) y el voltaje de alimentación (V_{DD}), y puede expresarse como:

$$Q_{crit} \sim 2V_{DD}C \quad (5.2)$$

Junto con la transferencia lineal de energía (LET, por sus siglas en inglés), que representa la energía absorbida por unidad de longitud del material, se puede caracterizar la vulnerabilidad de los dispositivos electrónicos frente a partículas energéticas y estimar la tasa de ocurrencia de SEE.

Para ello se utiliza la sección transversal o sección eficaz (σ), la cual, de acuerdo con [28], es una medida del área efectiva asociada a la probabilidad de que ocurra un SEE en un dispositivo para un valor determinado de LET, y puede expresarse matemáticamente como:

$$\sigma_{SEE}(LET) = \frac{N_{SEE}}{\phi} \quad (5.3)$$

En donde N_{SEE} representa el número de eventos observados bajo una fluencia de partículas ϕ , medida en partículas/cm², que indica cuántas partículas atraviesan una unidad de área perpendicular al haz incidente.

Otro aspecto relevante en tecnologías VLSI es el efecto de compartición de carga (*charge sharing*), que puede dar lugar a un *upset* de múltiples *bits* (MBU, por sus siglas en inglés), en el cual una sola partícula altera simultáneamente varias celdas de memoria debido a su proximidad física, como se ilustra en la figura 5.7.

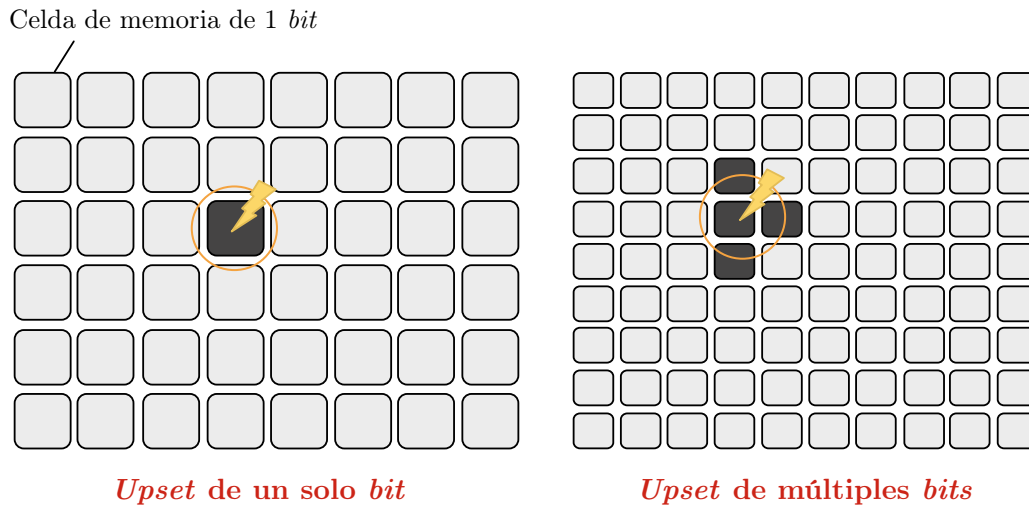


Figura 5.7. *Upset* de un solo *bit* frente a *upset* de múltiples *bits* (adaptado de [28]).

Transitorio por evento único (SET)

Los efectos transitorios por evento único (SET, por sus siglas en inglés) consisten en pulsos eléctricos generados en circuitos lógicos combinacionales que pueden propagarse y alterar memorias, aunque para que un SET afecte un elemento de memoria deben superarse varios mecanismos de enmascaramiento propios de los circuitos:

- **Enmascaramiento lógico:** Ocurre cuando el pulso SET se genera en una compuerta cuya salida no determina el estado lógico de la siguiente etapa, debido a que otra de sus entradas ya ha fijado dicho estado. En este caso, el pulso transitorio no se propaga a través del camino lógico, por lo que no produce alteraciones en la salida del circuito.
- **Enmascaramiento eléctrico:** Se produce cuando el pulso SET sufre una atenuación progresiva en su amplitud y duración al atravesar múltiples etapas lógicas, debido a pérdidas eléctricas, efectos capacitivos y limitaciones en la corriente de restauración del circuito, impidiendo que el pulso alcance un nivel suficiente para provocar un cambio de estado lógico.
- **Enmascaramiento por ventana de captura (*latching-window masking*):** Tiene lugar cuando el pulso SET no coincide temporalmente con la ventana de captura del *flip-flop* en el flanco del reloj, impidiendo que se produzca un cambio en el *bit* almacenado.

Interrupción funcional por evento único (SEFI)

Los efectos de interrupción funcional por evento único (SEFI, por sus siglas en inglés) se producen cuando una sección crítica de control o de retención de estado del dispositivo resulta afectada, provocando una interrupción de su operación. Este tipo de efecto puede originarse a partir de un SEU cuando este ocurre en una sección de control o de almacenamiento de estado de un dispositivo complejo, pudiendo desencadenar un SEFI. Este suele manifestarse como un reinicio del sistema o un bloqueo total, por lo que generalmente produce una alteración detectable en la operación del dispositivo.

5.2.4 Hard errors

Los *hard errors* se asocian con alteraciones persistentes o potencialmente destructivas en los dispositivos electrónicos, impidiendo su correcto funcionamiento, por lo que no pueden corregirse mediante reinicios ni mediante medidas de mitigación temporales, y el componente afectado debe ser reemplazado o considerado fuera de servicio. Estos errores suelen asociarse con efectos como el *latchup* de evento único (SEL, por sus siglas en inglés), la ruptura de transistores o el deterioro de la red de interconexión, lo que representa un riesgo significativo en sistemas espaciales, en donde la reparación física no es posible.

Entre los distintos efectos asociados a los *hard errors*, el SEL constituye uno de los efectos más relevantes en sistemas electrónicos espaciales debido al impacto que puede tener sobre la operación y confiabilidad de los dispositivos. Por esta razón, este efecto se describe con mayor detalle en la siguiente sección.

Latchup por evento único (SEL)

De acuerdo con [28], el SEL puede presentarse en dispositivos CMOS cuando una partícula de alta energía, como un protón o un ion pesado, atraviesa el dispositivo y deposita suficiente energía para generar una cantidad de carga capaz de activar estructuras parásitas presentes en su arquitectura interna. Estas estructuras pueden representarse mediante dos transistores bipolares parásitos, uno PNP y otro NPN, que forman una estructura equivalente de tipo PNPN con retroalimentación positiva. Cuando la carga generada por la interacción de la partícula es suficiente para activar esta estructura, se establece una trayectoria de baja impedancia entre la alimentación y tierra, produciendo un incremento significativo de la corriente de alimentación.

Este fenómeno puede provocar una interrupción del funcionamiento o daño permanente en el dispositivo. Su susceptibilidad depende de características como la tecnología de fabricación, la geometría del dispositivo, la tensión de alimentación, la temperatura y el LET de la partícula incidente.

A diferencia del SEU, que se manifiesta principalmente como un cambio en el estado lógico de un elemento de memoria, el SEL se caracteriza por un incremento sostenido de la corriente de alimentación asociado con la activación de estructuras parásitas del dispositivo. Debido a que el presente trabajo se centra en la inyección y caracterización de fallas asociadas con cambios de estado en microcontroladores, el SEL se presenta únicamente como parte de los principales SEE que pueden afectar a los dispositivos electrónicos.

5.3 Ensayos de radiación

Los ensayos de radiación se emplean para evaluar la robustez y confiabilidad de los dispositivos electrónicos frente a entornos con alta radiación, como los presentes en aplicaciones espaciales. Estas pruebas consisten en exponer el dispositivo bajo prueba (DUT, por sus siglas en inglés) a partículas ionizantes, tales como protones, neutrones o iones pesados, con el objetivo de reproducir los efectos de la radiación espacial y caracterizar su impacto sobre el comportamiento.

Una de las principales ventajas de estos ensayos es que permiten estudiar directamente la respuesta del dispositivo ante la radiación, proporcionando resultados experimentales representativos de los mecanismos que pueden presentarse durante su operación en el entorno espacial. Sin embargo, su ejecución requiere instalaciones especializadas, fuentes de radiación, instrumentación específica y procedimientos de seguridad, lo que incrementa el costo y la complejidad de los ensayos. Asimismo, estos ensayos presentan un control limitado sobre la ubicación interna en la que se produce un evento de radiación, lo que dificulta la reproducibilidad de las campañas experimentales.

Ante estas limitaciones, la FI constituye una alternativa a los ensayos de radiación para el estudio del comportamiento de los sistemas electrónicos ante los efectos producidos por la radiación, ya que permite introducir fallas de manera controlada y reproducible sin requerir instalaciones especializadas. Se ha demostrado experimentalmente que los resultados obtenidos mediante FI pueden presentar una buena correlación con los obtenidos en ensayos de radiación, permitiendo aproximar la respuesta del sistema ante la ocurrencia de estos efectos [29].

6. INYECCIÓN DE FALLAS (FI)

La FI se emplea para evaluar la confiabilidad de un sistema mediante experimentos controlados que permiten reproducir determinados efectos producidos por la radiación sobre dispositivos electrónicos. Esta técnica consiste en la inserción intencional de fallas que simulen los efectos de los SEU, con el propósito de observar la respuesta del sistema ante condiciones adversas y evaluar su impacto sobre el desempeño y la confiabilidad. Además, los resultados obtenidos pueden contrastarse con los correspondientes a ensayos de radiación. [30].

La FI puede aplicarse a simulaciones, modelos o prototipos funcionales, lo que permite identificar debilidades y puntos críticos antes de la implementación final. No obstante, su objetivo principal es evaluar la efectividad de los mecanismos de tolerancia a fallas, determinando si el sistema es capaz de mantener un funcionamiento frente a condiciones adversas.

Las herramientas y mecanismos empleados para la FI pueden clasificarse como invasivos o no invasivos, dependiendo de si dejan un rastro físico o alteran el comportamiento del sistema durante las pruebas, o si introducen las fallas sin interferir en su funcionamiento normal [31].

Contexto histórico

De acuerdo con [31], el primer acercamiento a la FI se realizó en 1972 por IBM con el objetivo de estimar la confiabilidad de programas de cómputo a partir del número de fallas artificialmente insertadas que eran detectadas durante las pruebas, además de las fallas reales identificadas en el sistema. Durante la década de 1980, la FI se consolidó como una herramienta de creciente interés en el ámbito académico, utilizándose principalmente para investigación, análisis experimental y validación de sistemas críticos.

Inicialmente, estos trabajos se centraron en comprender los mecanismos de propagación de errores y en evaluar la eficacia de los métodos de detección de fallas. En la actualidad, la FI se ha consolidado como un área de gran relevancia, especialmente en el desarrollo y validación de sistemas utilizados en aplicaciones espaciales y en infraestructuras críticas, como redes de comunicación y servicios de internet.

Entorno de una herramienta de inyección de fallas

La estructura de una herramienta de FI puede variar dependiendo de las especificaciones del sistema y del diseño implementado; sin embargo, la mayoría de los entornos de FI suelen incluir un conjunto de módulos básicos encargados de la generación, inyección, monitoreo y análisis de fallas. En la figura 6.1 se muestra un entorno típico de FI conformado por los siguientes elementos [29]:

- **Inyector de fallas:** Inyecta fallas en el sistema objetivo durante su ejecución, mientras este procesa las cargas generadas por el generador de cargas de trabajo.
- **Biblioteca de fallas:** Contiene los diferentes tipos de fallas, sus ubicaciones, los instantes de tiempo en los que ocurren y las estructuras de *hardware* o *software* involucradas.
- **Generador de cargas de trabajo:** Define y ejecuta ejemplos de cargas de trabajo¹ sobre el sistema objetivo para evaluar su comportamiento bajo condiciones específicas.
- **Controlador:** Supervisa y coordina el experimento de inyección de fallas, asegurando la correcta interacción entre los distintos módulos.
- **Monitor:** Supervisa la ejecución de los comandos y activa la recopilación de datos cuando es necesario.
- **Recolector de datos:** Se encarga de la adquisición y almacenamiento de los datos generados durante las pruebas de inyección.

¹Conjunto de entradas, comandos o secuencias de ejecución aplicadas al sistema objetivo para evaluar su comportamiento bajo condiciones específicas.

- **Analizador de datos:** Procesa y analiza los resultados, permitiendo identificar el impacto de las fallas y la efectividad de los mecanismos de tolerancia implementados.

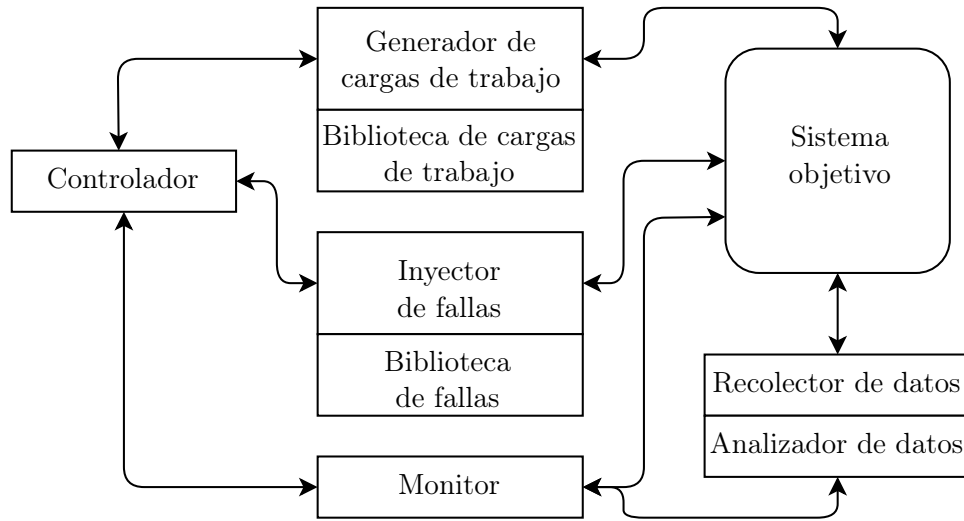


Figura 6.1. Entorno de una herramienta de inyección de fallas (adaptado de [31]).

6.1 Clasificación de las herramientas de inyección de fallas [31]

Las herramientas de FI pueden clasificarse de acuerdo con su implementación en cinco categorías principales: *software*, *hardware*, simulación, híbridas y de depuración, las cuales permiten reproducir el comportamiento de las fallas mediante la modificación del código, la alteración de señales físicas, la manipulación de modelos virtuales o la combinación de estos.

Asimismo, cada categoría emplea diferentes técnicas para inyectar fallas en el sistema objetivo, también denominado DUT, lo que da lugar a distintos niveles de intrusividad, precisión y fidelidad experimental. Estos factores influyen directamente en la validez y representatividad de los resultados obtenidos al evaluar la tolerancia a fallas del sistema.

A continuación se describen las características, ventajas y desventajas de cada una de estas categorías.

6.1.1 Inyección de fallas por hardware

La FI por *hardware* consiste en aplicar perturbaciones de manera directa sobre el sistema objetivo mediante dispositivos físicos diseñados para generar fallas controladas y analizar sus efectos. Estas perturbaciones se introducen principalmente a nivel de pines en los circuitos integrados.

En la FI por *hardware* es posible inyectar fallas permanentes o transitorias, como las fallas *stuck-at*, en las que una señal permanece forzada a un valor lógico constante, ya sea '0' (*stuck-at-0*) o '1' (*stuck-at-1*), independientemente del valor esperado por el sistema, así como cortocircuitos. Su nivel de intrusividad depende de la técnica empleada: en principio, puede considerarse no intrusivo si la perturbación se introduce únicamente en el punto seleccionado. Sin embargo, la incorporación de módulos o dispositivos externos puede alterar los tiempos de propagación de las señales y las características eléctricas del sistema, por lo que en la práctica se clasifica como parcialmente intrusivo.

En este contexto, se distinguen dos técnicas principales para la FI. La primera es la técnica de forzado, en la cual la falla se introduce directamente en un pin del circuito integrado, conector o punto de prueba, sin desconectar ningún componente del sistema. En este caso, el dispositivo inyector impone niveles lógicos altos o bajos sobre los nodos seleccionados con el fin de simular condiciones de falla, como las fallas del tipo *stuck-at*.

La segunda corresponde a la técnica de inserción, en la cual se sustituye temporalmente una parte del circuito por un dispositivo especializado que introduce la perturbación. Para ello, se interrumpe una conexión del circuito y la inyección se realiza sobre el nodo que se encuentra desacoplado y en estado de alta impedancia, permitiendo evitar el uso de señales de forzado directo y reducir el riesgo de daño en los componentes electrónicos.

Ventajas

La FI por *hardware* permite evaluar sistemas físicos bajo condiciones controladas, ofreciendo resultados experimentales que reflejan con alta fidelidad el comportamiento real del circuito en operación. Al no requerir modelos de simulación ni modificaciones del *software*, se reduce significativamente el tiempo de preparación de los experimentos, lo que permite la realización de campañas de prueba de forma más eficiente.

Además, esta técnica permite la FI permanentes y transitorias, adaptándose a distintos escenarios de evaluación y objetivos de análisis y, al emplear la técnica de inserción, contribuye a reducir el riesgo de daño en los componentes electrónicos en comparación con la aplicación directa de señales forzadas, mejorando así la seguridad del proceso experimental.

Desventajas

A pesar de sus ventajas, la FI por *hardware* puede implicar un riesgo significativo de daño físico en el sistema bajo prueba, particularmente cuando se aplican perturbaciones directas sobre los componentes. Asimismo, presenta una limitada portabilidad, ya que el montaje experimental suele depender estrechamente del diseño específico del *hardware*, lo que dificulta su reutilización en distintos sistemas. Además, la accesibilidad física al circuito restringe el número de puntos de inyección disponibles.

Por último, la FI por *hardware* requiere de equipo especializado y una preparación cuidadosa de cada experimento, lo que incrementa la complejidad del procedimiento y puede limitar la observabilidad y el control sobre los efectos de las fallas inyectadas.

6.1.2 Inyección de fallas por software

La FI por *software* consiste en modificar de manera controlada el código con el fin de simular fallas que podrían ocurrir durante la operación del sistema, permitiendo evaluar su comportamiento ante la ocurrencia de errores sin necesidad de *hardware* especializado. Al actuar directamente sobre el *software* en ejecución, este método permite inyectar fallas en registros, variables o áreas de memoria accesibles al programador.

Ventajas

La FI por *software* puede aplicarse en sistemas en donde la FI por *hardware* resulta difícil o inviable, ampliando así su campo de aplicación. Su principal ventaja radica en la flexibilidad y el bajo costo de infraestructura, ya que, como se mencionó, no requiere de *hardware* especializado, simplificando el proceso experimental. Además, puede emplearse tanto en sistemas físicos en operación como en plataformas de simulación y emulación.

Asimismo, permite ejecutar y repetir un gran número de campañas de inyección, facilitando el análisis de la respuesta del sistema ante distintos escenarios de prueba. Cuando estas campañas se llevan a cabo sobre un sistema real, la FI por *software* permite identificar errores de diseño existentes, ofreciendo una evaluación más representativa de su comportamiento en condiciones operativas.

Desventajas

Aunque la FI por *software* ofrece flexibilidad y bajo costo en instrumentación requerida, presenta ciertas limitaciones, ya que el rango de instantes de tiempo y ubicaciones en los que pueden inyectarse fallas suele estar restringido

a elementos accesibles desde el *software*, como variables, instrucciones del programa o cualquier recurso manipulable mediante código.

Además, este método requiere modificar el código fuente o el entorno de ejecución, de modo que el *software* bajo prueba no es estrictamente idéntico al empleado en la operación real, lo que puede introducir ligeras desviaciones en el comportamiento observado. La capacidad de observación y control también es limitada, centrándose principalmente en registros internos y áreas de memoria accesibles desde el *software*. Por último, resulta difícil simular fallas permanentes, ya que este método se basa principalmente en la modificación temporal de estados internos del sistema mediante *software*, sin alterar físicamente los componentes de *hardware*.

6.1.3 Inyección de fallas por simulación

La FI por simulación consiste en la construcción de un modelo detallado del DUT, que incluye el procesador y sus componentes, en el cual se inyectan fallas con el fin de analizar el comportamiento del sistema sin necesidad de intervenir el *hardware* físico. Este método permite reducir tanto los riesgos como los costos asociados a la evaluación, dado que los modelos se desarrollan mediante lenguajes de descripción de *hardware*, como VHDL o Verilog, los cuales permiten reproducir de manera precisa la operación del DUT y de sus elementos asociados.

Ventajas

La FI por simulación permite inyectar fallas en todos los niveles de abstracción del sistema, brindando un control completo sobre los modelos de falla y los mecanismos de inyección. Al no intervenir el *hardware* físico, se trata de un método completamente no intrusivo que reduce riesgos durante las pruebas, facilita la automatización de los experimentos y proporciona retroalimentación inmediata para los ingenieros de diseño, permitiendo ajustar rápidamente los modelos y las estrategias de tolerancia a fallas. Este método ofrece la posibilidad de modelar tanto fallas transitorias como permanentes, por ejemplo mediante la imposición persistente de estados lógicos incorrectos o comportamientos defectuosos dentro del modelo simulado.

Desventajas

Dentro de sus desventajas, la FI por simulación requiere el desarrollo de modelos detallados del DUT, lo que puede resultar complejo y laborioso, así como implicar costos asociados con la infraestructura y las herramientas de simulación requeridas. La disponibilidad y precisión de los modelos puede estar limitada, de modo que la exactitud de los resultados depende directamente de que la descripción del modelo represente de manera equivalente el funcionamiento interno de la implementación física del DUT. Por otro lado, los modelos podrían no contemplar todas las fallas de diseño presentes en el *hardware* físico, lo que reduce la fidelidad de la simulación frente a situaciones reales.

6.1.4 Inyección de fallas híbrida

La FI híbrida combina dos o más métodos con el objetivo de evaluar de manera más completa el DUT. Por ejemplo, al integrar técnicas empleadas por simulación, se incrementa significativamente la capacidad de observación y control, permitiendo el análisis de los componentes internos que no son accesibles mediante métodos exclusivamente de *hardware* o *software*. Esta combinación proporciona un equilibrio entre realismo y flexibilidad, permitiendo reproducir escenarios complejos, capturar fallas transitorias y permanentes, y evaluar con mayor precisión la efectividad de los mecanismos de tolerancia a fallas implementados en el sistema.

Ventajas

La FI híbrida ofrece ventajas significativas, ya que permite una evaluación más completa al combinar distintos métodos, incrementando la cobertura sobre distintos componentes y niveles de abstracción del DUT. Al aprovechar las fortalezas de cada técnica, se reducen los tiempos de prueba y se mejora la eficiencia del proceso experimental,

aumentando la observabilidad y el acceso a componentes internos que podrían no ser accesibles mediante un único método.

Desventajas

La FI híbrida presenta una mayor complejidad en su implementación debido a la necesidad de integrar y sincronizar múltiples herramientas y entornos de prueba. Esto incrementa el esfuerzo de configuración y mantenimiento del sistema experimental, así como los costos y recursos requeridos para su ejecución. La combinación de distintas técnicas también puede complicar el análisis y la correlación de los efectos observados, especialmente cuando intervienen diferentes niveles de abstracción y mecanismos de FI.

6.1.5 Inyección de fallas por depuración

La depuración en sistemas embebidos consiste en el uso de mecanismos de *hardware* y *software* que permiten observar y modificar el estado interno del DUT durante su operación, proporcionando acceso a memoria, registros y periféricos mediante interfaces de depuración (DI, por sus siglas en inglés), como JTAG o SWD. La FI por depuración aprovecha estas capacidades para inyectar fallas de manera controlada y analizar sus efectos sobre el comportamiento del sistema.

Durante el proceso, el *software* se ejecuta bajo condiciones controladas, interrumpiendo temporalmente la ejecución para alterar los valores de interés e inyectar la falla, y posteriormente se reanuda la operación con el fin de observar el comportamiento resultante. Este método resulta especialmente útil en entornos de desarrollo de sistemas embebidos, ya que permite validar la robustez del *software* y los mecanismos de tolerancia a fallas.

Ventajas

La FI por depuración permite inyectar fallas precisas en registros, memoria o periféricos sin necesidad de modificar el *software*, lo que brinda un control detallado sobre el momento y la duración de la falla; esta técnica facilita la identificación de vulnerabilidades y la evaluación de los mecanismos de tolerancia a fallas, y al aplicarse directamente sobre sistemas físicos, proporciona resultados que reflejan de manera precisa el comportamiento real del sistema en operación.

Desventajas

A pesar de sus ventajas, la efectividad de la FI por depuración depende de las capacidades del depurador y de la DI del DUT, lo que puede limitar el alcance de la FI y restringir la cobertura frente a fallas que ocurran en áreas no accesibles. La interrupción temporal de la ejecución puede alterar el comportamiento normal del sistema.

Herramientas de depuración

Como se mencionó, la FI por depuración se realiza mediante herramientas de *software* empleadas para depurar el DUT, cuyas capacidades de acceso y control sobre el DUT permiten la inyección de fallas. Aunque todas permiten ejercer control sobre el DUT, difieren en la forma en que gestionan dicho acceso y en el nivel de automatización que ofrecen.

Se pueden distinguir tres niveles: el nivel bajo, que proporciona acceso directo al *hardware* y permite automatizar completamente experimentos y campañas; el nivel intermedio, en el cual los comandos del depurador se transmiten a través de un servidor que los traduce a instrucciones compatibles con la DI del DUT; y el nivel alto, correspondiente a entornos de desarrollo integrados (IDE, por sus siglas en inglés), en donde la depuración se combina con interfaces gráficas y herramientas auxiliares, facilitando su uso pero limitando la automatización avanzada en comparación con los niveles inferiores.

Depuradores de bajo nivel (acceso directo al *hardware*)

En este nivel, las herramientas interactúan directamente con el *hardware* del DUT, lo que permite integrar campañas de FI de forma altamente automatizada y al no depender de servidores ni de IDE, se pueden diseñar flujos de trabajo repetibles y controlar registros, memoria y periféricos de manera flexible. A continuación, se presentan las principales herramientas utilizadas en este nivel:

- **OpenOCD:** De acuerdo con [32], OpenOCD destaca por su amplia compatibilidad con distintos núcleos y adaptadores de *hardware*, incluyendo ARM7, ARM9 y Cortex-M, así como por su capacidad para programar memorias *flash* internas y externas. Además, puede funcionar como servidor para clientes como *GNU Debugger* (GDB, por sus siglas en inglés), con la capacidad para integrarla con diferentes entornos de desarrollo y aprovechar sus recursos desde otras herramientas, convirtiéndola en una opción versátil y robusta para tareas avanzadas de depuración y experimentos complejos.
- **pyOCD:** Está diseñada para arquitecturas ARM Cortex-M y se distingue por su API en Python, que facilita la creación de *scripts* para ejecutar tareas complejas de depuración y campañas de inyección de fallas de manera eficiente y repetible. Puede actuar como servidor para el cliente de depuración GDB, permitiendo que este envíe comandos al DUT y pyOCD los traduzca en señales comprensibles por la MCU. Al mismo tiempo, ofrece un entorno flexible que facilita la automatización de experimentos y la integración en flujos de trabajo repetibles [33].

Depuradores de nivel intermedio (acceso mediante un servidor de depuración)

En el nivel intermedio, los depuradores funcionan como clientes que envían comandos a un servidor de depuración, el cual establece la conexión física con el DUT y traduce estos comandos a señales JTAG o SWD. Un ejemplo es GDB, un depurador de código abierto ampliamente utilizado para programas en C, C++ y otros lenguajes de programación, aunque existen otros clientes que operan de manera similar. Al depender del servidor para interactuar con el *hardware*, este enfoque permite utilizar el depurador en distintos entornos sin necesidad de conectarse directamente al DUT. Sin embargo, la automatización de campañas de inyección de fallas queda más limitada que con las herramientas de acceso directo al *hardware*.

Depuradores de nivel alto (depurador integrado en la IDE)

En el nivel alto, los depuradores están integrados dentro de la IDE, combinando edición de código, compilación y depuración en una misma interfaz gráfica. Este enfoque prioriza la interacción visual y la facilidad de uso, haciendo más accesible el seguimiento de registros, memoria, periféricos, *breakpoints* y ejecución paso a paso del programa.

Aunque internamente estos depuradores se apoyan en servidores o depuradores de *software* para conectarse al DUT, el usuario percibe un control directo sobre él, mientras que la automatización de campañas de FI queda más limitada que con herramientas de acceso directo al *hardware* o de acceso mediante un servidor de depuración.

Algunos IDE conocidos que integran esta funcionalidad son Keil uVision, Eclipse, STM32CubeIDE y MPLAB X, ofreciendo depuradores gráficos que simplifican la interacción con el DUT sin necesidad de configurar manualmente los mecanismos de acceso al *hardware*.

La tabla 6.1 presenta un resumen de los distintos niveles de herramientas de depuración, destacando su acceso al *hardware*, automatización, dependencia de servidores y facilidad de uso, permitiendo contextualizar su aplicación en campañas de FI en el DUT.

Tabla 6.1. Comparativa de herramientas de depuración.

Nivel	Ejemplos de herramientas	Acceso al hardware	Automatización	Dependencia de servidor	Facilidad de uso	Observaciones
Bajo	OpenOCD, pyOCD	Directo	Alta	No	Media	Ideal para experimentos repetibles y control detallado
Intermedio	GDB	Mediado por servidor	Media	Sí	Media	Flexibilidad limitada en automatización
Alto	IDE (Keil, Eclipse, STM32CubeIDE, MPLAB X)	Mediado por IDE / servidor	Baja	Sí	Alta	Fácil de usar, menor control directo, automatización limitada

6.2 Confiabilidad

La confiabilidad se define como la capacidad de un sistema para mantener un funcionamiento correcto y asegurar el cumplimiento del servicio que proporciona. En este contexto, los términos falla, error y fallo poseen significados técnicos específicos, los cuales se detallan más adelante en esta sección. De acuerdo con [31], su evaluación requiere considerar un conjunto de métricas que sirven como base para realizar un análisis cuantitativo de esta propiedad, las cuales son:

1. **Fiabilidad:** Probabilidad de que un sistema funcione correctamente durante un periodo de tiempo determinado, evaluando cuánto tiempo puede operar sin presentar fallas.
2. **Disponibilidad:** Probabilidad de que el sistema se encuentre en funcionamiento y disponible para realizar sus funciones en cualquier instante de tiempo, es decir, si está listo para ser utilizado en ese momento.
3. **Seguridad:** Probabilidad de que el sistema lleve a cabo sus funciones correctamente o, en caso contrario, suspenda su operación de manera segura, evitando consecuencias catastróficas para el entorno.
4. **Tiempo medio hasta el fallo (MTTF, por sus siglas en inglés):** Tiempo promedio que se espera que un sistema funcione antes de que ocurra un fallo.
5. **Cobertura:** Probabilidad de que el sistema sea capaz de recuperarse adecuadamente tras la ocurrencia de una falla, reflejando la efectividad de los mecanismos de recuperación incorporados.
6. **Latencia de falla:** Tiempo que transcurre entre la aparición de una falla y la manifestación del error asociado a dicha falla.
7. **Mantenibilidad:** Medida de la facilidad con la que un sistema puede ser reparado o restaurado a su estado funcional tras la ocurrencia de una falla.
8. **Testabilidad:** Capacidad de determinar la existencia y calidad de ciertos atributos dentro de un sistema, reflejando qué tan fácilmente puede probarse su correcto funcionamiento.

6.2.1 Factores que afectan la confiabilidad: falla, error y fallo

Los términos falla, error y fallo suelen usarse de manera indistinta en el lenguaje cotidiano. En el contexto de la inyección de fallas y de los sistemas tolerantes a fallas, estos conceptos tienen significados claramente diferenciados y de carácter técnico.

De acuerdo con [34], una falla o fallo puede referirse tanto a un defecto físico como a un desperfecto en el diseño del *software*, mientras que un error se entiende como la manifestación observable de esa falla o fallo en el sistema.

Por otra parte, [31] ofrece una distinción más precisa entre estos términos: define una falla como una imperfección física, defecto o anomalía que ocurre en algún componente de *hardware* o *software*; un error corresponde a la manifestación de esa falla; mientras que un fallo se refiere a la no realización de una acción esperada o requerida por el sistema, es decir, a su mal funcionamiento.

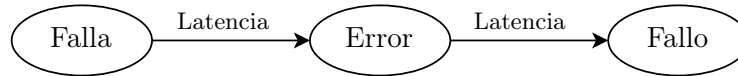


Figura 6.2. Cadena de falla, error y fallo (adaptado de [31]).

Como se observa en la figura 6.2, cuando ocurre una falla en alguna parte del sistema, esta puede generar uno o varios errores con potencial de propagarse por todo el sistema. En este contexto, los mecanismos de tolerancia a fallas juegan un papel fundamental, ya que buscan manejar dichas fallas y evitar que sus efectos afecten negativamente el comportamiento del sistema. No obstante, si estos mecanismos no actúan correctamente o resultan insuficientes, el sistema puede presentar un mal funcionamiento [31].

Las fallas pueden clasificarse como *hardware* o *software*, y cada una presenta características propias. A continuación, se detalla cada una de estas clasificaciones.

Fallas de *hardware*

Las fallas de *hardware* se producen durante el período de operación de un sistema y pueden clasificarse de acuerdo con su duración y persistencia. Esta clasificación permite identificar el tipo de efecto que generan y definir las estrategias adecuadas de detección y mitigación.

- **Fallas permanentes:** Son aquellas causadas por daños irreversibles en los componentes físicos del sistema. Una vez que ocurren, el componente afectado no puede recuperar su funcionalidad sin intervención externa [31].
- **Fallas transitorias:** Provocan un mal funcionamiento temporal. El componente se recupera por completo una vez que cesa la causa de la falla. Estas fallas son típicamente inducidas por condiciones externas, como interferencias electromagnéticas o partículas ionizantes [34].
- **Fallas intermitentes:** Se caracterizan por presentarse de forma no determinista. El componente alterna entre estados correctos (funcionamiento normal) y estados defectuosos (falla activa), dificultando su detección y corrección [34].

Fallas de *software*

A diferencia de las fallas de *hardware*, las fallas de *software* no se originan en defectos físicos, sino en problemas introducidos durante las etapas de especificación, diseño o implementación del código. Estas fallas pueden permanecer latentes y manifestarse únicamente bajo ciertas condiciones específicas, como cargas de trabajo elevadas o secuencias poco comunes de eventos [31]. Las fallas de *software* se pueden clasificar en:

- **Fallas funcionales:** Funcionalidades mal implementadas o que no cumplen con los requisitos del sistema.
- **Fallas en algoritmos:** Defectos en la lógica o secuencia de operaciones que no necesariamente afectan la arquitectura general.
- **Fallas de temporización/serialización:** Problemas por accesos concurrentes a recursos compartidos, generando condiciones de carrera o bloqueos.
- **Fallas de verificación:** Estructuras de control mal implementadas, como bucles o expresiones lógicas incorrectas.
- **Fallas de asignación:** Uso de valores mal definidos o variables no inicializadas, provocando comportamientos impredecibles o inestables.

6.3 Tolerancia a fallas

La tolerancia a fallas es la capacidad de un sistema para mantener un funcionamiento correcto a pesar de la presencia de fallas que podrían alterar su comportamiento. En las MCU, esta característica resulta esencial, dado que la continuidad de su operación puede mantenerse mediante mecanismos de redundancia que permitan detectar o enmascarar las fallas ocasionadas por radiación, interferencias electromagnéticas y/o defectos de *software*.

De acuerdo con [34], los sistemas tolerantes a fallas se diseñan aplicando estrategias que permiten detectar, corregir o recuperarse de las fallas sin interrumpir el servicio. Estas estrategias se basan en el uso de redundancia, es decir, la incorporación de recursos adicionales a los estrictamente necesarios para realizar una tarea, de manera que, a medida que ocurren fallas, la redundancia se aprovecha para enmascarar sus efectos, manteniendo así la funcionalidad del sistema.

Existen cuatro formas principales de redundancia (*hardware*, *software*, información y tiempo), cada una aplicada según el tipo de falla y las restricciones del sistema.

- **Redundancia de *hardware*:** Se implementa mediante la incorporación de componentes adicionales en el diseño con el objetivo de detectar o anular los efectos de un elemento defectuoso. Puede presentarse de forma estática, en donde los elementos redundantes actúan simultáneamente para enmascarar fallas de inmediato, o dinámica, en donde se activan componentes de reserva cuando un elemento activo deja de funcionar. También es posible combinar ambas en configuraciones híbridas.
- **Redundancia de tiempo:** Se basa en repetir una misma operación o ejecución para detectar fallas transitorias, dado que muchas fallas de *hardware* no se repiten en ejecuciones consecutivas. Esta técnica resulta efectiva para su detección y, aunque su costo en *hardware* y *software* es bajo, introduce una penalización en el rendimiento debido al tiempo adicional requerido.
- **Redundancia de información:** Consiste en agregar *bits* adicionales a los datos originales para permitir la detección y, en algunos casos, la corrección de errores. Este enfoque se aplica comúnmente en memorias y dispositivos de almacenamiento mediante códigos de detección y corrección de errores, así como en canales de comunicación en donde los datos pueden verse afectados por fallas transitorias. Si el código solo permite detectar errores, se puede recurrir a la retransmisión, aplicando así redundancia temporal.
- **Redundancia de *software*:** Se utiliza principalmente para mitigar fallas lógicas o de diseño. Una estrategia común consiste en desarrollar múltiples versiones independientes del mismo programa, de modo que si una falla, otra pueda asumir su función; estas versiones pueden ejecutarse en paralelo o de manera secuencial, aplicando así redundancia temporal.

La implementación adecuada de estos tipos de redundancia permite aumentar la disponibilidad del sistema y reducir la probabilidad de fallas críticas, y en el contexto de la validación de MCU, su estudio resulta esencial para comprender los mecanismos que pueden emplearse para detectar, mitigar o recuperar perturbaciones durante las pruebas de inyección de fallas.

7. UNIDAD DE MICROCONTROLADOR (MCU)

Las MCU son elementos fundamentales en los sistemas embebidos, los cuales, de acuerdo con [35], pueden definirse como computadoras pequeñas integradas en un solo chip, diseñadas para gestionar tareas específicas dentro de un sistema integrado sin requerir un sistema operativo complejo. En este contexto, una MCU puede entenderse como un dispositivo computacional, diseñado para ejecutar programas y controlar periféricos, lo que le permite funcionar de manera autónoma e interactuar con el entorno físico para ejecutar tareas específicas de forma eficiente [36].

7.1 Arquitectura de un microcontrolador

La arquitectura funcional de una MCU puede describirse como la integración de tres componentes principales: la unidad central de procesamiento (CPU, por sus siglas en inglés), la unidad de memoria y la interfaz de entrada/salida (E/S), tal como se muestra en la figura 7.1. Esta organización permite la ejecución de programas, así como el almacenamiento y procesamiento de datos, y la interacción directa con el entorno físico a través de periféricos integrados.

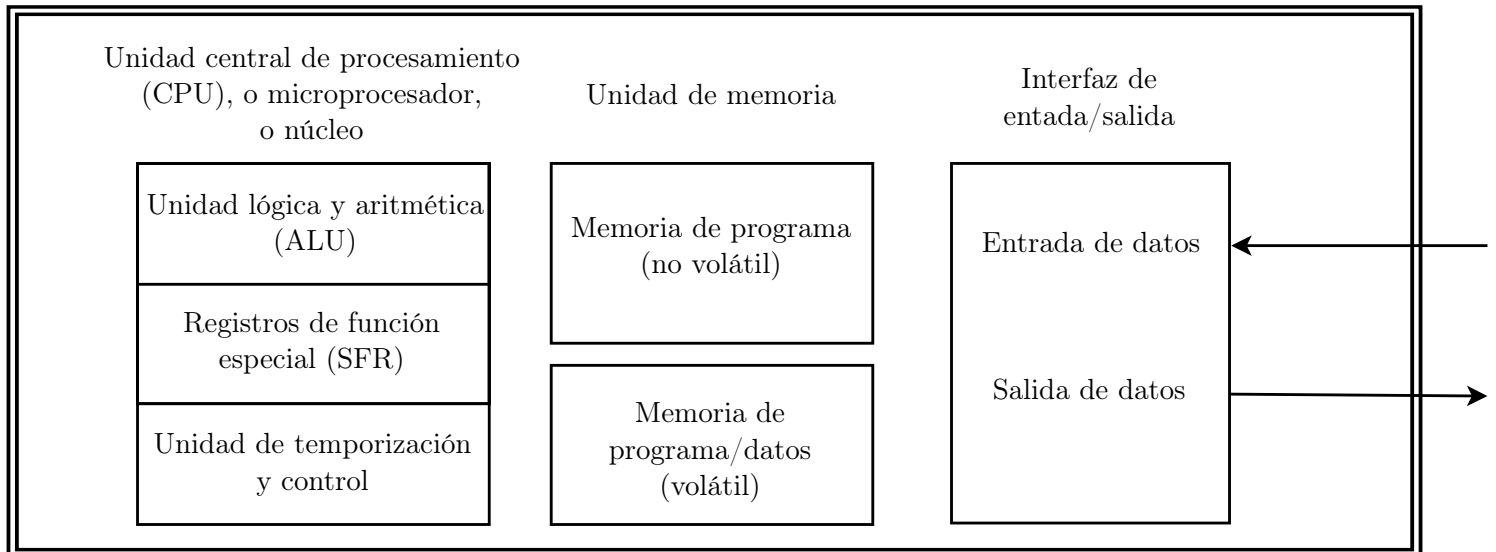


Figura 7.1. Arquitectura funcional de un microcontrolador (adaptado de [37]).

Si bien la arquitectura funcional presentada permite identificar los principales componentes que conforman una MCU, resulta importante analizar la manera en que estos se interconectan internamente. Desde una perspectiva estructural, la CPU, la memoria y la interfaz de E/S se encuentran enlazadas mediante buses internos, los cuales permiten el intercambio coordinado de información.

La figura 7.2 muestra una representación simplificada de esta interconexión, en donde se distinguen el bus de datos, el bus de direcciones y la señal de reloj del sistema. El bus de direcciones permite seleccionar la ubicación de memoria o el periférico al que se desea acceder, mientras que el bus de datos transporta la información entre los distintos componentes. Por su parte, la señal de reloj sincroniza las operaciones internas de la MCU [38].

Con base en la arquitectura funcional y estructural previamente presentada, a continuación se analizan de manera individual los componentes de la MCU con el propósito de comprender sus funciones específicas y su contribución al funcionamiento global del sistema.

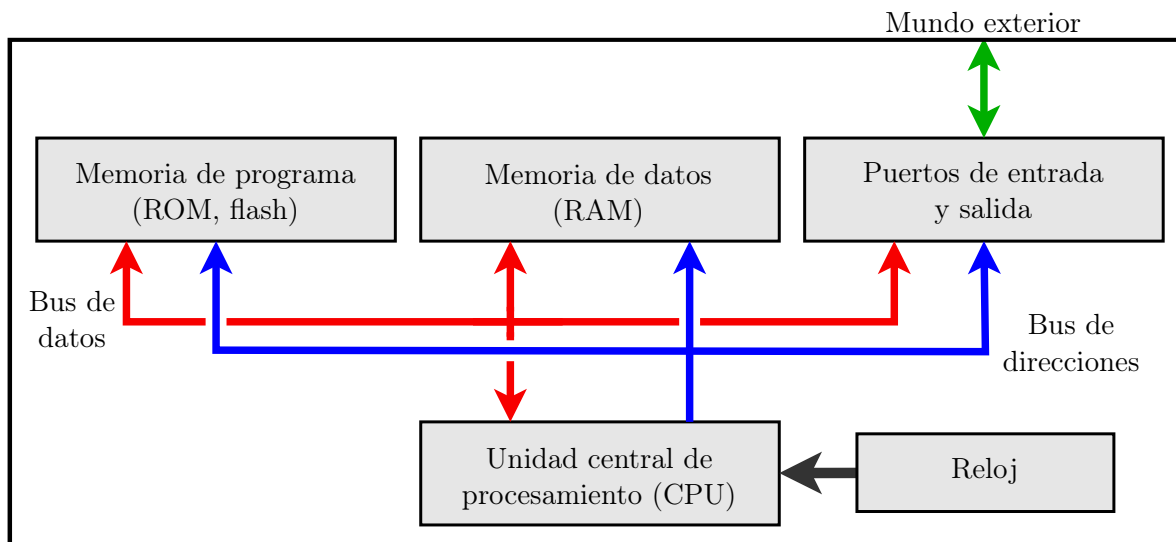


Figura 7.2. Estructura interna simplificada de un microcontrolador (adaptado de [38]).

7.1.1 Unidad central de procesamiento (CPU) [37]

La CPU, también denominada microprocesador o núcleo, es el componente responsable de ejecutar el *software* almacenado en la unidad de memoria, mediante la ejecución secuencial de instrucciones, constituyendo el elemento principal de control de la MCU. De acuerdo con [37], su funcionamiento puede describirse mediante tres etapas fundamentales que conforman el ciclo de instrucción:

- **Búsqueda de la instrucción:** Consiste en leer desde la memoria la instrucción correspondiente y transferirla a registros internos para su procesamiento.
- **Decodificación de la instrucción:** La CPU interpreta el tipo de operación que debe realizarse, ya sea aritmética, lógica, transferencia de datos o control de flujo.
- **Ejecución de la instrucción:** Se lleva a cabo la operación indicada, lo que puede implicar la actualización de registros internos, posiciones de memoria o señales de control hacia otros componentes del sistema.

Este proceso se repite continuamente durante la operación de la MCU, siguiendo el flujo definido por el programa, salvo cuando una instrucción de control, como un salto condicional o una interrupción, se modifica la secuencia normal de ejecución.

Desde el punto de vista estructural, la CPU está integrada por diversos módulos que permiten llevar a cabo estas funciones. Entre ellos se encuentra la unidad lógica y aritmética (ALU, por sus siglas en inglés), encargada de realizar operaciones matemáticas y lógicas; un conjunto de registros internos y registros de función especial (SFR, por sus siglas en inglés), que almacenan datos temporales, direcciones y estados del sistema, incluyendo el contador de programa, que indica la dirección de la siguiente instrucción a ejecutar; y la unidad de temporización y control, responsable de coordinar la secuencia de ejecución y sincronizar las operaciones internas con base en la señal de reloj del sistema.

7.1.2 Unidad de memoria

La unidad de memoria en una MCU es el componente encargado de almacenar tanto el programa que define el comportamiento del sistema como los datos necesarios para su operación. Las memorias pueden clasificarse con base en distintas características, entre ellas su volatilidad, la cual indica su capacidad para conservar o perder la información almacenada cuando se interrumpe la alimentación eléctrica.

Memoria no volátil [37]

La memoria no volátil es aquella que conserva la información almacenada aun cuando se interrumpe la alimentación del sistema. En las MCU, este tipo de memoria se utiliza comúnmente para almacenar el *firmware* y otros datos necesarios para la operación del sistema. El *firmware* corresponde al *software* que configura y controla el *hardware*, incluyendo los puertos de E/S, temporizadores e interfaces de comunicación, y que establece las condiciones de operación del sistema desde su inicialización.

Históricamente, esta memoria ha sido denominada memoria de solo lectura (ROM, por sus siglas en inglés). Sin embargo, en los sistemas modernos suele implementarse mediante tecnologías como memoria programable de solo lectura (PROM, por sus siglas en inglés), memoria programable de solo lectura borrable (EPROM, por sus siglas en inglés), memoria programable de solo lectura borrable eléctricamente (EEPROM, por sus siglas en inglés) o, más comúnmente, memoria *flash*. Debido a las características físicas de los dispositivos semiconductores empleados, presenta mayores tiempos de escritura y un mayor consumo energético durante las operaciones de programación, en comparación con la memoria volátil. No obstante, su capacidad para retener información aun en ausencia de alimentación eléctrica la convierte en un medio adecuado para almacenar el *firmware* y otros datos persistentes del sistema.

Memoria volátil

La memoria volátil es aquella que requiere alimentación para conservar la información almacenada, perdiendo su contenido cuando se interrumpe la energía. El principal tipo de memoria volátil en las MCU es la memoria de acceso aleatorio (RAM, por sus siglas en inglés), que se utiliza para almacenar temporalmente los datos y variables necesarios durante la ejecución del programa, permitiendo lecturas y escrituras rápidas y eficientes, lo que la hace adecuada para el procesamiento dinámico de información en tiempo real.

7.1.3 Interfaz de entrada/salida (E/S)

La interfaz de E/S permite que la MCU intercambie información con el entorno a través de sus puertos, recibiendo señales externas y generando respuestas de control hacia dispositivos externos. Los puertos de entrada captan señales provenientes de sensores, interruptores, teclados u otros sistemas electrónicos, y estas pueden ser digitales, que representan estados lógicos discretos, o analógicas, que requieren conversión mediante convertidores analógico-digitales (ADC, por sus siglas en inglés) antes de su procesamiento.

Por otra parte, los puertos de salida permiten activar y controlar dispositivos como LED, pantallas, motores o actuadores. En muchas MCU modernas, estas funciones se complementan con periféricos especializados, como temporizadores con modulación por ancho de pulso (PWM, por sus siglas en inglés) o convertidores digital-analógicos (DAC, por sus siglas en inglés), ampliando las capacidades de control del sistema.

7.2 Arquitectura ARM

ARM define arquitecturas de procesadores de acuerdo con los distintos tipos de aplicación, considerando aspectos como el desempeño, la respuesta temporal, el consumo energético y los requisitos de operación. Estas arquitecturas se organizan en diferentes perfiles, cada uno orientado a necesidades específicas.

7.2.1 Perfiles de la arquitectura ARM

De acuerdo con [39], ARM define tres perfiles principales de arquitectura: Cortex-A, orientado a sistemas de alto rendimiento capaces de ejecutar sistemas operativos complejos; Cortex-R, orientado a sistemas con requisitos de tiempo real; y Cortex-M, orientado a MCU y sistemas embebidos.

El perfil Cortex-A está diseñado para sistemas que requieren un elevado desempeño computacional y la ejecución de sistemas operativos, por lo que se emplea en computadoras, servidores, teléfonos inteligentes y otros sistemas de procesamiento de alto rendimiento.

Por otra parte, el perfil Cortex-R está diseñado para aplicaciones que requieren una respuesta temporal determinista, en las que las operaciones deben ejecutarse dentro de límites de tiempo definidos. Por ello, se utiliza en sistemas de control, aplicaciones automotrices, equipos médicos y otros sistemas con estrictos requisitos temporales.

Finalmente, el perfil Cortex-M está diseñado para dispositivos de menor tamaño, bajo consumo y alta eficiencia energética, por lo que se emplea principalmente en MCU y sistemas embebidos. Su diseño permite cubrir diferentes requisitos de procesamiento y funcionalidad dentro de este tipo de sistemas.

Procesadores Cortex-M

Los Cortex-M comprenden una familia de procesadores ampliamente empleados en sistemas embebidos. Estos procesadores se proporcionan como propiedad intelectual (IP, por sus siglas en inglés), permitiendo que distintos fabricantes los integren en sus propios dispositivos junto con memorias, periféricos, buses e interfaces de comunicación para conformar una MCU.

Dentro del perfil Cortex-M se encuentran diferentes procesadores orientados a distintos requisitos de procesamiento, consumo energético y funcionalidad. Entre ellos se encuentran el Cortex-M0, orientado a sistemas de bajo consumo y recursos limitados; el Cortex-M3, diseñado para aplicaciones embebidas de propósito general con un equilibrio entre desempeño y consumo; el Cortex-M4, que incorpora capacidades de procesamiento digital de señales y puede integrar una unidad de punto flotante; y el Cortex-M7, orientado a aplicaciones que requieren una mayor capacidad de procesamiento. Aunque cada uno presenta características particulares, comparten elementos generales de arquitectura y un modelo de programación común [39].

7.2.2 Modelo de programación de Cortex-M

El modelo de programación de los Cortex-M corresponde a los registros y recursos del procesador que son accesibles al *software*, así como a los mecanismos que permiten interactuar con la memoria y los periféricos. A pesar de que cada núcleo presenta características y aplicaciones diferentes, comparten una organización general que consta de registros de 32 *bits* utilizados para almacenar datos y direcciones, así como para controlar el flujo de ejecución del software [40, 41, 42].

El banco de registros está conformado por los registros de propósito general R0–R12 y los registros R13–R15, que desempeñan funciones específicas durante la ejecución del programa. El registro R13 corresponde al puntero de pila (SP, por sus siglas en inglés), R14 al registro de enlace (LR, por sus siglas en inglés) y R15 al contador de programa [41].

Los registros R0–R12 se utilizan principalmente para almacenar operandos, resultados intermedios y direcciones durante la ejecución de instrucciones. El registro de contador de programa contiene la dirección de la instrucción que se encuentra asociada al flujo de ejecución del programa, mientras que el LR almacena la dirección de retorno utilizada durante las llamadas a subrutinas. Finalmente, el SP permite gestionar la pila empleada para almacenar temporalmente información durante la ejecución del programa y el manejo de excepciones [41].

El puntero de pila puede hacer referencia al puntero de pila principal (MSP, por sus siglas en inglés) o al puntero de pila de proceso (PSP, por sus siglas en inglés), dependiendo del modo de operación y de la configuración del procesador. Estos recursos forman parte del estado de ejecución del núcleo y permiten mantener la información necesaria para controlar la secuencia de ejecución y responder ante eventos internos o externos [40, 41, 42].

Tabla 7.1. Principales registros del modelo de programación Cortex-M [41].

Registro	Denominación	Función principal
R0–R12	Registros de propósito general	Almacenan operandos, resultados intermedios, datos y direcciones utilizados durante la ejecución de instrucciones.
R13	Puntero de pila (SP)	Indica la posición actual de la pila. Puede hacer referencia al MSP o al PSP.
R14	Registro de enlace (LR)	Almacena información relacionada con el retorno de llamadas a subrutinas y determinados procesos de retorno de excepción.
R15	Contador de programa	Contiene la dirección asociada al flujo de ejecución del programa.

7.2.3 Organización de la memoria en Cortex-M

Los procesadores Cortex-M emplean un espacio de direcciones de 32 *bits*, permitiendo direccionar hasta 4 GB de memoria. Este espacio se encuentra organizado en regiones destinadas a distintos tipos de recursos, entre ellos código, memoria de acceso aleatorio estática (SRAM, por sus siglas en inglés), periféricos y componentes internos del sistema [41].

Espacio de direcciones

De manera general, las regiones inferiores de la memoria se destinan al código y a la SRAM, mientras que otras regiones se reservan para periféricos, memorias externas y componentes internos del procesador. La disponibilidad y uso específico de cada región depende de la implementación realizada por el fabricante de la MCU [41]. Las principales regiones del espacio de direcciones se muestran en la tabla 7.2.

Tabla 7.2. Organización general del espacio de direcciones en Cortex-M [41].

Rango de direcciones	Región	Descripción general
0x00000000–0x1FFFFFFF	Código	Región destinada principalmente a memoria de programa y ejecución de instrucciones.
0x20000000–0x3FFFFFFF	SRAM	Región destinada principalmente a memoria de datos de tipo SRAM.
0x40000000–0x5FFFFFFF	Periféricos	Región destinada al acceso a periféricos mapeados en memoria.
0x60000000–0x9FFFFFFF	RAM externa	Región destinada a memorias externas cuando la implementación dispone de ellas.
0xA0000000–0xDFFFFFFF	Dispositivos externos	Región destinada a dispositivos y periféricos externos.
0xE0000000–0xFFFFFFFF	Sistema	Región reservada para recursos internos del sistema, incluyendo componentes asociados al control, configuración y depuración.

Periféricos mapeados en memoria

Los periféricos pueden ser accesibles mediante registros asociados a direcciones dentro del espacio de memoria. Este mecanismo, denominado mapeo de periféricos en memoria, permite que el procesador acceda a los registros de control, configuración, estado y datos de los periféricos mediante operaciones de lectura y escritura sobre direcciones específicas. De esta manera, recursos como puertos de E/S, temporizadores, convertidores e interfaces de comunicación pueden ser configurados mediante la modificación de los valores almacenados en sus registros.

Cada periférico ocupa una región determinada dentro del mapa de memoria de la MCU y dispone de una dirección base, a partir de la cual se localizan sus diferentes registros mediante desplazamientos. La distribución concreta de estas direcciones es definida por el fabricante de la MCU, de acuerdo con los periféricos incorporados en cada dispositivo.

7.2.4 Excepciones e interrupciones en Cortex-M

Los Cortex-M incorporan excepciones que permite modificar temporalmente el flujo normal de ejecución para atender eventos internos o externos al procesador. Estas excepciones pueden ser generadas por condiciones propias del núcleo, como el reinicio o la detección de fallas, así como interrupciones asociadas a eventos provenientes de los periféricos y otros elementos del sistema [41].

La gestión de estos eventos se realiza mediante mecanismos integrados en la arquitectura que permiten identificar la fuente de la excepción, establecer prioridades y transferir la ejecución hacia la rutina correspondiente. Entre los elementos principales involucrados se encuentran el controlador de interrupciones vectorizadas anidadas (NVIC, por sus siglas en inglés) y la tabla de vectores.

Controlador NVIC

El controlador NVIC es el componente encargado de gestionar las interrupciones y parte de las excepciones en los procesadores Cortex-M. Su funcionamiento permite habilitar o deshabilitar interrupciones, identificar aquellas que se encuentran pendientes o activas y establecer niveles de prioridad para determinar el orden en que deben ser atendidas [41].

Una característica importante del NVIC es el manejo de prioridades, mediante el cual una excepción de mayor prioridad puede interrumpir la ejecución de otra de menor prioridad. Este mecanismo permite responder de manera eficiente a eventos cuya atención presenta diferentes niveles de importancia dentro del sistema.

La cantidad de interrupciones disponibles y el número de niveles de prioridad implementados pueden variar entre los diferentes procesadores y dispositivos Cortex-M, dependiendo de las características definidas por la arquitectura y de la implementación realizada por el fabricante.

Tabla de vectores

La tabla de vectores es una estructura almacenada en memoria que contiene la información necesaria para iniciar la ejecución del procesador y localizar las rutinas asociadas a las diferentes excepciones. Cada entrada de la tabla corresponde a un valor de 32 *bits* relacionado con una excepción específica [41].

La primera entrada contiene el valor inicial del MSP, mientras que la segunda contiene la dirección de la rutina de reinicio, denominada *Reset Handler*. Las entradas posteriores almacenan las direcciones de las rutinas encargadas de atender las demás excepciones e interrupciones disponibles en el sistema.

Cuando ocurre una excepción, el procesador identifica el número asociado al evento y utiliza este valor para localizar en la tabla de vectores la dirección de la rutina correspondiente, transfiriendo posteriormente la ejecución hacia dicha dirección.

Ubicación de la tabla de vectores

La ubicación de la tabla de vectores puede definirse mediante el registro de desplazamiento de la tabla de vectores (VTOR, por sus siglas en inglés) en aquellos procesadores Cortex-M que implementan este recurso. Este registro permite establecer una dirección base diferente para la tabla, haciendo posible su reubicación hacia otra región del espacio de memoria, como la RAM [41]. La posibilidad de modificar la ubicación de la tabla resulta útil en sistemas que requieren ejecutar *software* desde diferentes regiones de memoria o modificar dinámicamente la organización del sistema durante su operación.

Durante la entrada a una excepción, el procesador conserva automáticamente parte del estado de ejecución mediante la pila, permitiendo que la operación normal pueda continuar una vez finalizada la rutina de atención. Este mecanismo almacena información necesaria para restaurar el contexto previo y retornar al punto de ejecución interrumpido [41].

7.2.5 Infraestructura de depuración en Cortex-M

Los procesadores Cortex-M incorporan una infraestructura de depuración que permite acceder a diferentes recursos internos del sistema durante el desarrollo y análisis del *software*. Mediante esta infraestructura es posible controlar la ejecución del procesador y consultar o modificar información almacenada en registros y memoria. La comunicación con estos recursos se establece mediante interfaces de depuración como JTAG o SWD. En los sistemas que incorporan procesadores ARM, esta infraestructura se encuentra asociada con la arquitectura *CoreSight*, la cual proporciona los mecanismos necesarios para acceder y controlar diferentes recursos internos del sistema [43].

Depuración en chip (OCD)

La OCD es un enfoque de depuración basado en recursos de *hardware* integrados dentro del propio dispositivo, los cuales permiten observar y controlar su estado interno durante el desarrollo y análisis del sistema. Estos recursos conforman la infraestructura de depuración del dispositivo y permiten interactuar con el procesador sin necesidad de modificar directamente el *software*.

Mediante OCD es posible detener y reanudar la ejecución del procesador, ejecutar instrucciones paso a paso, consultar o modificar registros y posiciones de memoria, así como establecer puntos de interrupción (*breakpoints*) y puntos de observación (*watchpoints*). Estas capacidades permiten conocer el estado interno del sistema y analizar su comportamiento durante la ejecución.

Interfaces de depuración JTAG y SWD

La comunicación necesaria para realizar tareas de depuración en una MCU puede establecerse mediante interfaces diseñadas para transmitir información y comandos entre la computadora y el dispositivo. En los procesadores Cortex-M se emplean principalmente las interfaces JTAG y SWD.

JTAG se basa en el estándar IEEE 1149.1 y utiliza cinco señales para establecer la comunicación con el dispositivo. Por su parte, SWD fue desarrollada por ARM como una alternativa que requiere un menor número de señales, utilizando principalmente una línea bidireccional de datos y una señal de reloj [43].

Arquitectura CoreSight

CoreSight es una arquitectura desarrollada por ARM para proporcionar capacidades de depuración y trazado en sistemas que incorporan procesadores ARM. En el caso de los procesadores Cortex-M, esta infraestructura proporciona los mecanismos necesarios para acceder, observar y controlar diferentes recursos internos del sistema. Entre estas capacidades se encuentran la posibilidad de leer y modificar registros del procesador y de los periféricos, acceder a diferentes regiones del espacio de memoria y detectar eventos específicos durante la ejecución, como el acceso a una determinada dirección de memoria o la llegada a un punto concreto del programa [44]. Para utilizar estas capacidades, la PC anfitriona se conecta con la MCU a través de una unidad de depuración, un dispositivo físico que actúa como intermediario entre ambos sistemas. Esta unidad recibe las operaciones solicitadas desde la PC y las transmite hacia la MCU mediante una interfaz de depuración, como JTAG o SWD.

Una vez establecida la comunicación, *CoreSight* proporciona el acceso a los recursos internos necesarios para realizar las tareas de depuración. De esta manera, es posible consultar o modificar posiciones de memoria y registros, así como controlar temporalmente la ejecución del procesador. La figura 7.3 muestra de manera general la comunicación entre la PC, la unidad de depuración y un dispositivo que incorpora la infraestructura *CoreSight*.

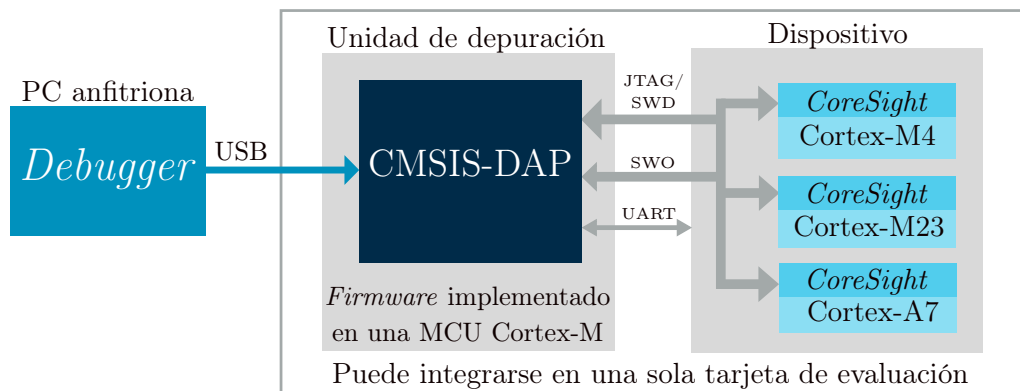


Figura 7.3. Arquitectura de depuración basada en CMSIS-DAP y CoreSight (adaptado de [45]).

Como se observa en la figura 7.3, la unidad de depuración permite establecer la comunicación entre la PC y los recursos de depuración disponibles en el dispositivo. Dentro de la arquitectura CoreSight se encuentran los siguientes elementos:

- **Puerto de acceso de depuración (DAP, por sus siglas en inglés):** comunica la interfaz de depuración con los diferentes recursos internos del sistema [45].
- **Puerto de acceso a memoria (MEM-AP, por sus siglas en inglés):** permite realizar operaciones de lectura y escritura sobre direcciones del sistema [46].
- **Registro de control y estado de depuración (DHCSR, por sus siglas en inglés):** permite conocer el estado del procesador y realizar operaciones como detener o reanudar su ejecución [46].
- **Unidad Flash Patch and Breakpoint (FPB, por sus siglas en inglés):** permite establecer puntos de interrupción (*breakpoints*) para detener la ejecución cuando el programa alcanza una dirección determinada [46].
- **Unidad Data Watchpoint and Trace (DWT, por sus siglas en inglés):** permite observar eventos durante la ejecución del procesador y detectar accesos a determinadas direcciones mediante puntos de observación (*watchpoints*) [46].

En conjunto, estos elementos proporcionan las capacidades necesarias para acceder y controlar el estado interno de una MCU Cortex-M. El DAP establece el acceso a los recursos internos del dispositivo, los MEM-AP realizan operaciones sobre el espacio de memoria y los recursos de depuración del procesador controlan determinados aspectos de su ejecución.

Estas capacidades resultan especialmente relevantes para la FI mediante OCD, ya que permiten detener temporalmente la ejecución del procesador, consultar el contenido de una dirección de memoria o de un registro, modificar su valor y posteriormente reanudar la ejecución. De esta manera, es posible introducir alteraciones controladas en recursos como la memoria RAM y los registros de configuración de periféricos sin modificar directamente el *software*.

Por otro lado, los *breakpoints* permiten seleccionar puntos determinados del programa en los que se desea detener la ejecución antes de realizar una modificación, proporcionando control sobre el momento en el que se interviene el sistema. Por lo tanto, la arquitectura CoreSight constituye un elemento fundamental para la implementación de técnicas de FI mediante depuración en procesadores Cortex-M.

7.2.6 Archivos para la descripción y análisis del sistema

Durante el desarrollo de *software* bajo prueba se generan y utilizan diferentes archivos que contienen información sobre la organización del programa, su distribución en memoria y las características del dispositivo sobre el que se ejecuta. Entre estos se encuentran los archivos de formato ejecutable y enlazable (ELF, por sus siglas en inglés) y MAP, asociados al proceso de compilación y enlazado del *software*, así como los archivos SVD, utilizados para describir los periféricos y registros disponibles en una MCU.

La información proporcionada por estos archivos permite conocer tanto la organización del *software* como la de los recursos de *hardware* del dispositivo, facilitando su análisis mediante herramientas externas.

Archivo ELF

El ELF es un formato utilizado para representar archivos ejecutables y bibliotecas generados durante el proceso de compilación de *software*. Su estructura almacena tanto el código y los datos que conforman un programa, como información adicional necesaria para su enlazado, carga y análisis [47].

Un archivo ELF contiene un encabezado principal y diferentes estructuras que describen la información que contiene. Entre estas estructuras se encuentran las secciones, que organizan los diferentes tipos de información del programa, y los segmentos, que describen las partes del archivo que deben considerarse durante su carga en memoria [47].

Entre las secciones comúnmente presentes en un archivo ELF se encuentran `.text`, que contiene las instrucciones ejecutables del programa; `.data`, destinada a datos inicializados; `.bss`, asociada a datos no inicializados; y `.rodata`, utilizada generalmente para datos de solo lectura. Asimismo, el archivo puede incorporar tablas de símbolos que relacionan identificadores del programa con direcciones y características de los elementos definidos durante la compilación y el enlazado.

Archivo MAP

El archivo MAP es un reporte generado durante el proceso de enlazado que describe la distribución de los elementos del programa dentro del espacio de memoria definido para el sistema. A diferencia del archivo ELF, que contiene de forma estructurada el código, los datos e información asociada al ejecutable, el archivo MAP presenta esta información en un formato legible, lo que permite identificar las regiones de memoria utilizadas, las direcciones asignadas a las diferentes secciones, su tamaño y la ubicación de los símbolos y módulos que forman parte del programa. Esta información depende de las reglas establecidas durante el enlazado, particularmente de la configuración definida mediante el *linker script*.

En este contexto, el archivo MAP facilita la consulta de la distribución del programa en memoria y permite obtener información sobre las direcciones y tamaños de las diferentes regiones que lo componen.

Archivo SVD

El archivo SVD describe de manera estructurada los recursos disponibles en una MCU basada en Cortex-M, particularmente los periféricos y sus registros mapeados en memoria [48]. Estos archivos emplean una estructura jerárquica en la que se describe el dispositivo y, dentro de este, los periféricos que lo conforman. Cada periférico contiene diferentes registros, cuya información puede incluir su dirección, tamaño y permisos de acceso.

8. ESTADO DEL ARTE DE LAS HERRAMIENTAS DE INYECCIÓN DE FALLAS

El estado del arte corresponde a una revisión del conocimiento existente sobre un tema, con el propósito de identificar los últimos avances, enfoques y soluciones propuestas en la literatura. Además, cumple la función de situar el proyecto dentro del contexto actual de investigación, permitiendo conocer qué se ha hecho, cómo se ha hecho y cuáles son las áreas que aún requieren desarrollo.

En el ámbito de la FI, esta revisión es especialmente relevante debido a la variedad de técnicas, niveles de aplicación y metodologías diseñadas para evaluar la confiabilidad de sistemas electrónicos, particularmente en aplicaciones espaciales, en donde la FI representa una alternativa más accesible y flexible frente a los ensayos de radiación. La evolución de estas herramientas ha permitido mejorar la automatización y abordar arquitecturas cada vez más complejas, lo que ha impulsado el desarrollo continuo de nuevas soluciones de inyección orientadas a simular de manera controlada las alteraciones asociadas a los efectos SEU.

Con el fin de facilitar la comprensión, en esta sección se clasifican las herramientas de FI en tres grandes categorías: *software*, simulación y depuración; analizando sus principales características, ventajas y limitaciones, con el objetivo de ofrecer una visión integral y actualizada del estado del arte en el ámbito de la FI.

8.1 Herramientas de inyección de fallas por *software*

Las herramientas de FI por *software* representan una evolución clave en el campo al trasladar la inyección directamente al nivel de ejecución del programa; mediante la inserción de instrucciones en puntos estratégicos, así como la modificación dinámica de datos y operaciones durante la ejecución, es posible simular fallas lógicas, corrupción de variables y alteraciones en el flujo de control, lo que facilita la evaluación de la respuesta del sistema sin requerir *hardware* adicional, además de que permite realizar experimentos repetibles.

Su principal ventaja radica en la flexibilidad y en la posibilidad de ejecutar campañas de inyección a gran escala. Sin embargo, la fidelidad en la reproducción de los efectos inducidos por radiación es inferior a la obtenida mediante ensayos experimentales. A pesar de ello, estas herramientas siguen siendo ampliamente utilizadas para la validación inicial de la tolerancia a fallas en una MCU y microprocesadores, especialmente durante las etapas de desarrollo y prueba de *software* embebido.

8.1.1 Inyección de fallas genérica orientada a objetos (GOOFI) [49]

La herramienta de inyección de fallas genérica orientada a objetos (GOOFI, por sus siglas en inglés), desarrollada en la Universidad Tecnológica de Chalmers, Suecia, permite la ejecución de campañas de FI en diversos sistemas objetivo mediante múltiples técnicas. Su implementación en Java le confiere portabilidad y facilidad de extensión, simplificando la adaptación y reutilización de componentes en distintos escenarios de prueba. Entre las técnicas soportadas destacan la inyección de fallas implementada por *software* (SWIFI, por sus siglas en inglés) y mediante cadenas de escaneo (SCIFI, por sus siglas en inglés).

Inyección de fallas genérica orientada a objetos versión 2 (GOOFI-2) [50]

La evolución de GOOFI hacia su segunda versión, GOOFI-2, surge de la necesidad de contar con una herramienta aún más flexible que permita integrar nuevas técnicas de FI sin comprometer la compatibilidad con las plataformas

ya soportadas, incorporando tres enfoques relevantes de inyección por *software*: la inyección basada en *nexus*¹, la inyección basada en excepciones y la inyección basada en instrumentación.

El primer enfoque incorporado en GOOFI-2 corresponde a la FI basada en *nexus*, la cual se apoya en interfaces avanzadas de depuración que permiten acceder a la memoria y a determinados registros del procesador mediante una herramienta externa. Adicionalmente, el uso de *breakpoints* permite detener la ejecución del sistema en instantes precisos, brindando la posibilidad de modificar de forma controlada el contenido de registros o memoria antes de reanudar la ejecución.

Este enfoque ofrece un alto grado de control y repetibilidad, ya que la falla se inyecta inmediatamente antes de la ejecución de una instrucción específica, sin requerir modificaciones en la carga de trabajo. No obstante, su principal limitación es la elevada intrusividad temporal asociada a la detención del procesador, así como la dependencia de puertos de depuración especializados que no están disponibles en todos los dispositivos.

Como alternativa al enfoque basado en *nexus*, GOOFI-2 incorpora un segundo enfoque basado en excepciones, en el cual, al inicio de cada experimento se carga en el sistema objetivo una rutina de excepción diseñada específicamente para la falla que se desea inyectar. Esta rutina se activa mediante *breakpoints* de *hardware* durante la ejecución de la carga de trabajo, interrumpiendo temporalmente el programa y permitiendo modificar información interna del sistema, como registros o ubicaciones de memoria, con el fin de provocar la falla. Una vez realizada la inyección, el sistema reanuda su ejecución hasta completar el experimento.

Si bien este enfoque permite realizar inyecciones de fallas sin requerir de interfaces externas de depuración, presenta una observabilidad limitada, ya que el estado interno del sistema únicamente puede ser observado al finalizar cada ejecución, sin permitir su monitoreo durante la operación normal.

Finalmente, GOOFI-2 incorpora la FI basada en instrumentación, orientada a escenarios en los que no se dispone de soporte de depuración avanzado en el *hardware*. Este enfoque se basa en la modificación directa del *software* antes de su ejecución. El archivo ejecutable es instrumentado mediante la inserción de código adicional que introduce puntos de inyección, comúnmente implementados como *breakpoints* por *software*, los cuales desvían temporalmente el flujo de ejecución hacia una rutina encargada de alterar el contenido de registros o ubicaciones de memoria específicas.

Este enfoque se caracteriza por no requerir *hardware* de depuración externo. Sin embargo, la instrumentación introduce sobrecarga temporal en la ejecución y altera el comportamiento temporal del sistema objetivo. Además, la técnica no proporciona mecanismos propios de observabilidad, por lo que el análisis de los efectos de la falla suele depender de herramientas complementarias.

En conjunto, los enfoques incorporados en GOOFI-2 amplían el alcance funcional de su predecesor y permiten su empleo como herramienta para la evaluación de la fiabilidad. Sin embargo, también revela limitaciones relevantes, ya que carece de mecanismos para extrapolar los resultados a configuraciones distintas y no ofrece estimaciones sobre la incertidumbre asociada a los análisis realizados, lo que dificulta la generalización de los resultados y limita su utilidad como evidencia en sistemas críticos. De este modo, aunque representa un avance significativo, su implementación aún enfrenta desafíos importantes para consolidarse como una solución robusta y confiable en la evaluación de la tolerancia a fallas.

8.1.2 Herramienta de inyección de fallas en microprocesadores (MiFIT) [52]

La herramienta de inyección de fallas en microprocesadores (MiFIT, por sus siglas en inglés), se caracteriza por su diseño modular, su naturaleza de código abierto y su compatibilidad con múltiples entornos. Está desarrollada en

¹Estándar de depuración basada en la especificación de depuración IEEE ISTO 5001 [51].

*BASH*², lo que permite adaptarla fácilmente a diversas plataformas y la convierte en una opción accesible y flexible para investigadores y desarrolladores.

Como se muestra en la figura 8.1, MiFIT se organiza en tres módulos principales: el configurador de campañas de inyección de fallas, la interfaz de inyección y el graficador. Esta arquitectura modular permite separar la definición del experimento, la ejecución de las campañas y el análisis de los resultados.

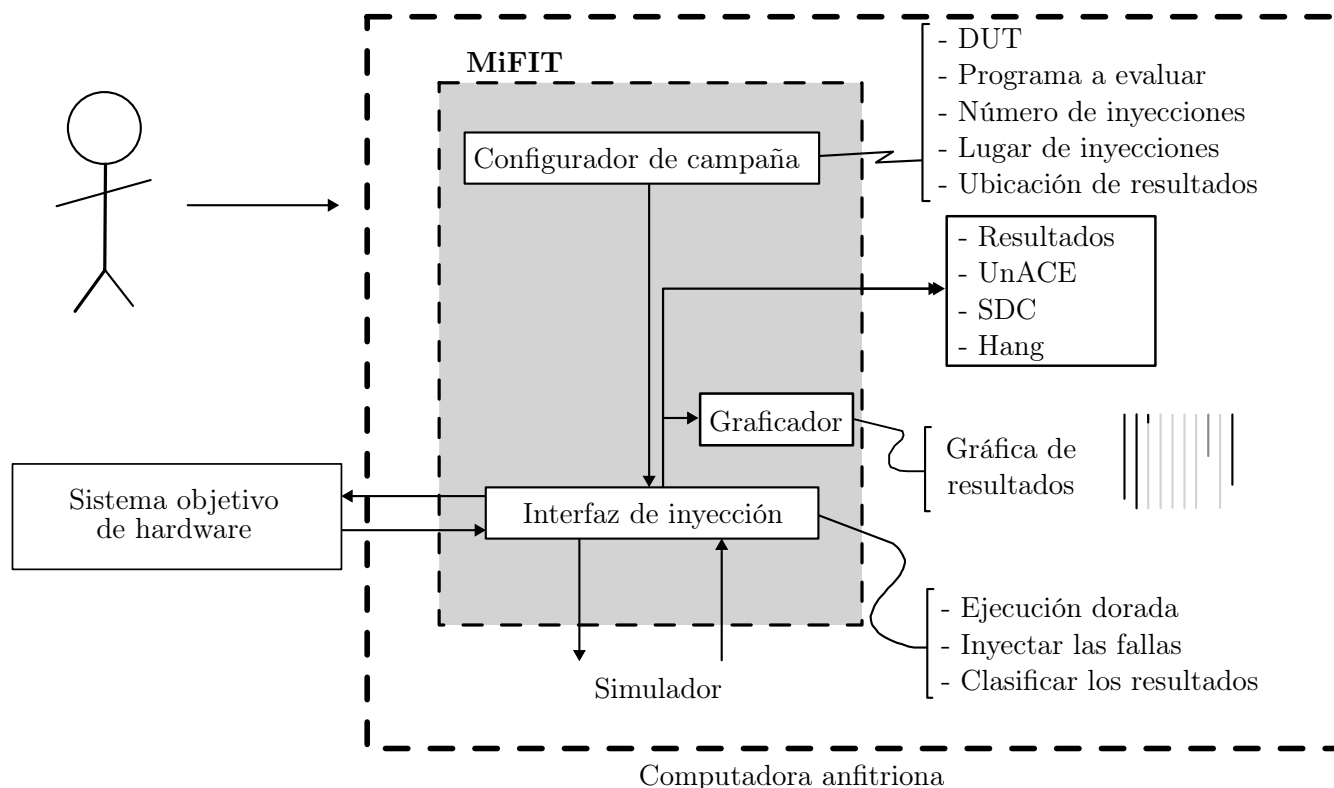


Figura 8.1. Arquitectura MiFIT (adaptado de [52]).

El configurador de campañas corresponde a la interfaz gráfica de usuario (GUI, por sus siglas en inglés) de la herramienta, desde la cual se realiza la configuración inicial de la campaña de FI, permitiendo definir parámetros como el DUT, el archivo binario a evaluar, el número de inyecciones y el lugar en donde se almacenan los resultados. Una vez establecida la configuración, esta información se transfiere a la interfaz de inyección para su ejecución.

La interfaz de inyección es responsable de preparar y ejecutar las campañas sobre el sistema objetivo, ya sea en entornos de simulación, emulación o mediante herramientas de OCD. Durante la ejecución, los resultados obtenidos se comparan con una ejecución de referencia o ejecución dorada, lo que permite clasificar los efectos de las fallas inyectadas en tres categorías: corrupción silenciosa de datos (SDC, por sus siglas en inglés), cuando la ejecución finaliza pero se obtienen resultados erróneos; innecesario para la ejecución arquitectónicamente correcta (UnACE, por sus siglas en inglés), cuando se obtienen resultados correctos y la falla no produce efectos observables; y *hang*, cuando la falla provoca bloqueos o terminaciones anómalas del sistema.

Finalmente, el módulo graficador organiza la información recopilada durante las campañas de inyección y produce representaciones gráficas que facilitan el análisis y la interpretación de los resultados.

Una de las principales fortalezas de MiFIT radica en su capacidad para incorporar nuevas técnicas de inyección mediante modificaciones a nivel de *software* y adaptarse a arquitecturas específicas, además de automatizar gran parte

²Programa que interpreta comandos y también funciona como un lenguaje de programación y se ejecuta dentro del entorno del *shell* en sistemas operativos Unix [53].

del flujo de trabajo, desde la configuración hasta el análisis, lo que incrementa notablemente la eficiencia en el estudio de fallas transitorias en microprocesadores.

Sin embargo, esta flexibilidad también implica desafíos, ya que su funcionamiento depende de características específicas del DUT, lo que requiere adaptar o reescribir la interfaz de inyección para cada nueva plataforma. Además, la herramienta solo permite la inyección en registros de propósito general, lo que restringe su alcance frente a otras métodos, como la depuración, que permiten intervenir en cualquier zona de la MCU.

8.1.3 Banco de pruebas para estudios en ambientes hostiles sobre circuitos integrados (THESIC) [29]

El banco de pruebas para estudios en ambientes hostiles sobre circuitos integrados (THESIC, por sus siglas en inglés) es una plataforma experimental diseñada para la inyección controlada de fallas en circuitos integrados, con el objetivo de emular los efectos de eventos transitorios inducidos por radiación. Su diseño permite automatizar la inyección de fallas en microprocesadores, MCU y otros dispositivos electrónicos, mediante la generación de interrupciones y la modificación controlada de registros y memoria.

A diferencia de otras herramientas, THESIC ha sido validada mediante ensayos de radiación y emplea la técnica de *upsets* emulados por código (CEU, por sus siglas en inglés). Esta técnica consiste en inyectar *bit-flips* en el contenido de registros o memoria del DUT, utilizando interrupciones para detener momentáneamente la ejecución, realizar la inyección sobre los sistemas objetivos y reanudar el programa sin alterar su funcionamiento general.

Este mecanismo de inyección presenta un bajo impacto temporal, ya que requiere pocos ciclos de reloj y es independiente de la complejidad de la aplicación, permitiendo realizar múltiples inyecciones durante la ejecución del programa y obtener estadísticas confiables sobre el comportamiento del sistema frente a las fallas emuladas.

La arquitectura de THESIC se organiza en un sistema de dos placas: una placa base, encargada de gestionar la operación del sistema y servir como interfaz con el usuario; y una placa hija, responsable de adaptar el DUT al protocolo de la placa base mediante una interfaz mapeada en memoria (MMI, por sus siglas en inglés).

La comunicación entre ambas placas se realiza de manera asíncrona a través de la MMI. Durante la ejecución de una prueba, el DUT interrumpe para indicar que existen datos listos para ser transferidos. Ante esta señal, la placa base interrumpe a la placa del DUT con el fin de leer los resultados almacenados y, a partir de su análisis, detectar la presencia de errores derivados de las fallas inyectadas.

Adicionalmente, THESIC incorpora un *watchdog* por *software* programable que restablece el sistema en caso de errores críticos que afecten la ejecución del DUT, además de combinar técnicas de inyección pseudoaleatorias y deterministas. Esta flexibilidad permite tanto el análisis estadístico sobre la sensibilidad del programa a las perturbaciones, como la reproducción específica de eventos críticos para estudios detallados.

La validación de esta herramienta se llevó a cabo mediante experimentos de inyección, en los que se aplicaron miles de *bit-flips* en diferentes instantes de tiempo y ubicaciones de memoria durante la ejecución de programas en dispositivos basados en una MCU 80C51 y un procesador digital de señales 320C50. De acuerdo con [29], los resultados presentan coherencia con datos provenientes de pruebas de radiación, lo que confirma la eficacia del enfoque para evaluar la confiabilidad de sistemas en condiciones equivalentes a ambientes hostiles, sin exponerlos directamente a condiciones radiactivas.

La figura 8.2 muestra el sistema THESIC integrado dentro de una cámara de vacío, utilizada durante los ensayos experimentales.

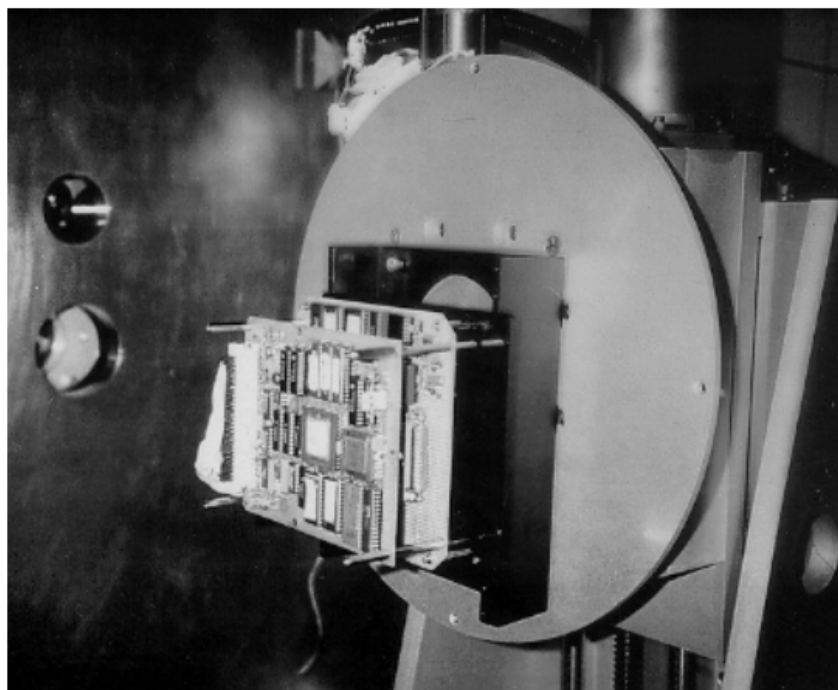


Figura 8.2. Sistema THESIC dentro de la cámara de vacío disponible en instalaciones del L. B. L. (Berkeley California) [29].

8.1.4 Inyección automática en tiempo real de fallas y errores (*FERRARI*) [54]

Otra herramienta relevante de FI por *software* es la inyección automática en tiempo real de fallas y errores (*FERRARI*, por sus siglas en inglés), diseñada para ejecutarse en sistemas operativos basados en *Unix*³. Su objetivo es alterar de forma controlada el estado de ejecución de programas durante su funcionamiento, con el fin de analizar la respuesta del sistema frente a distintos tipos de fallas y errores.

Estas fallas pueden implementarse mediante distintos mecanismos: corrupción de memoria, en donde se altera parte del contenido de la memoria del programa antes de su ejecución sustituyendo datos por valores incorrectos; inyección espacial, que se produce al alcanzar una dirección específica del código después de que esta haya sido ejecutada un número determinado de veces; e inyección temporal, que consiste en introducir la falla dentro de un intervalo de tiempo predeterminado medido sobre la ejecución libre de errores.

FERRARI opera mediante la ejecución simultánea de dos procesos: un proceso de inyección de fallas y errores y un proceso que ejecuta el programa objetivo, comunicándose ambos mediante llamadas al sistema, lo que permite modificar de forma controlada el estado de ejecución del programa objetivo y determinar con precisión el instante y la ubicación en la que se realiza la inyección.

FERRARI se organiza de manera modular y está compuesta por cuatro módulos principales: un módulo de inicialización y activación, encargado de configurar el entorno de inyección y lanzar el programa objetivo; un módulo de información del usuario, responsable de definir los parámetros del experimento y las características de las fallas y errores a inyectar; un módulo de inyección de fallas y errores, que realiza la alteración del estado de ejecución del programa objetivo; y un módulo de recolección y análisis, destinado a gestionar los resultados durante los experimentos. La coordinación entre estos módulos es gestionada por un módulo central, que controla el flujo de información y la secuencia de operaciones durante el proceso de inyección.

³Sistema operativo que permite que varios usuarios lo utilicen al mismo tiempo (multiusuario) y que puede ejecutar múltiples procesos de manera simultánea (multiproceso) [55].

Una de las principales ventajas de FERRARI es que no requiere *hardware* especializado, lo que reduce costos y permite su uso sin depender de plataformas como FPGA. Sin embargo, su implementación se apoya en mecanismos específicos del sistema operativo *Unix*, lo que limita su portabilidad a otras arquitecturas, en particular a aquellas empleadas en sistemas embebidos.

8.1.5 Evaluador de tolerancia a fallas y desempeño en red (NFTAPE) [56]

El evaluador de tolerancia a fallas y desempeño en red (NFTAPE, por sus siglas en inglés) se caracteriza por una arquitectura modular que separa de manera funcional los componentes responsables de la FI, el disparo de las inyecciones, la generación de cargas de trabajo, la detección de errores y el registro de resultados (véase figura 8.3). Esta separación permite seleccionar y combinar de forma independiente distintos inyectores, disparadores y cargas de trabajo.

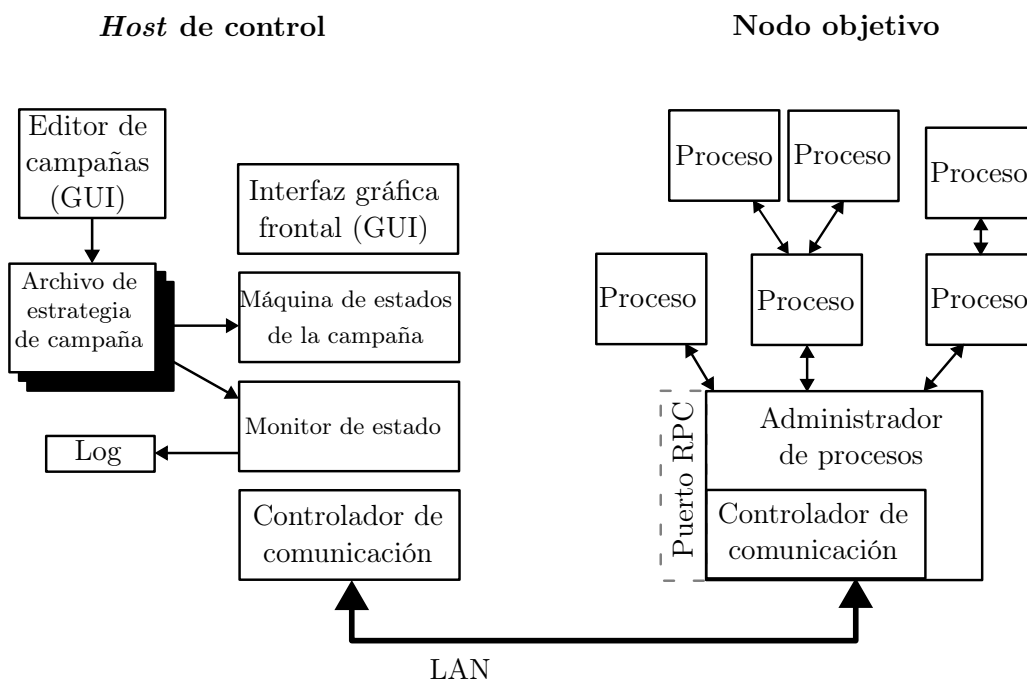


Figura 8.3. Arquitectura NFTAPE (adaptado de [56]).

La gestión de las campañas de FI recae en el *host* de control, componente central de la arquitectura NFTAPE, cuya función principal es procesar el *script* de estrategia de campaña. Dicho archivo especifica los parámetros necesarios para la ejecución de un experimento de FI, así como la secuencia general de acciones a realizar. El editor de campañas permite al usuario crear y modificar dicha estrategia, mientras que la interfaz gráfica frontal actúa como la interfaz de usuario de la herramienta.

Durante la ejecución de una campaña, el *host* de control recopila información en tiempo de ejecución mediante el monitor de estado, el cual recibe mensajes generados por el nodo objetivo y el administrador de procesos. Esta información es utilizada por la máquina de estados de la campaña para determinar de forma dinámica la secuencia de acciones del experimento, incluyendo la respuesta ante la detección de errores, así como la definición del número de repeticiones de cada ejecución experimental.

Para llevar a cabo la ejecución de la campaña, el *host* de control se apoya en el administrador de procesos, componente responsable de iniciar, supervisar y registrar todos los procesos involucrados en el experimento, tales como generadores de cargas de trabajo, monitores, disparadores e inyectores de fallas ligeros (LWFI, por sus siglas en inglés). Cada proceso y evento es etiquetado con identificadores únicos, lo que permite correlacionar la salida de los

procesos con la ejecución correspondiente, así como detectar condiciones anómalas, como bloqueos mediante el uso de *timeouts* o terminaciones inesperadas.

La FI se realiza mediante LWFI, los cuales son programas simples dedicados exclusivamente a provocar fallas y que no incorporan mecanismos de disparo ni lógica de detección de errores. El momento en el que se lleva a cabo la inyección es determinado por disparadores de fallas configurables de forma independiente, los cuales pueden basarse en temporizadores o *breakpoints*.

En cuanto a la evaluación de resultados, NFTAPE proporciona un marco flexible que permite medir distintas métricas de confiabilidad, como disponibilidad, cobertura o latencia de errores, adaptándose a los objetivos específicos de cada experimento. Sin embargo, la determinación de si una falla inyectada produce un error depende de los mecanismos de detección definidos por el usuario.

Desde una perspectiva comparativa, NFTAPE se distingue por su arquitectura modular, que facilita la portabilidad a plataformas como Solaris, Linux, Windows y LynxOS, y permite ejecutar experimentos de inyección de fallas por *software*, *hardware* o simulación. Si bien esta flexibilidad es una ventaja significativa, NFTAPE presenta algunas limitaciones: su enfoque está orientado principalmente a sistemas distribuidos en plataformas específicas, lo que puede dificultar su aplicación directa en sistemas embebidos o MCU. Además, la integración de nuevos métodos de inyección depende de contar con un LWFI compatible.

8.2 Herramientas de inyección de fallas por simulación

Las herramientas de FI por simulación se han consolidado como un recurso esencial en las primeras etapas del ciclo de diseño, cuando la validación del sistema aún no depende de *hardware* físico. Su principal ventaja es la posibilidad de inyectar fallas directamente en modelos de simulación, lo que permite reproducir escenarios críticos y evaluar la respuesta bajo condiciones controladas, facilitando la detección temprana de vulnerabilidades y el análisis de mecanismos de mitigación, además de reducir costos y tiempos asociados al desarrollo de prototipos físicos.

En los últimos años, la investigación ha impulsado el desarrollo de entornos de simulación más precisos y versátiles, capaces de modelar fallas transitorias y permanentes, integrarse con plataformas de verificación y adaptarse a arquitecturas heterogéneas, consolidando su papel como herramienta estratégica para garantizar la confiabilidad desde las fases iniciales del desarrollo.

8.2.1 Evaluación de la confiabilidad basada en VHDL mediante inyección eficiente de fallas (VERIFY) [57]

La evaluación de la confiabilidad basada en VHDL mediante inyección eficiente de fallas (VERIFY, por sus siglas en inglés) es una herramienta desarrollada en VHDL para la evaluación de parámetros de fiabilidad a partir de modelos descritos en el mismo lenguaje, tales como la confiabilidad del sistema y el tiempo medio entre fallos. Al estar completamente implementada en VHDL, VERIFY permite cubrir múltiples niveles de abstracción, desde el nivel de sistema y transferencia de registros hasta el nivel de compuertas, facilitando la evaluación del diseño a lo largo de sus distintas fases.

La principal contribución de VERIFY radica en la integración explícita de los parámetros de fallos dentro de la descripción comportamental de los componentes VHDL. Para cada componente, el comportamiento libre de fallas y el comportamiento bajo condiciones de falla se describen de manera conjunta, incorporando parámetros como el tiempo medio entre fallos, la duración de la falla y sus efectos. Esta integración directa en el propio modelo VHDL evita la necesidad de desarrollar modelos separados del sistema o emplear herramientas externas que reproduzcan su comportamiento, lo que resulta especialmente eficiente durante las fases tempranas del desarrollo.

Las fallas se describen mediante señales específicas, denominadas señales de inyección de fallas (FIS, por sus siglas en inglés), que permiten al simulador activar una falla determinada en un instante preciso. Cada modo de falla previamente identificado para un componente, es decir, los comportamientos defectuosos que pueden presentarse de acuerdo con su función, tecnología o experiencia previa, puede representarse mediante una señal independiente, sin necesidad de modificar la interfaz ni la estructura del componente, limitándose a la extensión de su descripción comportamental. Este enfoque permite, por ejemplo, modelar fallas como *stuck-at*, así como otros comportamientos defectuosos más complejos, dependiendo del conocimiento del diseñador o del fabricante del componente.

VERIFY incorpora un compilador que extrae automáticamente las señales de FI y sus parámetros asociados, los cuales son utilizados por el simulador para ejecutar de forma completamente automatizada los experimentos de inyección. Para reducir la sobrecarga computacional, la herramienta implementa una técnica de FI multihilo basada en una única ejecución de referencia libre de fallas (ejecución dorada). A partir de esta ejecución, se generan múltiples ejecuciones con fallas de corta duración, las cuales se comparan dinámicamente con la ejecución dorada hasta que ocurre la recuperación del sistema o se alcanza un tiempo límite.

Este mecanismo permite determinar de manera precisa el tiempo de recuperación del sistema asociado a cada falla inyectada, así como analizar la forma en que los errores se manifiestan y se propagan dentro del sistema. Adicionalmente, VERIFY permite excluir señales específicas durante el proceso de comparación de estados, lo cual resulta útil cuando ciertas variables, por ejemplo contadores de error, no regresan necesariamente a su valor original tras la recuperación.

No obstante, la necesidad de extender manualmente las descripciones comportamentales en VHDL para incorporar los mecanismos de falla puede aumentar la complejidad del modelo, especialmente en sistemas de gran escala o con un número elevado de modos de falla. Asimismo, la precisión de los resultados depende directamente de la fidelidad de los modelos de falla definidos por el diseñador, por lo que una caracterización incompleta o excesivamente simplificada podría limitar el grado de realismo de la evaluación de la confiabilidad.

VERIFY ha sido validada mediante la FI en procesadores RISC, demostrando su eficacia para evaluar la tolerancia a fallas sin requerir redundancia adicional. Actualmente, se utiliza en proyectos colaborativos como 3D-Smart-Pixels [57], en donde permite evaluar estrategias de tolerancia en sistemas optoelectrónicos e incluso modelar fallas específicas, como la diafonía, mediante extensiones VHDL.

8.2.2 Herramienta de simulación de inyección de fallas-errores multinivel (MEFISTO) [58]

La herramienta de simulación de inyección de fallas-errores multinivel (MEFISTO, por sus siglas en inglés) se presenta como un entorno integrado de FI en modelos VHDL, capaz de operar en distintos niveles del sistema. Entre sus principales aplicaciones se incluyen la estimación de la cobertura de mecanismos de tolerancia a fallas, la investigación de técnicas para el mapeo de resultados entre distintos niveles de abstracción, y la validación de modelos de falla empleados en experimentos de inyección realizados sobre la implementación de sistemas tolerantes a fallas, como SWIFI o la inyección a nivel de pines.

Una de las principales características de MEFISTO es su capacidad para inyectar fallas en múltiples niveles del sistema, desde señales individuales hasta elementos más complejos como variables, registros o subcomponentes, lo que permite analizar no solo la ocurrencia de una falla, sino también su propagación y su impacto funcional en el sistema. Para ello, la herramienta emplea dos mecanismos principales que requieren modificación del modelo: los *saboteurs* y los *mutants*, y además soporta técnicas de FI mediante comandos del simulador, como la manipulación de señales y variables.

Los *saboteurs* son componentes VHDL que permanecen inactivos durante el funcionamiento normal del sistema y se activan únicamente durante la FI, con el objetivo de alterar el valor o las características temporales de una o varias señales seleccionadas. Durante su activación, permiten inyectar fallas de manera controlada sin alterar la descripción funcional general del sistema.

Los *mutants* consisten en versiones modificadas de componentes que reemplazan temporalmente a los originales. Cuando están inactivos, se comportan de forma idéntica al componente original y, al activarse, reproducen el comportamiento del sistema bajo condiciones de falla, permitiendo modelar fallas tanto a nivel estructural como comportamental. Estos *mutants* pueden generarse mediante distintas estrategias: añadiendo *saboteurs* a descripciones de componentes estructurales o comportamentales, mutando descripciones estructurales de componentes mediante el reemplazo de subcomponentes, modificando automáticamente sentencias de descripciones comportamentales del componente, y realizando modificaciones manuales sobre descripciones del componente para implementar modelos de falla complejos y detallados.

La manipulación de señales consiste en inyectar fallas alterando directamente los valores de las señales del modelo VHDL, mientras que la manipulación de variables permite inyectar fallas en modelos comportamentales, modificando temporalmente los valores de las variables utilizadas dentro de los procesos VHDL. Ambos enfoques permiten analizar el impacto de fallas sin alterar la descripción funcional original del sistema.

La arquitectura de MEFISTO está organizada en tres fases que guían al usuario desde la definición de la campaña hasta el análisis de resultados. En la fase de configuración, la herramienta analiza el modelo VHDL del sistema objetivo y, a partir de este análisis, permite definir el conjunto de fallas, los nodos de inyección, los instantes de inyección y las salidas que se registrarán. Como resultado de esta fase, MEFISTO genera un modelo ejecutable del sistema y una lista de control de experimentos, la cual describe de manera precisa cómo debe ejecutarse cada experimento de la campaña.

En la fase de simulación, el modelo ejecutable generado durante la fase de configuración y la lista de control de experimentos se utilizan para ejecutar los experimentos de FI definidos en la campaña. Durante esta fase se controla la FI, la ejecución de cada experimento y el almacenamiento de los resultados necesarios para su posterior análisis. Finalmente, en la fase de procesamiento de datos, los resultados se procesan para obtener métricas de interés, como cobertura de detección de errores o medidas relacionadas con la propagación y latencia de los errores, dependiendo de los objetivos específicos de la campaña.

Una limitación de MEFISTO es que los resultados dependen directamente del nivel de detalle y de la precisión del modelo VHDL del sistema, así como de los modelos de falla utilizados durante la simulación. En consecuencia, la representatividad de los resultados puede verse limitada respecto al comportamiento real del *hardware* físico. A pesar de ello, MEFISTO constituye una solución flexible y escalable para la FI en entornos simulados, facilitando el análisis de la propagación de fallas y apoyando la evaluación de mecanismos de tolerancia a fallas en sistemas descritos en VHDL.

8.2.3 FOCUS [59]

FOCUS es un entorno experimental propuesto para el análisis de sensibilidad a fallas en diseños VLSI mediante simulaciones jerárquicas, permitiendo rastrear la propagación de errores desde el nivel de dispositivo hacia el nivel de compuertas y bloques funcionales del diseño, sin necesidad de prototipos físicos, lo que lo convierte en una herramienta valiosa durante las etapas iniciales del ciclo de diseño. El funcionamiento de FOCUS se apoya en el simulador *SPLICE*⁴, lo que facilita la observación sistemática de los efectos de las fallas a distintos niveles del diseño y el análisis de su propagación dentro del sistema objetivo.

El mecanismo de FI en FOCUS se implementa mediante la modificación del modelo del diseño durante la simulación, el cual es ejecutado por el simulador *SPLICE*, permitiendo introducir perturbaciones controladas que alteran temporalmente su comportamiento eléctrico. Este enfoque admite la inyección tanto de fallas transitorias como permanentes, con especial énfasis en el análisis de fallas transitorias, y ofrece al usuario control sobre la localización, el instante de ocurrencia y la duración de la falla. Asimismo, el entorno incorpora capacidades de trazado y análisis

⁴Simulador jerárquico de modo mixto empleado como motor de simulación en FOCUS, que integra análisis eléctrico y lógico para el estudio de diseños VLSI [59].

gráfico que facilitan el seguimiento de la propagación de errores y la evaluación de su impacto sobre compuertas, módulos funcionales e interconexiones relevantes del sistema.

En conjunto, FOCUS permite analizar los efectos de las fallas inyectadas y su manifestación en forma de errores dentro del diseño VLSI, mediante el uso de mecanismos de trazado y análisis. No obstante, como toda herramienta basada en simulación, los resultados dependen directamente de la fidelidad y del nivel de detalle de los modelos empleados, lo que debe considerarse al interpretar la respuesta del sistema.

8.3 Herramientas de inyección de fallas por depuración

Las herramientas de FI por depuración han consolidado su papel en el análisis de la tolerancia a fallas en MCU y sistemas embebidos, al permitir la modificación y supervisión del estado interno del dispositivo sin necesidad de intervención física directa. A través de interfaces estándar como JTAG y SWD, es posible acceder y manipular registros, memoria y el flujo de ejecución, permitiendo la FI de manera controlada y precisa.

Estas herramientas se distinguen por su elevada flexibilidad y alta reproducibilidad, lo que las convierte en un recurso clave para la validación del sistema en etapas avanzadas del desarrollo y durante las pruebas de *software*. En la actualidad, la investigación se orienta a ampliar la cobertura sobre interfaces de depuración protegidas, así como a integrar mecanismos de inyección automatizada con sistemas de pruebas continuas, con el objetivo de incrementar la eficiencia y la profundidad del análisis en entornos industriales y aeroespaciales.

8.3.1 Depuración en chip-inyección de fallas (OCD-FI) [60]

La propuesta de OCD-FI presentada en [60] amplía las capacidades tradicionales de los sistemas OCD mediante la integración de un módulo especializado que permite la inyección automática de fallas durante la ejecución del sistema. La infraestructura OCD-FI se implementa en VHDL con el objetivo de garantizar portabilidad y mantener un alto nivel de compatibilidad con múltiples arquitecturas, siendo especialmente adecuada para microprocesadores que incorporan mecanismos OCD compatibles con *nexus*, así como para modelos o diseños de microprocesadores descritos en VHDL e implementados en plataformas FPGA durante las fases de desarrollo y validación.

La arquitectura interna de la infraestructura OCD, mostrada en la figura 8.4, está compuesta por tres módulos principales: el módulo de mensajería y gestión de colas (MQM, por sus siglas en inglés), el módulo de control de ejecución y traza (RCT, por sus siglas en inglés) y el módulo de acceso de lectura y escritura (RWA, por sus siglas en inglés), además de los módulos de interfaz con el bus. Esta estructura modular permite diferenciar claramente las funciones de control, comunicación y acceso a los recursos de la CPU.

El módulo MQM gestiona la comunicación de depuración dentro de la infraestructura OCD, convirtiendo las operaciones del procesador en mensajes que pueden ser comprendidos por los otros módulos y controlando su flujo mediante colas y señales de control. El módulo RCT se encarga de supervisar la ejecución del procesador y generar información de traza, lo que permite detectar eventos relevantes como *breakpoints*, *watchpoints*, saltos y excepciones durante la ejecución de los programas. Finalmente, el módulo RWA proporciona acceso directo tanto a los registros del OCD como a otros recursos de la CPU, lo que permite leer y modificar la memoria y los registros del sistema de manera controlada.

La infraestructura OCD puede interactuar directamente con los distintos recursos conectados al bus del sistema. En la arquitectura de la figura 8.4, los módulos ROM, RAM y los puertos de E/S representan dichos recursos accesibles mediante los mecanismos de depuración. En particular, la ROM se utiliza para identificar direcciones de instrucción que actúan como disparadores de la inyección de fallas, mientras que el espacio RAM se emplea como objetivo principal para inyectar fallas del tipo *bit-flip* y la observación de su propagación en el sistema. Finalmente, el bloque de E/S representa periféricos o registros mapeados en memoria que también pueden ser accedidos a través del mismo mecanismo de depuración.

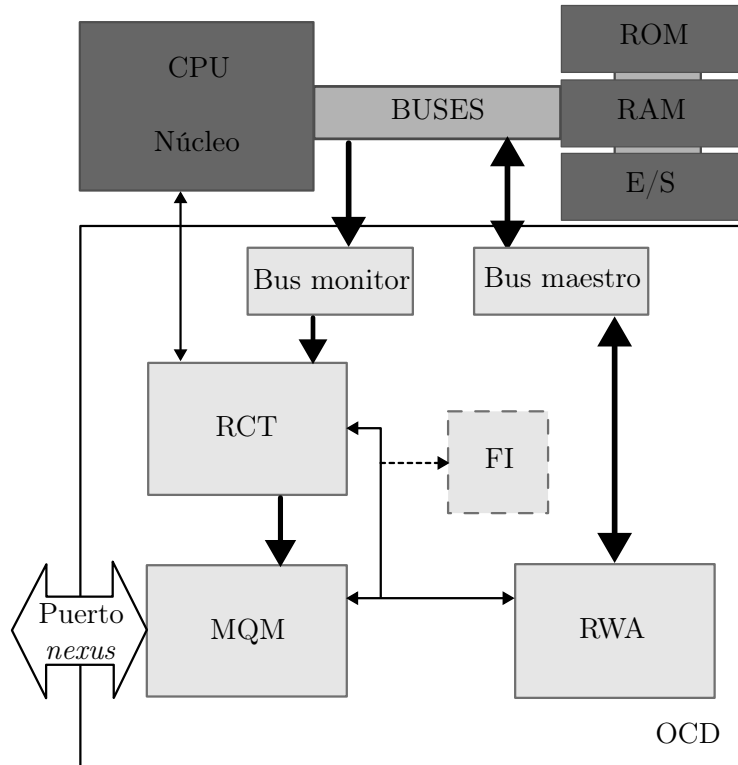


Figura 8.4. Infraestructura OCD (adaptado de [60]).

A diferencia de los enfoques convencionales, en los que la FI depende de un depurador externo, OCD-FI incorpora un módulo de FI directamente dentro de la infraestructura OCD. Este módulo monitorea eventos de disparo, como *watchpoints* o *breakpoints*, y ejecuta automáticamente la inyección sin necesidad de comandos adicionales del depurador. Como resultado, el retardo entre la detección del evento y la inyección se reduce de decenas de ciclos de reloj a apenas dos ciclos, mejorando significativamente la precisión temporal, la repetibilidad de las campañas de inyección y evitando que la CPU sobrescriba los datos objetivo antes de la inyección.

Entre sus principales ventajas se encuentran FI no intrusiva y de alta velocidad en la memoria durante la ejecución normal del procesador, así como el bajo impacto en los recursos de *hardware*, ya que OCD-FI reutiliza componentes y funciones que ya forman parte del módulo OCD, como las operaciones de lectura y escritura del módulo RWA, los registros internos del módulo OCD y las señales de *watchpoints* y *breakpoints* generadas por el módulo RCT, incorporando únicamente lógica adicional de control y coordinación. En particular, el módulo FI precarga direcciones y datos en los registros del módulo OCD y utiliza las señales de disparo para ejecutar automáticamente la inyección de fallas, sin necesidad de comandos adicionales del depurador. Asimismo, la propuesta ofrece flexibilidad para definir tanto los disparadores como las ubicaciones de las fallas, lo que permite adaptar la inyección a diferentes escenarios de prueba.

OCD-FI conserva la compatibilidad con el estándar *nexus*, lo que permite integrarlo en microprocesadores con infraestructura OCD modificable, ya que su implementación requiere poder añadir el módulo FI dentro del OCD existente. En microprocesadores comerciales con OCD cerrado, en donde no es posible realizar modificaciones, la implementación directa es limitada, quedando principalmente restringida a sistemas modelados o descritos en VHDL. En estos casos, el módulo FI puede implementarse en un FPGA que emule el procesador junto con la infraestructura OCD, recurso especialmente útil durante las etapas de desarrollo y validación.

8.3.2 Depuración en chip (OCD) empleando GDB [61]

El sistema propuesto en [61] tiene como objetivo evaluar la confiabilidad de procesadores COTS frente a fallas por *upsets* mediante campañas de FI realizadas directamente sobre el DUT. Para ello, utiliza la infraestructura de

depuración de *hardware* OCD, junto con el depurador estándar de *GNU Debugger* (GDB), permitiendo inyectar fallas durante la ejecución del programa, afectando tanto registros de propósito general como registros especiales y elementos de memoria. Adicionalmente, se emplea un módulo de temporizador de *hardware*, presente en la mayoría de los procesadores COTS actuales, con el fin de mejorar la eficiencia del proceso.

La herramienta tiene una alta portabilidad ya que es aplicable en procesadores físicos y puede operar también en entornos de simulación, emulación o máquinas virtuales, lo que permite su uso en distintas fases del desarrollo y la validación de sistemas, manteniendo un enfoque experimental de bajo costo, además de no requerir el diseño de modelos de *hardware* adicionales.

La arquitectura de la herramienta se organiza en cuatro componentes principales, como se muestra en la figura 8.5: el gestor de inyección de fallas (FIM, por sus siglas en inglés), el GDB, la puerta de enlace y la DI. El FIM ejecuta primero el programa sin realizar inyección de fallas para obtener una ejecución de referencia de la carga de trabajo. Además, recopila información valiosa de la ejecución, como las direcciones de inicio y fin del programa y el número total de pasos de ejecución. Con esta información, el FIM se encarga de gestionar las campañas de inyección de fallas, generar automáticamente los *scripts* de inyección para GDB a partir de los parámetros definidos por el usuario, y analizar y clasificar los resultados.

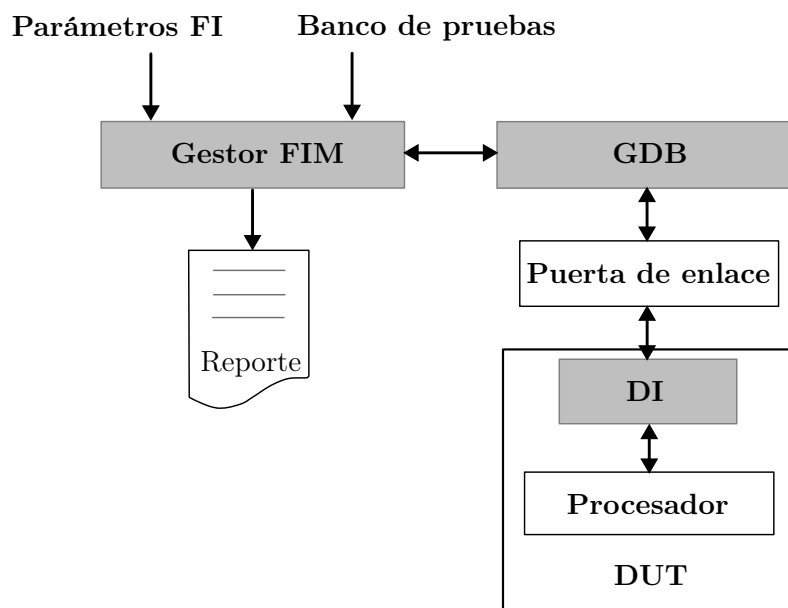


Figura 8.5. Arquitectura OCD-GDB (adaptado de [61]).

La puerta de enlace, diseñada específicamente para el DUT, gestiona la comunicación entre el GDB y el sistema objetivo. En el caso de microprocesadores COTS, este componente se ejecuta en el sistema anfitrión, típicamente una computadora para controlar las campañas de FI, mientras que en entornos simulados o emulados forma parte del propio simulador o emulador. La DI implementa los comandos de depuración del GDB, como la carga de código, la ejecución paso a paso y la inserción de *breakpoints*. En sistemas COTS, esta función es desempeñada por la propia infraestructura OCD del procesador, mientras que en simuladores y emuladores se utiliza un *GDB stub software*, es decir, una interfaz de depuración implementada en *software* que emula las funcionalidades de la infraestructura OCD y permite la comunicación entre GDB y el modelo del procesador.

La FI se basa en la modificación de un único bit mediante fallas de tipo *bit-flip*, aplicadas de forma aleatoria tanto en el tiempo como en la localización. El procedimiento consiste en leer la ubicación y el contenido del recurso interno del procesador en donde se inyectará la falla (como memoria o registros), aplicar una operación XOR con una máscara de falla y escribir el valor modificado de vuelta en la misma localización. Los efectos de las fallas se clasifican en categorías como UnACE, SDC y *hang*.

Entre sus principales ventajas se encuentran la alta portabilidad a diferentes arquitecturas de procesadores y tipos de implementación (softcore y hardcore), el bajo costo de implementación, la ausencia de *hardware* especializado adicional y la flexibilidad para adaptarse a múltiples plataformas físicas o virtuales compatibles con el GDB. En entornos de simulación como QEMU⁵, la herramienta permite acelerar las campañas de inyección y obtener estimaciones precisas de confiabilidad.

8.3.3 Depuración en chip empleando JTAG [63]

En [63] se presenta un entorno de FI diseñado para procesadores que incorporan una infraestructura OCD accesible mediante la interfaz estándar JTAG, implementado en *hardware* sobre FPGA para controlar la campaña de FI de manera autónoma. Este enfoque permite acelerar significativamente el proceso de evaluación y eliminar la necesidad de comunicación continua con una computadora externa, estableciéndose únicamente al final de la campaña, reduciendo así la sobrecarga. Además, al tratarse de un método no intrusivo que no requiere modificar el sistema objetivo, y dada la presencia de JTAG en microprocesadores actuales, resulta aplicable a una amplia variedad de procesadores.

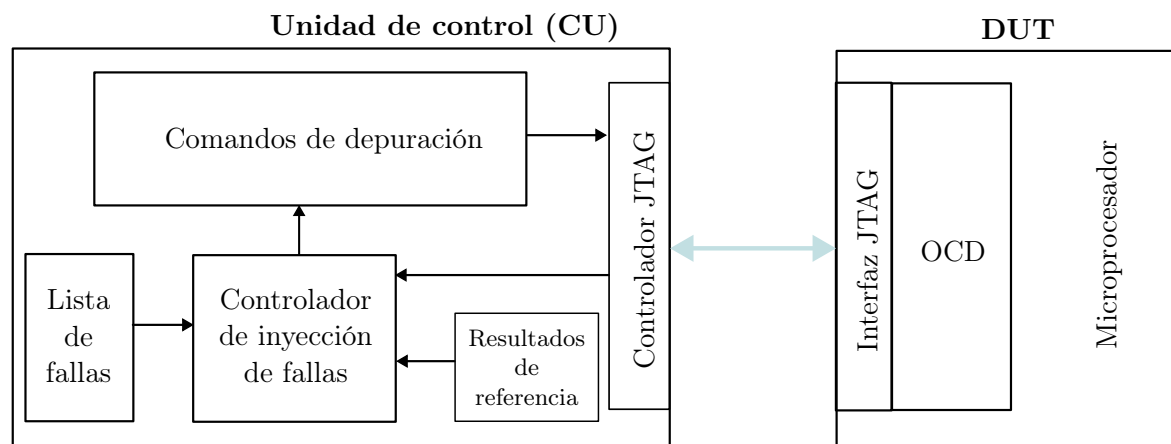


Figura 8.6. Arquitectura OCD-JTAG (adaptado de [63]).

La arquitectura del sistema, mostrada en la figura 8.6, se organiza en torno a una unidad de control (CU, por sus siglas en inglés) que integra los siguientes módulos:

- **Lista de fallas:** Bloque de memoria que almacena la ubicación y el instante de inyección de cada falla, definidos mediante puntos de interrupción (*breakpoints*), y leídos por el controlador de inyección para ejecutar la campaña.
- **Resultados de referencia libres de fallas:** Memoria que almacena los resultados durante ejecuciones libres de fallas, necesarios para determinar y clasificar los efectos de las fallas.
- **Controlador JTAG:** Encargado de gestionar la interfaz del puerto de acceso de prueba (TAP⁶, por sus siglas en inglés), controlando la secuencia de operaciones de los registros de prueba (registros internos del JTAG/TAP utilizados para depuración y prueba) del procesador.
- **Controlador de inyección de fallas:** Máquina de estados finita que coordina la campaña, lee la lista de fallas, emite los comandos OCD necesarios, compara los resultados con los de referencia y clasifica los efectos observados.
- **Módulo de comandos de depuración:** Bloque que contiene las secuencias de comandos JTAG necesarias para implementar las capacidades de OCD, tales como la lectura y escritura de registros y memoria, la configuración de *breakpoints*, el reinicio de la ejecución y el control del flujo del programa.

⁵Es una herramienta de virtualización y emulación de *hardware* de código abierto, ampliamente utilizada para simular diferentes arquitecturas y plataformas de cómputo [62].

⁶Es el conjunto de señales definido en el estándar IEEE 1149.1, diseñado para facilitar y automatizar el test de circuitos impresos sin requerir acceso físico directo mediante el uso de registros de prueba integrados en los dispositivos [64].

El flujo de inyección comprende una fase de configuración inicial, una ejecución de referencia libre de fallas, la lectura y modificación controlada de valores en las ubicaciones internas del procesador en instantes de tiempo específicos, y la posterior comparación con los resultados de referencia.

Una de las características más relevantes de esta propuesta es su elevada adaptabilidad, ya que la extensión a distintos procesadores se logra mediante la modificación de parámetros genéricos del estándar JTAG, como la longitud del registro de instrucciones y los códigos de instrucción, así como mediante la actualización de las secuencias de comandos del módulo de depuración. Esto la posiciona como una herramienta portable, aplicable a cualquier procesador que soporte un OCD con interfaz JTAG, para evaluar la sensibilidad a fallas de tipo *bit-flip* en microprocesadores modernos, constituyendo una alternativa competitiva frente a enfoques puramente por simulación o en instrumentación intrusiva, sin requerir modificaciones en el sistema objetivo.

8.4 Análisis de las herramientas de inyección de fallas

Las herramientas por *software*, como GOOFI, MiFIT o FERRARI, representan una alternativa accesible y flexible, ya que permiten introducir fallas mediante distintos mecanismos, como interrupciones, excepciones o la modificación de registros y posiciones de memoria. Esto facilita la experimentación sin requerir infraestructura física compleja; sin embargo, su carácter intrusivo puede modificar el comportamiento natural del sistema, debido a que algunas técnicas requieren alterar el código o el entorno de ejecución del DUT.

Las herramientas de simulación como FOCUS, MEFISTO o VERIFY permiten inyectar fallas transitorias o permanentes en modelos de simulación, lo que posibilita la exploración de distintos escenarios en fases tempranas del diseño y la detección de vulnerabilidades antes de la implementación en *hardware*. No obstante, su principal limitación radica en que la precisión de los resultados depende de la fidelidad del modelo utilizado, así como del costo computacional asociado a las simulaciones.

Finalmente, las herramientas de depuración como OCD-FI, OCD empleando JTAG y OCD empleando GDB permiten modificar de manera controlada el estado interno del sistema mediante interfaces de depuración. Estos mecanismos posibilitan la inyección de *bit-flips* sobre registros o posiciones de memoria del DUT, permitiendo realizar experimentos directamente sobre *hardware* físico sin requerir modificaciones en el código del programa. Sin embargo, la detención temporal del procesador durante la inyección puede introducir intrusividad temporal en la ejecución.

En síntesis, las herramientas analizadas presentan diferencias en cuanto a accesibilidad, intrusividad, control y representatividad. Las herramientas por *software* ofrecen flexibilidad y facilidad para la ejecución de campañas, aunque pueden requerir modificaciones en el programa o en el entorno de ejecución; las herramientas de simulación permiten un alto grado de control sobre los experimentos, pero sus resultados dependen de la fidelidad del modelo empleado; mientras que las herramientas de depuración permiten realizar alteraciones controladas directamente sobre el estado interno del DUT, constituyendo una alternativa adecuada para la simulación de efectos SEU en memoria y registros de una MCU.

En la tabla 8.1 se muestra un análisis comparativo de las herramientas de FI estudiadas, considerando sus principales características, formas de implementación y limitaciones propias de cada técnica.

Tabla 8.1. Comparativa entre las diferentes herramientas de inyección de fallas.

Herramienta	Sistema objetivo	Tipo de fallas	Implementación
GOOFI	Microprocesadores	<i>Bit-flip</i>	<i>software</i>
GOOFI-2	Microprocesadores	<i>Bit-flip</i>	<i>software</i>
MiFIT	Microprocesadores	<i>Bit-flip</i>	<i>software</i>
THESIC	Circuitos integrados	<i>Bit-flip</i>	<i>software</i>

Continúa en la siguiente página

Tabla 8.1. (Continuación) Comparativa entre la diferentes herramientas de inyección de fallas.

Herramienta	Sistema objetivo	Tipo de fallas	Implementación
FERRARI	Sistemas operativas	<i>Bit-flip</i>	<i>software</i>
NFTAPE	Sistemas distribuidos	<i>Bit-flip</i>	<i>software</i>
FOCUS	Modelos de simulación	Transitorias, permanentes	Simulación
MEFISTO	Modelos VHDL	Transitorias	Simulación
VERIFY	Modelos VHDL	Transitorias y permanentes	Simulación
OCD-FI	Microprocesadores, modelos VHDL	<i>Bit-flip</i>	Depuración
OCD empleando GDB	Microprocesadores, modelos de simulación	<i>Bit-flip</i>	Depuración
OCD empleando JTAG	Microprocesadores	<i>Bit-flip</i>	Depuración

8.5 Conclusiones del estado del arte

De acuerdo con el análisis de las herramientas existentes, se identificó que, aunque algunas son capaces de modificar el contenido de la memoria y de los registros internos, no todas permiten realizar estas alteraciones directamente sobre una MCU física sin modificar el *software* bajo prueba o depender de un modelo de simulación. Asimismo, aspectos como la automatización de las campañas y la reproducibilidad de los experimentos continúan siendo elementos relevantes para el desarrollo de este tipo de herramientas.

En este sentido, el uso de técnicas de depuración resulta especialmente adecuado, ya que permite acceder de manera controlada al estado interno de una MCU mediante interfaces de depuración disponibles en el propio dispositivo. Esto permite realizar modificaciones en memoria y registros sobre el *hardware* físico, manteniendo sin cambios el programa ejecutado por el DUT y facilitando la automatización de las inyecciones.

A partir de lo anterior, se identifica la necesidad de desarrollar una herramienta que no dependa exclusivamente de una MCU específica, sino que pueda adaptarse a diferentes dispositivos basados en Cortex-M, manteniendo un mecanismo común para la configuración y ejecución de las campañas de inyección. También es necesario contar con mecanismos de selección de los elementos sobre los cuales realizar las inyecciones, automatizar su ejecución, registrar los resultados y garantizar la reproducibilidad de los experimentos. Estos elementos constituyen la base para definir los requisitos y orientar el diseño de la herramienta desarrollada en este trabajo.

SECCIÓN III
DISEÑO

9. DISEÑO CONCEPTUAL

El diseño conceptual constituye la etapa inicial en el desarrollo de un sistema o proyecto de ingeniería, en donde se definen las ideas fundamentales, la arquitectura general y los principios de funcionamiento, con el objetivo de traducir los requisitos y metas del proyecto en una solución técnica sin entrar todavía en los detalles de implementación, permitiendo así establecer una base sólida que guíe las fases posteriores del desarrollo.

En este proyecto, el diseño conceptual se enfoca en la definición de los requisitos y en la descripción general del sistema, identificando los elementos que lo conforman, las funciones asociadas a cada uno y la relación entre ellos. Esta visión permite comprender la integración del sistema para cumplir los objetivos de inyección, monitoreo y registro de fallas en una MCU, así como establecer la estructura funcional y operativa que servirá como referencia para las etapas siguientes de diseño.

9.1 Requisitos del sistema

Los requisitos del sistema constituyen la base para el diseño y desarrollo de cualquier proyecto de ingeniería, ya que corresponden a las necesidades y condiciones que el sistema debe satisfacer con el fin de cumplir con los objetivos propuestos. De acuerdo con [65], los requisitos se clasifican en funcionales y no funcionales.

Los requisitos funcionales corresponden a aquellos que especifican las acciones que debe realizar el sistema, es decir, describen su comportamiento observable ante entradas determinadas y situaciones concretas. Por otro lado, los requisitos no funcionales definen cómo debe ejecutarse el sistema, estableciendo características relacionadas con el desempeño, la confiabilidad, la seguridad y las restricciones de diseño. En el caso del inyector de fallas, los requisitos funcionales establecen las capacidades específicas de inyección, monitoreo y configuración, mientras que los no funcionales definen las condiciones operativas necesarias para su correcto funcionamiento.

Para definir estos requisitos, es necesario precisar el alcance del proyecto, el cual se limita a la inyección controlada de fallas en la memoria RAM, en los registros de configuración de los periféricos y en una memoria *flash* simulada mediante la ejecución del *firmware* desde RAM. Estos recursos permiten evaluar el efecto de alteraciones controladas tanto sobre datos y registros internos como sobre información asociada con la ejecución del *software*.

A continuación, en la tabla 9.1 se presentan los requisitos de la herramienta de FI, clasificados en funcionales y no funcionales, junto con su método de verificación.

Tabla 9.1. Requisitos de la herramienta de inyección de fallas.

Requisito	Tipo	Método de verificación
Inyectar fallas en RAM.	Funcional	Los <i>bits</i> /palabras objetivo muestran la modificación esperada.
Inyectar fallas en registros de configuración de periféricos.	Funcional	La lectura de los registros confirma la alteración.
Inyectar fallas en la memoria <i>flash</i> simulada en RAM.	Funcional	Los <i>bits</i> /palabras objetivo muestran la modificación esperada.
Registrar la información asociada a cada inyección, incluyendo su identificador, tipo de falla, dirección y recurso afectado.	Funcional	El historial de la campaña contiene los campos definidos para cada inyección.

Continúa en la siguiente página

Tabla 9.1. (Continuación) Requisitos de la herramienta de inyección de fallas.

Requisito	Tipo	Método de verificación
Registrar los valores del recurso objetivo antes y después de cada inyección.	Funcional	Los archivos de salida contienen el valor previo y el valor posterior asociados al identificador de cada inyección.
La herramienta debe permitir seleccionar el tipo de falla a inyectar.	Funcional	Se ejecutan campañas con el tipo de falla seleccionada y se confirma la correcta aplicación en los registros o memorias seleccionadas.
La herramienta debe permitir acceder y controlar el estado interno del DUT durante las campañas de inyección.	No funcional	Se verifica que la herramienta permita consultar y modificar los recursos definidos dentro del alcance y controlar la ejecución del DUT.
La herramienta debe garantizar la repetibilidad de las campañas de inyección.	No funcional	La repetición de una misma campaña de inyección bajo las mismas condiciones de configuración produce resultados consistentes.
El sistema debe mantener la integridad de la MCU bajo prueba, evitando escrituras no autorizadas en memoria <i>flash</i> .	No funcional	Se verifica que las operaciones de inyección se realicen únicamente sobre los recursos definidos dentro del alcance del sistema.
El sistema debe contar con una interfaz de usuario para configurar campañas y visualizar los resultados.	Funcional	El usuario puede definir parámetros desde la GUI y visualizar los datos de inyección.
La herramienta debe ser portable y compatible con diferentes placas de desarrollo basadas en MCU de la misma arquitectura.	No funcional	Prueba de conexión, control y ejecución básica en al menos una placa de desarrollo distinta a la de referencia.

9.2 Descripción del sistema

La descripción del sistema tiene como objetivo presentar una visión general de su estructura y funcionamiento, con lo cual se establecen las bases conceptuales necesarias para comprender su arquitectura y permitir, a su vez, identificar los elementos principales que lo conforman, ofreciendo un marco general que servirá como base para el análisis más detallado de los componentes y de su interacción en etapas posteriores.

En el desarrollo de este proyecto, la descripción del sistema se centra en una herramienta de inyección de fallas para MCU, cuyo propósito consiste en inyectar fallas controladas y analizar el comportamiento del dispositivo frente a eventos transitorios generados por la radiación espacial, considerando principalmente los efectos *upset*, ya que constituyen una de las manifestaciones más frecuentes de los SEU en órbitas bajas LEO [66].

Para cumplir con este propósito, la herramienta debe permitir definir las condiciones de una campaña de inyección, seleccionar los recursos que serán modificados, introducir las fallas correspondientes, supervisar el desarrollo de las pruebas y registrar la información generada para su posterior análisis.

De manera conceptual, el sistema se encuentra conformado por una computadora de control y un DUT. La computadora de control representa el entorno desde el cual se configura y opera la herramienta de FI, mientras que el DUT corresponde a la MCU sobre la cual se realizan las inyecciones y se observa el comportamiento resultante. La relación general entre ambos elementos se muestra en la figura 9.1.

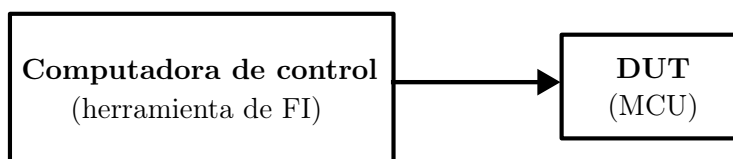


Figura 9.1. Diagrama conceptual

10. DISEÑO A NIVEL SISTEMA

El diseño a nivel sistema corresponde a una etapa intermedia en el desarrollo de un sistema o proyecto de ingeniería, en la cual se define su arquitectura, se identifican los principales elementos que lo conforman y se establece la forma en que estos interactúan entre sí. De esta manera, esta etapa sirve como enlace entre el diseño conceptual y el diseño de detalle, proporcionando una visión estructurada del sistema antes de definir los elementos específicos de implementación.

En el desarrollo de la herramienta de FI, el diseño a nivel sistema parte de la relación conceptual establecida entre la computadora de control y el DUT. A partir de esta base se identifican los elementos necesarios para permitir la interacción entre ambos, realizar las operaciones de inyección y supervisar el comportamiento del dispositivo durante la ejecución de las campañas de FI.

10.1 Arquitectura a nivel sistema

La arquitectura a nivel sistema establece los elementos necesarios para hacer posible la interacción entre la computadora de control y el DUT definida durante el diseño conceptual. Para ello, se consideran las características de los procesadores Cortex-M descritas previamente, en los cuales la memoria y los registros asociados a los periféricos se encuentran mapeados dentro del espacio de direcciones de la MCU, mientras que los recursos de OCD integrados en la MCU permiten realizar operaciones de lectura y escritura sobre estos elementos, además de controlar temporalmente la ejecución del procesador.

La arquitectura a nivel sistema se conforma por tres elementos principales: una computadora de control, una unidad de depuración y el DUT, como se muestra en la figura 10.1.

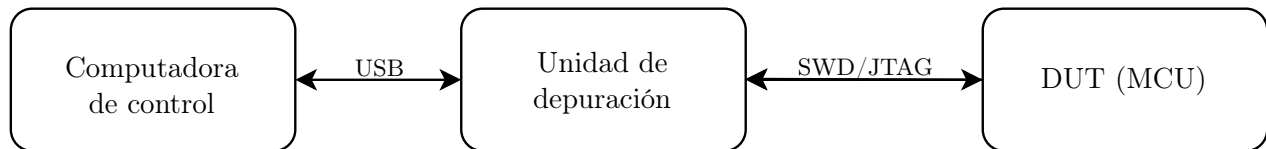


Figura 10.1. Diagrama del sistema.

Computadora de control

La computadora de control constituye el elemento encargado de gestionar la operación general de la herramienta de FI y coordinar la ejecución de las campañas de inyección. Sus principales funciones son:

- Definir los parámetros de la prueba de inyección.
- Seleccionar el tipo de falla que se desea inyectar.
- Enviar los comandos de inyección al DUT.
- Recopilar los resultados durante la campaña de FI.
- Almacenar los resultados para su posterior análisis.

De esta manera, la computadora de control constituye el punto principal de interacción con la herramienta, concentrando las funciones necesarias para configurar, ejecutar y supervisar las campañas de FI, así como organizar la información obtenida para su posterior análisis.

Unidad de depuración

La unidad de depuración constituye el elemento intermediario entre la computadora de control y el DUT. Su función principal es establecer la comunicación con la MCU y transmitir las operaciones requeridas para acceder y controlar los recursos internos del dispositivo.

La conexión con la computadora de control se realiza mediante USB, mientras que la comunicación con la MCU se establece a través de una interfaz de depuración, como JTAG o SWD. Mediante esta unidad se aprovechan los recursos OCD disponibles en el DUT para controlar temporalmente la ejecución del procesador y realizar operaciones de lectura y escritura sobre memoria y registros.

Dentro de la arquitectura propuesta, estas capacidades permiten que la herramienta de FI acceda a los recursos definidos como objetivos de inyección y realice las modificaciones correspondientes de manera controlada.

Dispositivo bajo prueba (DUT)

El DUT corresponde a la MCU sobre el cual se aplican las campañas de FI, ejecutando el *software* de aplicación bajo estudio, con el objetivo de analizar su comportamiento y su capacidad de respuesta ante condiciones que afectan su correcto funcionamiento.

Durante la operación de la herramienta, el DUT se encuentra en comunicación permanente con la computadora de control a través de la unidad de depuración, utilizando la DI correspondiente, lo que permite supervisar el estado de sus registros y memoria, observar los efectos de las fallas inyectadas y registrar los resultados experimentales para su posterior análisis.

10.2 Arquitectura de alto nivel de la herramienta de inyección de fallas (FI)

La arquitectura de alto nivel de la herramienta de FI describe la organización conceptual del sistema y la interacción entre sus distintos módulos, proporcionando una visión global de su funcionamiento sin abordar detalles específicos de implementación, los cuales se desarrollan en etapas posteriores.

La definición de esta arquitectura permite identificar de manera clara las etapas fundamentales del proceso de FI, entre las que se encuentran la configuración de campañas por parte del usuario, la comunicación con el DUT, la inyección controlada de fallas y el registro y análisis de los resultados (véase la figura 10.2).

Interfaz gráfica de usuario (GUI)

La GUI constituye el módulo de la herramienta de FI encargado de la interacción con el usuario, permitiéndole definir los parámetros de entrada que son utilizados para la generación de las campañas de inyección, como el tipo de falla y la cantidad de fallas a inyectar, además de seleccionar el MCU objetivo.

Asimismo, la GUI permite visualizar los resultados, presentando la información almacenada de forma estructurada para facilitar su análisis. En consecuencia, este módulo desempeña una función fundamental dentro de la herramienta, ya que constituye el punto de entrada para la configuración de las pruebas y el medio principal para la supervisión del comportamiento del sistema objetivo durante la ejecución de las campañas de inyección.

Módulo de configuración

El módulo de configuración adapta el funcionamiento de la herramienta de FI a la MCU seleccionada por el usuario, ajustando automáticamente los parámetros de operación en función de las características del dispositivo. Esto incluye la definición del mapa de memoria y las restricciones de acceso a registros, asegurando que las alteraciones se realicen únicamente en zonas válidas y representativas del *software* bajo prueba. Además, permite que la herramienta sea compatible con distintas MCU que compartan la misma arquitectura, adaptándose correctamente a sus recursos disponibles.

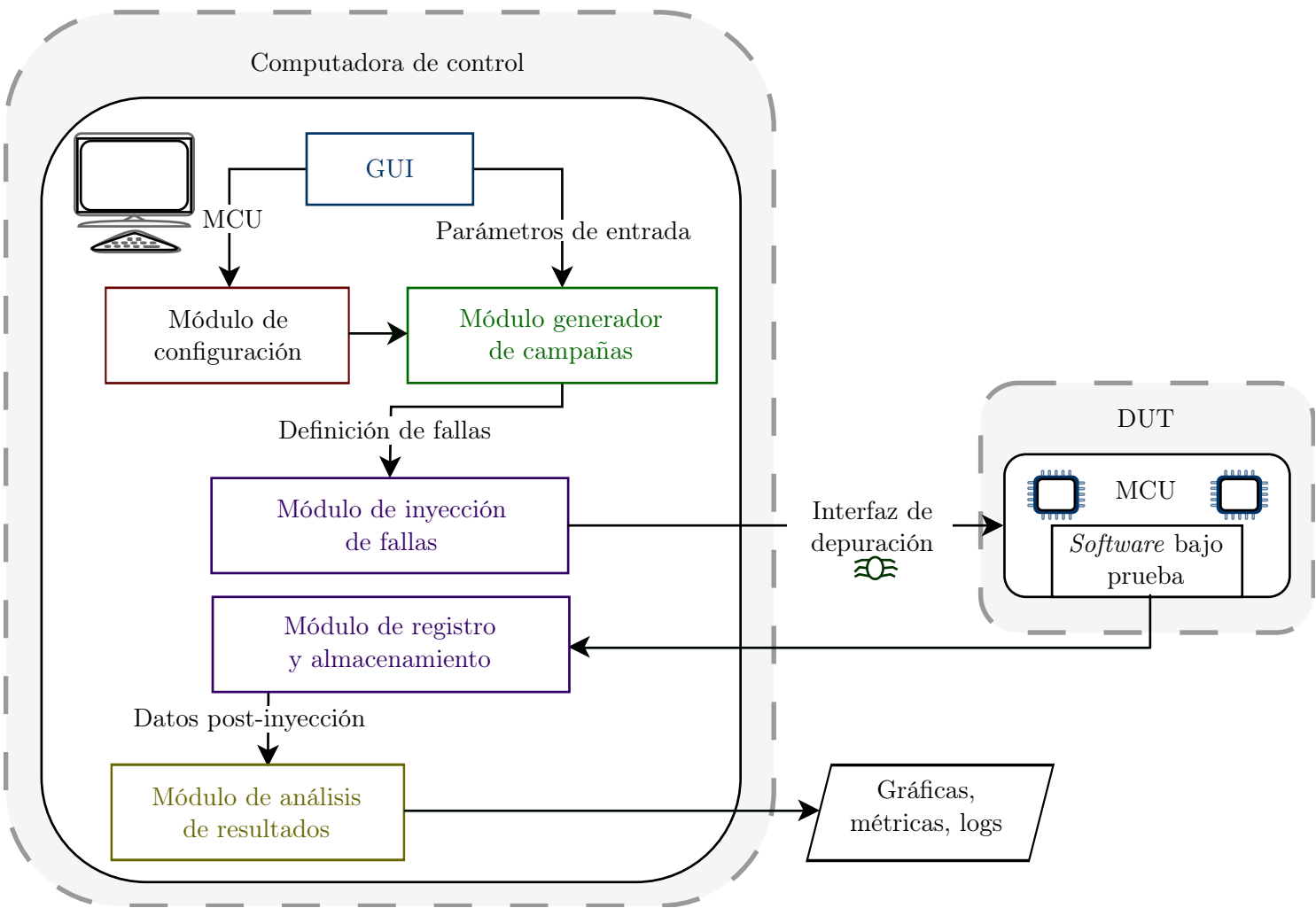


Figura 10.2. Arquitectura de alto nivel de abstracción de la herramienta de inyección de fallas.

Módulo generador de campañas

El módulo generador de campañas genera las listas de fallas que se aplican durante las pruebas, utilizando los parámetros establecidos por el usuario en la GUI y la información técnica proporcionada por el módulo de configuración. Cada falla generada cumple con las restricciones de la MCU y puede ser aplicada por el módulo de inyección de fallas.

Módulo de inyección de fallas

El módulo de inyección de fallas constituye el núcleo operativo de la herramienta, encargado de aplicar las alteraciones controladas sobre el DUT de acuerdo con las fallas definidas previamente. Para ello, traduce cada falla en comandos enviados a la MCU a través de la DI, modificando palabras de memoria y registros únicamente en las regiones válidas y representativas del dispositivo.

Módulo de registro y almacenamiento

El módulo de registro y almacenamiento captura y organiza la información generada durante la ejecución de las campañas de inyección, incluyendo los valores previos y posteriores a cada alteración y los parámetros asociados a cada prueba, asegurando la conservación estructurada de los datos experimentales y constituyendo la base para las etapas posteriores de análisis.

Módulo de análisis de resultados

El módulo de análisis de resultados procesa la información almacenada para facilitar la interpretación del comportamiento de la MCU ante las inyecciones realizadas, permitiendo agrupar, clasificar y representar los resultados mediante tablas y gráficas, y constituyendo la etapa final del flujo de operación de la herramienta al transformar los datos experimentales en información que permite evaluar el desempeño del DUT frente a la FI.

Software bajo prueba

El *software* de aplicación, o *software* bajo prueba, se ejecuta en el MCU y constituye el objetivo de las campañas de inyección de fallas; aunque no forma parte de la herramienta como un módulo, representa un elemento clave que permite evaluar cómo responde el DUT ante perturbaciones de manera controlada.

11. DISEÑO DE DETALLE E IMPLEMENTACIÓN

El diseño de detalle define con precisión la estructura interna del sistema, sus módulos y sus interacciones, partiendo del diseño conceptual. En esta etapa, los bloques funcionales se traducen en elementos concretos de *hardware* y *software*, especificando cómo se implementan, conectan y operan de manera conjunta para cumplir con los requisitos del sistema.

El diseño de detalle de la herramienta desarrolla los elementos definidos en el diseño conceptual, estableciendo su implementación y funcionamiento a nivel de *hardware* y *software*. Se describen las decisiones de diseño, la estructura interna de los módulos y los mecanismos utilizados para la FI de tipo *bit-flip*, *stuck-at-0* y *stuck-at-1*. De esta forma, se ofrece una visión completa de cómo cada módulo contribuye al control y monitoreo de fallas en la MCU.

11.1 Selección del microcontrolador y herramientas de desarrollo

La selección de la MCU y de las herramientas de desarrollo constituye un aspecto crítico, ya que determina el nivel de acceso a la memoria y los registros internos del DUT, las capacidades de depuración disponibles y la viabilidad de implementar mecanismos de FI mediante depuración.

En este contexto, se seleccionó la MCU STM32F407VGT6 debido a su utilización en aplicaciones relacionadas con sistemas satelitales mexicanos, particularmente en el desarrollo del sistema satelital SCMI asociado a la misión GuaraniSat-2 [67]. Esta selección permite que la herramienta de FI se desarrolle sobre un dispositivo con aplicación en el ámbito espacial, manteniendo además las capacidades de depuración necesarias para implementar las técnicas de inyección consideradas. La STM32F407VGT6 integra un núcleo ARM Cortex-M4 y se encuentra incorporada en la tarjeta de desarrollo STM32F4-Discovery de STMicroelectronics.

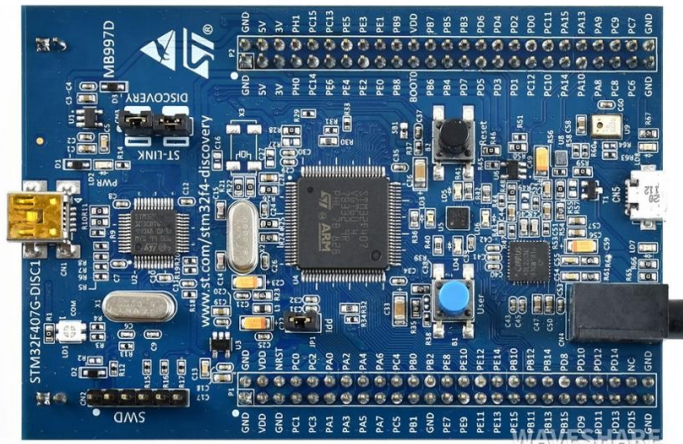


Figura 11.1. Tarjeta de desarrollo STM32F4-Discovery [68].

La tarjeta STM32F4-Discovery integra el programador y depurador ST-LINK/V2, que actúa como unidad de depuración entre la computadora anfitriona y la MCU. La comunicación con el microcontrolador se realiza mediante la interfaz SWD, permitiendo acceder a los recursos internos de depuración del dispositivo.

El acceso a la MCU se realiza mediante la herramienta pyOCD, desarrollada en Python y compatible con diferentes sondas de depuración, entre ellas ST-LINK. En este trabajo, pyOCD establece la comunicación con la MCU a través del ST-LINK/V2 y de la interfaz SWD. Mediante su API es posible controlar la ejecución del núcleo, así como consultar y modificar registros y posiciones de memoria, permitiendo automatizar el proceso de inyección de fallas.

Finalmente, para el desarrollo del *software* bajo prueba se utiliza STM32CubeIDE exclusivamente para la implementación y compilación del programa en la MCU. La ejecución de las campañas de FI se realiza de manera

independiente mediante pyOCD, desacoplando el entorno de desarrollo del experimental. Esta separación garantiza un mayor control sobre el proceso de inyección y mejora la reproducibilidad de los experimentos.

11.2 Implementación de la herramienta de inyección de fallas

El diseño de la FI sigue un enfoque modular con el objetivo de garantizar su compatibilidad con distintas MCU y escenarios de prueba. Cada submódulo implementa una función específica dentro del proceso de inyección, permitiendo la ejecución controlada de campañas y el análisis de los efectos de las fallas inyectadas sobre el sistema objetivo.

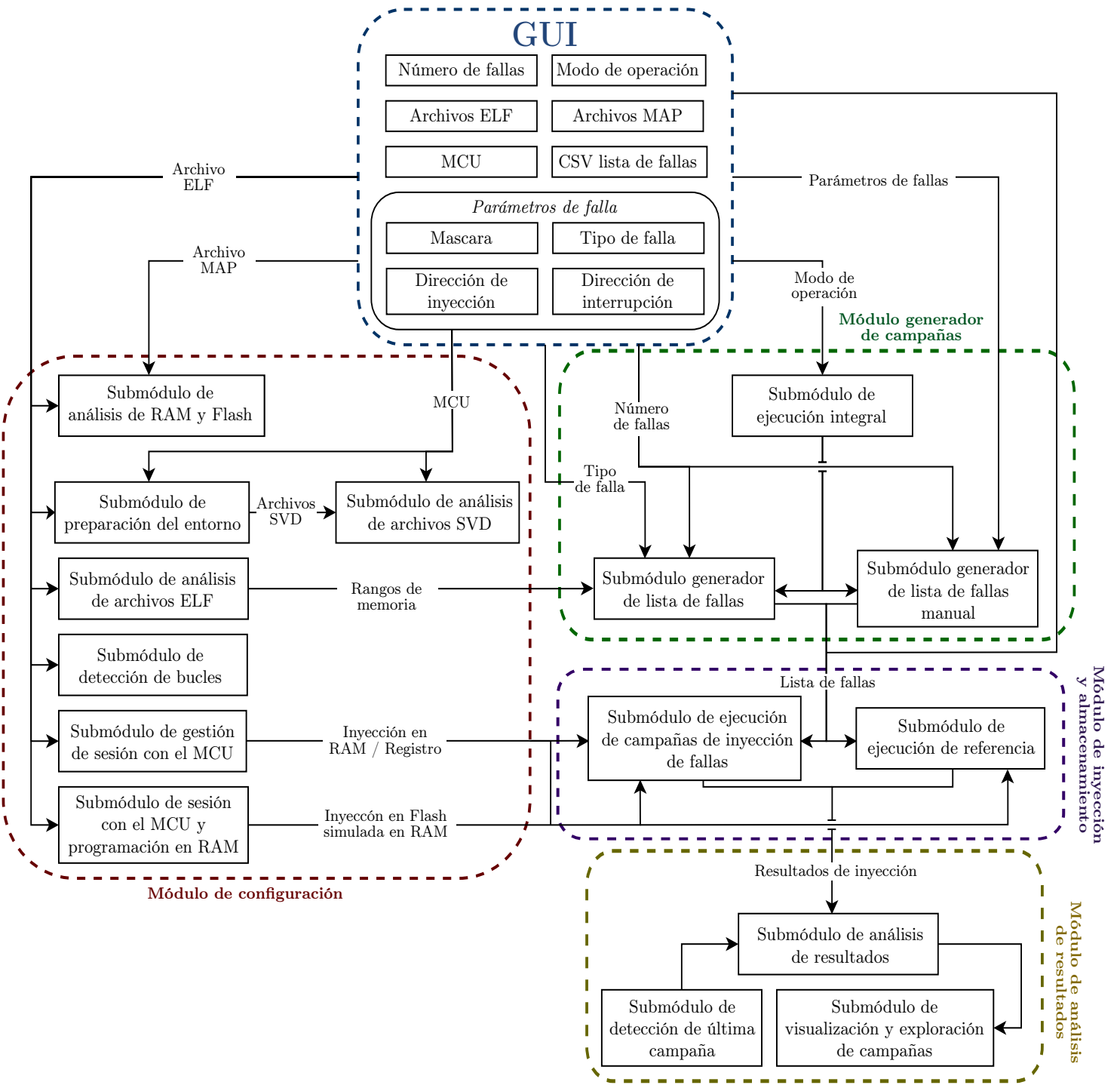


Figura 11.2. Diseño de detalle de la herramienta de inyección de fallas.

El diseño de la FI sigue un enfoque modular con el objetivo de garantizar su compatibilidad con distintas MCU y escenarios de prueba. Cada submódulo implementa una función específica dentro del proceso de inyección, permitiendo la ejecución controlada de campañas y el análisis de los efectos de las fallas inyectadas sobre el sistema objetivo.

En la figura 11.2 se presenta la arquitectura detallada de la herramienta, derivada de la arquitectura de alto nivel (figura 10.2) definida en el capítulo de diseño conceptual. Esta representación conserva los módulos de configuración, generador de campañas, inyección de fallas, registro y almacenamiento, y análisis de resultados, e ilustra su organización estructural interna, así como la interacción entre los submódulos durante la ejecución experimental.

En este contexto, aunque los módulos de inyección de fallas y de registro y almacenamiento fueron descritos funcionalmente como entidades independientes, en la arquitectura detallada se representan de forma integrada como un único módulo de inyección y almacenamiento. Esta integración responde a la naturaleza del flujo operativo, en el cual la ejecución de cada evento de inyección se encuentra directamente vinculada al registro inmediato de los datos generados. La representación conjunta permite modelar con mayor precisión la dinámica real de la herramienta durante una campaña de FI, sin alterar la separación funcional definida en el diseño conceptual.

La arquitectura detallada consolida una estructura coherente que integra preparación, ejecución y análisis, asegurando trazabilidad y control experimental en las campañas de FI. En las secciones siguientes se presenta, en primer lugar, la implementación de la GUI, seguida de la descripción de los submódulos que conforman la arquitectura propuesta.

11.2.1 Implementación de la interfaz gráfica de usuario (GUI)

En el capítulo de diseño conceptual se definió la GUI como el módulo responsable de la interacción entre el usuario y la herramienta de FI. A continuación, se describen las ventanas que conforman la interfaz, iniciando con la ventana principal y posteriormente las ventanas secundarias: configuración, preparación, modo automático, modo manual y modo CSV¹. En conjunto, estas ventanas permiten al usuario seleccionar la MCU objetivo, configurar los parámetros de la campaña, definir los eventos y ejecutar el proceso de inyección.

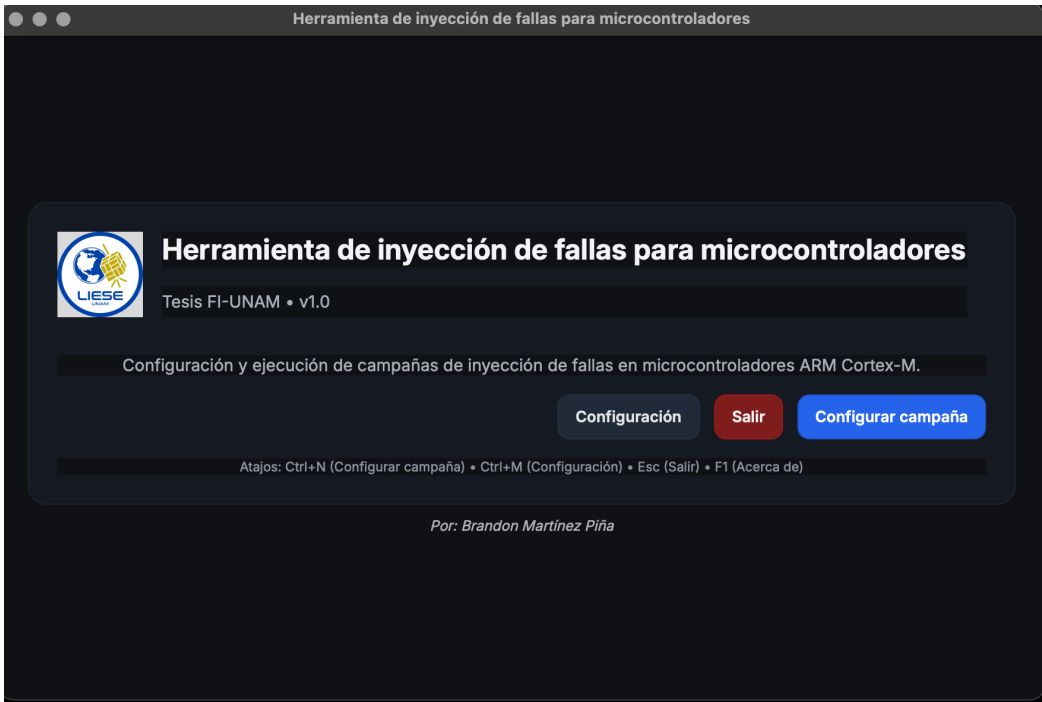


Figura 11.3. Ventana principal de la GUI.

¹ Archivo de texto plano que almacena datos estructurados en formato de tabla, en donde cada línea representa una fila y cada columna se separa por una coma.

Ventana principal

La ventana principal constituye la pantalla de inicio de la herramienta y actúa como punto de acceso al proceso experimental, cuya función es organizar el flujo de trabajo diferenciando la etapa de configuración del entorno de la etapa de ejecución de las campañas. Esta ventana no ejecuta directamente procesos de inyección ni permite definir parámetros experimentales, sino que centraliza el acceso a las ventanas secundarias en donde se realizan dichas tareas.

Como se muestra en la figura 11.3, la ventana presenta información general de la herramienta, como el título, la versión y el autor. Además, incorpora botones para configurar una campaña, configurar el entorno de ejecución o finalizar la aplicación. La inclusión de atajos de teclado facilita la interacción durante sesiones repetitivas, favoreciendo un flujo de trabajo más eficiente durante la ejecución de las campañas de inyección.

Ventana de configuración

La ventana de configuración es accesible desde la ventana principal mediante el botón de *Configuración* y tiene como función preparar el entorno de *software* necesario para que la herramienta pueda interactuar correctamente con la MCU seleccionada.

En la figura 11.4 se observa que la ventana incluye un selector de la MCU, un botón para iniciar la instalación de paquetes y librerías, una barra de progreso y una consola de mensajes. Además, dispone de un botón que permite cerrar la ventana una vez completadas las configuraciones necesarias.

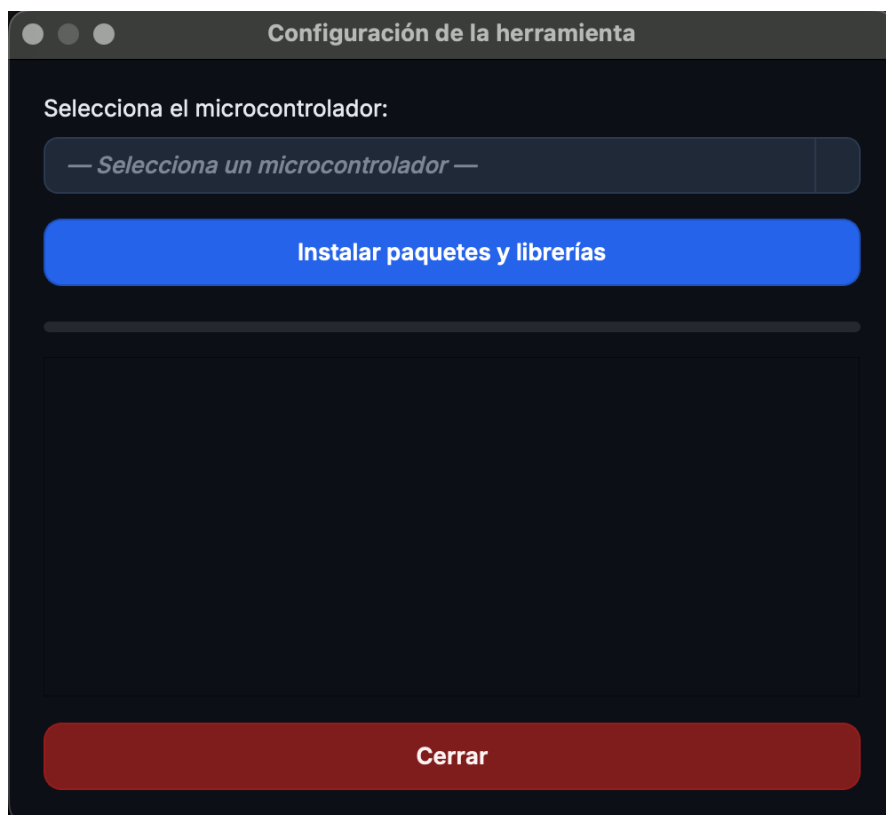


Figura 11.4. Ventana de configuración.

El proceso de configuración inicia cuando el usuario selecciona una MCU a partir de un listado predefinido, el cual está conformado por dispositivos que han sido previamente probados y validados dentro del entorno experimental, garantizando su correcto funcionamiento con la herramienta. En la tabla 11.1 se muestran las plataformas actualmente disponibles para selección.

Tabla 11.1. MCU disponibles en el listado predefinido de la herramienta de FI.

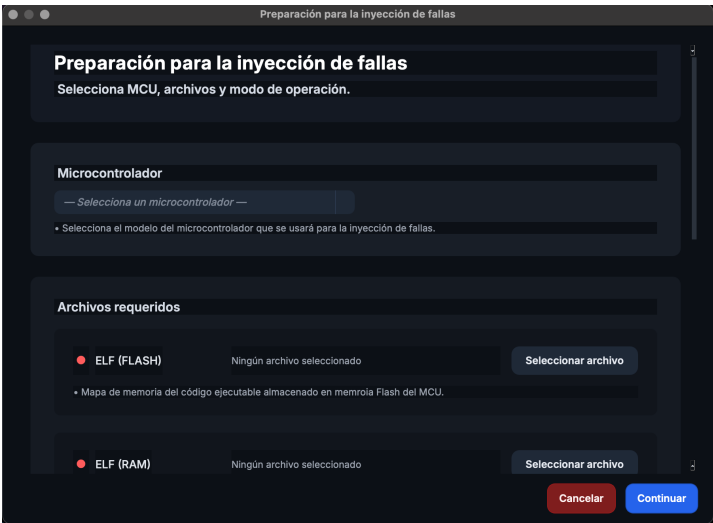
Placa / dispositivo	MCU	Núcleo
STM32F407G-DISC1	STM32F407VGT6	ARM Cortex-M4
NUCLEO-F446RE	STM32F446RET6	ARM Cortex-M4
NUCLEO-F439ZI	STM32F439ZIT6	ARM Cortex-M4
STM32F103C8T6	STM32F103C8T6	ARM Cortex-M3

La instalación de paquetes y librerías es gestionada por el submódulo de preparación del entorno de *software*, el cual verifica las dependencias necesarias e incorpora los componentes específicos de la MCU seleccionada, permitiendo que el entorno quede correctamente configurado antes de iniciar la ejecución de las campañas.

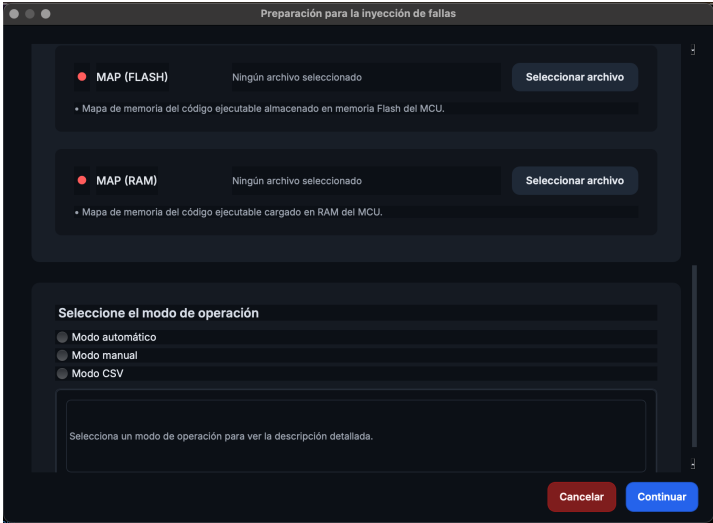
Ventana de preparación

La ventana de preparación se abre al seleccionar el botón de *Configurar campaña* desde la ventana principal y constituye el punto inicial para definir la campaña de FI. Su objetivo es integrar en un único espacio los elementos necesarios para que la herramienta opere correctamente, incluyendo la selección de la MCU, la carga de los archivos requeridos y la elección del modo de operación.

En las figuras 11.5a y 11.5b se presenta la ventana, organizada en tres apartados principales. El primero corresponde a la selección de la MCU, que define el modelo sobre el cual se inyectan las fallas y vincula esta elección con los archivos y configuraciones previamente instalados para dicho dispositivo. El segundo apartado, dedicado a la carga de archivos requeridos, permite incorporar la información necesaria para identificar las regiones de memoria de la MCU ocupadas por el programa. Los archivos ELF contienen el *software* de aplicación compilado para la MCU, organizado en secciones de código y datos, que se asignan a memoria *flash* o RAM según la función de cada sección y la ubicación de ejecución del *software* bajo prueba. En particular, ELF *flash* representa el *software* bajo prueba que se ejecuta desde la memoria *flash* de la MCU, mientras que ELF RAM contiene el *software* bajo prueba cargado para ejecutarse directamente desde la memoria volátil. De manera complementaria, los archivos MAP (*flash*/RAM) proporcionan información detallada sobre las direcciones de memoria, el tamaño de cada sección y la ubicación de funciones y variables para cada archivo ELF, permitiendo identificar exactamente qué función o variable se encuentra en cada dirección de memoria y a qué sección pertenece.



(a) Selección del microcontrolador y carga de archivos requeridos.



(b) Selección del modo de operación de la campaña.

Figura 11.5. Ventana de preparación.

El tercer apartado corresponde a la selección del modo de operación, que determina cómo se construye la campaña. En el modo automático, el usuario establece los parámetros generales a partir de los cuales la herramienta genera la lista de fallas correspondiente. El modo manual permite definir individualmente cada falla, brindando un control total

sobre la campaña, mientras que el modo CSV permite cargar una lista de fallas previamente definida, facilitando la repetición de campañas y la comparación de resultados experimentales.

Antes de avanzar a la siguiente etapa de configuración de la campaña, la herramienta verifica que se haya seleccionado una MCU, que los archivos requeridos hayan sido cargados y que se haya definido un modo de operación. Esta verificación evita que la campaña continúe con una configuración incompleta y asegura que las condiciones necesarias se cumplan antes de iniciar el proceso de inyección.

Ventana del modo automático

El modo automático permite configurar los parámetros necesarios para ejecutar una campaña de FI, a partir de los cuales la herramienta genera la lista de fallas, facilitando la creación de campañas de gran tamaño y la evaluación del comportamiento del sistema objetivo bajo múltiples condiciones de falla.

La figura 11.6 presenta la ventana, la cual incorpora un panel de estado que muestra la MCU seleccionada y los archivos cargados (ELF y MAP, tanto para *flash* como para RAM), proporcionando al usuario una referencia clara del entorno de ejecución. A continuación, se define el número de fallas a inyectar, determinando el tamaño de la campaña de acuerdo con los objetivos experimentales deseados.

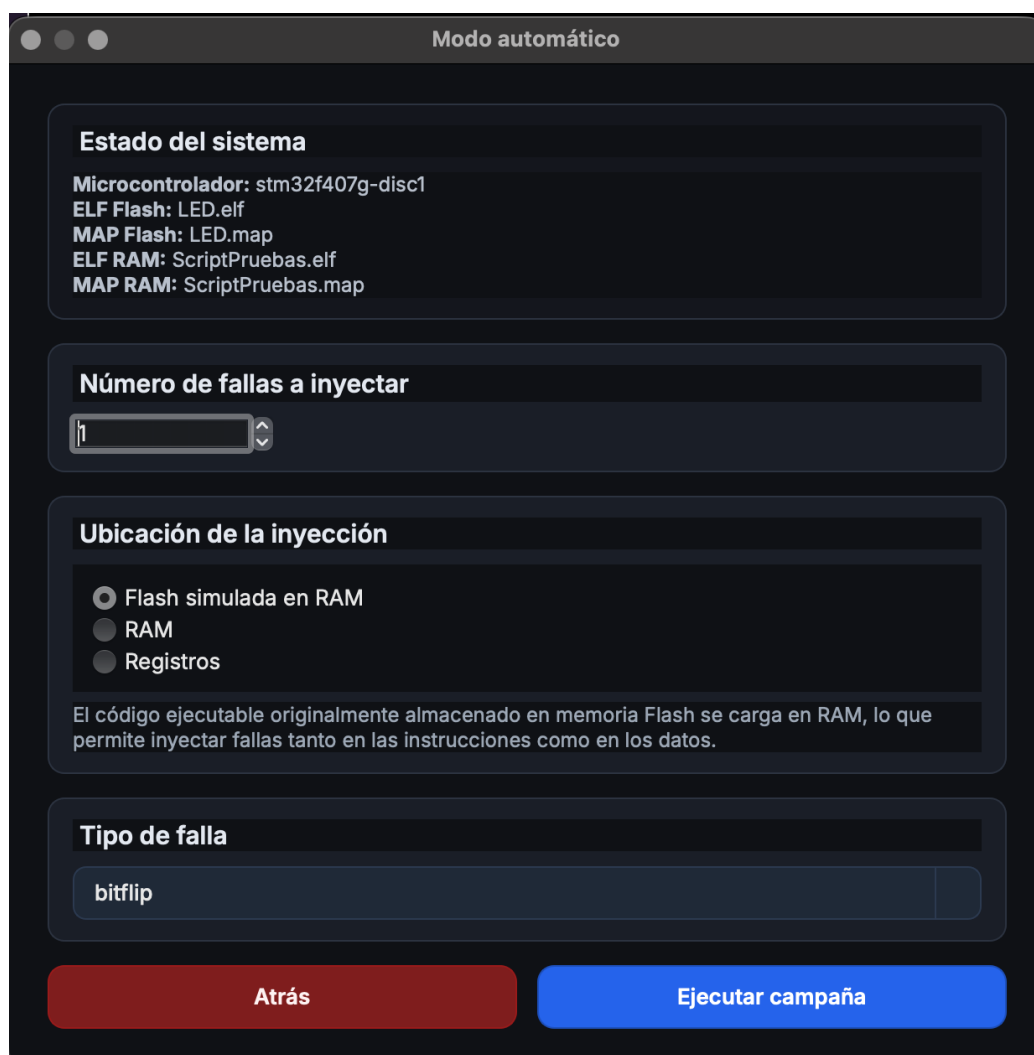


Figura 11.6. Ventana del modo automático.

Asimismo, la herramienta permite seleccionar la ubicación de la inyección, contemplando tres opciones: *flash* simulada en RAM, que consiste en cargar el *software* bajo prueba en la RAM, con el objetivo de inyectar tanto en el código ejecutable como en variables y datos temporales; RAM, en donde la inyección se aplica únicamente sobre los datos almacenados en memoria volátil, mientras que el *software* bajo prueba permanece en la memoria *flash*; y registros, destinada a la inyección sobre registros de configuración de periféricos. Cuando se selecciona esta última opción, la interfaz habilita la selección del periférico objetivo a partir de la descripción del dispositivo, asegurando que la inyección se aplique sobre un componente funcional específico del sistema objetivo.

Finalmente, el usuario define el tipo de falla y puede seleccionar entre *bit-flip*, *stuck-at-0*, *stuck-at-1* o la opción *todos*, en cuyo caso la herramienta selecciona aleatoriamente uno de los tipos disponibles para cada evento de inyección. Una vez definidos todos los parámetros, el proceso se inicia mediante el botón *Ejecutar campaña*, mostrando un indicador de progreso que señala que la ejecución se encuentra en curso e informando su finalización mediante un mensaje informativo.

Ventana del modo manual

La ventana de modo manual permite al usuario definir individualmente cada falla a inyectar en la MCU. Como se observa en la figura 11.7, la ventana muestra el estado del sistema, así como un apartado de configuración general, en donde se establece el número de fallas que formarán la campaña, determinando así el límite de eventos que pueden agregarse a la lista.

Modo Usuario – Lista de fallas definida por el usuario

Estado del sistema

Microcontrolador: stm32f407g-disc1
ELF Flash: LED.elf
MAP Flash: LED.map
ELF RAM: ScriptPruebas.elf
MAP RAM: ScriptPruebas.map

Configuración general

Número de fallas: 1

Configuración de cada falla

Ubicación: RAM
Tipo de falla: bitflip
Dirección de interrupción (32 bits): Ej: 0x20001000
Dirección de inyección (32 bits): Ej: 0x20002000
Máscara (32 bits): Ej: 0x00000020

Agregar falla a la lista

Fallas agregadas

Nueva campaña Generar solo CSV Generar CSV y ejecutar campaña Atrás

Figura 11.7. Ventana del modo manual.

En el apartado de configuración de cada falla, el usuario especifica los parámetros de cada evento: la ubicación (RAM o *flash* simulada en RAM), que indica en qué memoria se encuentra cargado el *software* bajo prueba; el tipo de falla (*bit-flip*, *stuck-at-0* o *stuck-at-1*); la dirección de inyección, correspondiente a la dirección de memoria en donde se aplicará la alteración; la dirección de interrupción (*breakpoint* o *stop*), que señala el instante en el cual el programa se detendrá para aplicar la falla; y la máscara de 32 *bits*, introducida en formato hexadecimal, que permite definir los *bits* sobre las cuales se realizará la modificación.

La herramienta incluye validaciones sobre el formato de las direcciones (8 dígitos hexadecimales) y asegura que el *bit* objetivo seleccionado se encuentre dentro del intervalo permitido (0–31), garantizando que cada falla cumpla con las restricciones de la campaña y evitando posibles errores durante la inyección.

Las fallas se almacenan en una lista visible dentro de la ventana, lo que permite al usuario ver y controlar los eventos de la campaña. Además, la herramienta ofrece la opción de generar únicamente el archivo CSV con la lista de fallas o de generar el CSV y ejecutar la campaña de inmediato, así como iniciar una nueva campaña, que reinicia la lista y facilita las ejecuciones sucesivas.

Ventana del modo CSV

En la figura 11.8 se muestra la ventana del modo CSV, que permite seleccionar un archivo en este formato, facilitando la repetición de campañas experimentales. El archivo contiene la descripción de las fallas a inyectar, y la herramienta valida automáticamente su estructura, verificando la presencia de las columnas requeridas, tales como el identificador de la falla, la ubicación, el tipo de falla, las direcciones de inyección y de interrupción, y la máscara correspondiente. Esta validación asegura la consistencia del formato antes de permitir su ejecución.

Una vez cargado el archivo, la ventana muestra una vista rápida de las fallas contenidas en el CSV, ofreciendo un panorama general de los eventos definidos; además, es posible consultar el contenido completo del archivo para revisar todos los parámetros asociados a cada falla antes de que se inicie la campaña. Al solicitar la ejecución, la herramienta procede a inyectar las fallas y, al finalizar, notifica al usuario sobre la conclusión del proceso, tal como ocurre en los otros modos de operación.

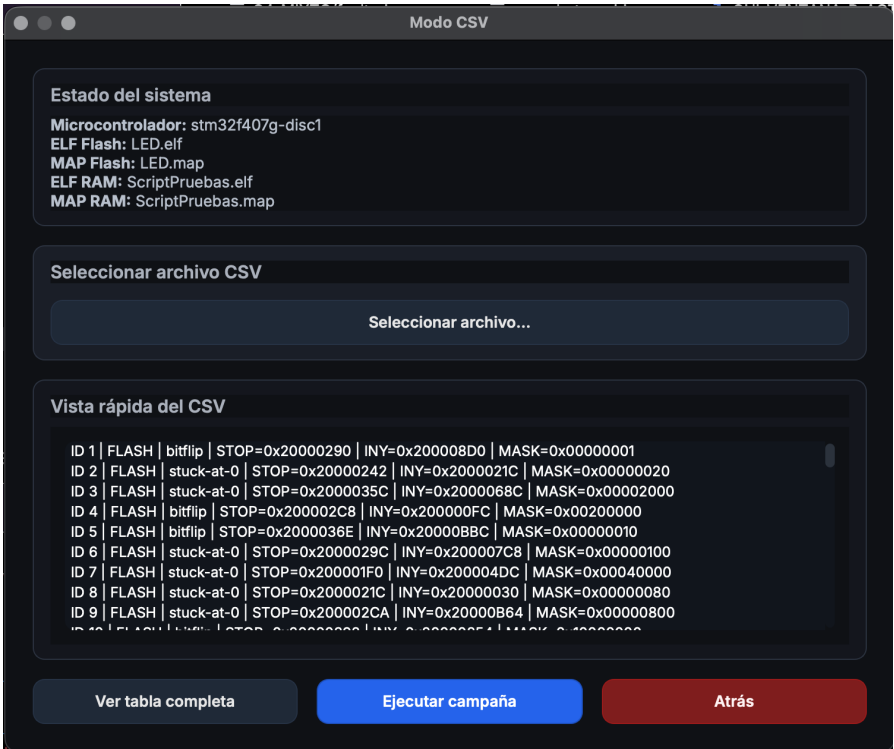


Figura 11.8. Ventana del modo CSV.

11.2.2 Submódulo de preparación del entorno de *software*

El proceso de FI mediante técnicas de depuración requiere un entorno de *software* correctamente configurado y estandarizado. Para que la computadora de control pueda comunicarse con el DUT, es indispensable contar con librerías de python actualizadas, con los paquetes específicos de la MCU, así como con las descripciones de *hardware* (archivos CMSIS-SVD).

El submódulo de preparación del entorno de *software* se encarga de verificar y gestionar estos recursos antes de ejecutar cualquier campaña, asegurando que la MCU seleccionada en la ventana de configuración cuente con todas las dependencias y archivos necesarios. Entre los principales objetivos de este submódulo se encuentran:

- Estandarizar el entorno de ejecución entre distintos equipos.
- Automatizar tareas que, de otro modo, requerirían de una instalación manual.
- Reducir la probabilidad de errores derivados de configuraciones incorrectas.
- Garantizar que el inyector de fallas opere sobre una base de *software* consistente y que el entorno pueda replicarse de manera exacta en cualquier computadora de control.

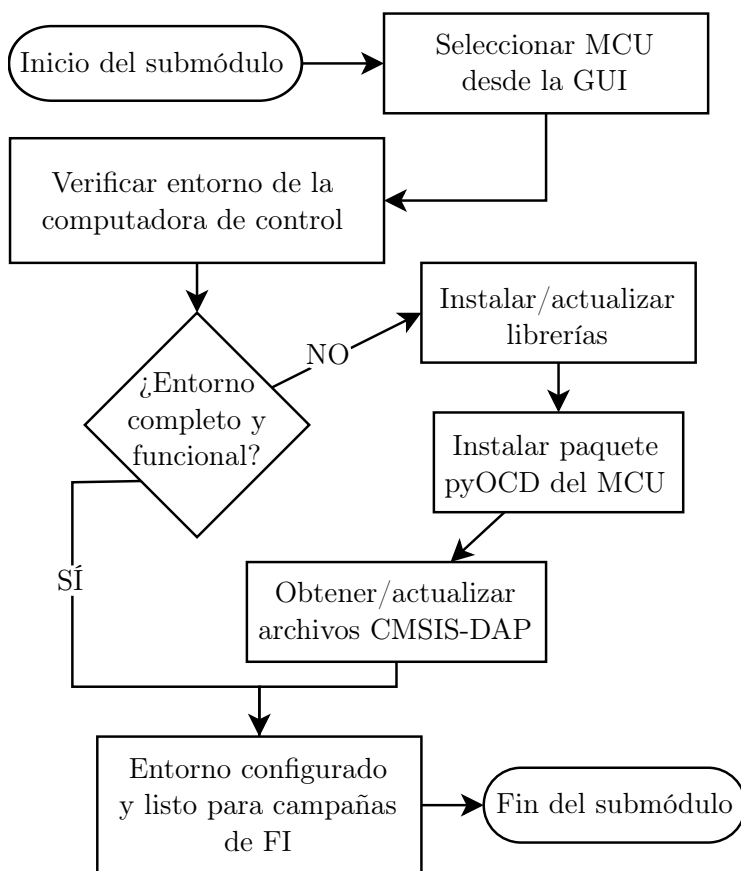


Figura 11.9. Diagrama de flujo del submódulo de configuración.

Descripción general del funcionamiento del submódulo

El submódulo de preparación del entorno de *software* opera a través de tres etapas principales:

1. Instalación de librerías generales.

Se instalan y actualizan las librerías necesarias para el funcionamiento de la herramienta de FI, garantizando que todas las dependencias se encuentren en su versión más reciente y evitando incompatibilidades ocasionadas por *software* obsoleto.

2. Instalación de paquetes específicos de la MCU.

Se descarga e instala el paquete correspondiente al DUT seleccionado, ya que estos paquetes, gestionados directamente por pyOCD, incluyen la información necesaria para que el depurador reconozca el MCU, configure correctamente su mapa de memoria y establezca la comunicación a través de la interfaz OCD. Este mecanismo permite que pyOCD pueda interactuar con distintos dispositivos sin cambios en la arquitectura de la herramienta, lo que aporta extensibilidad y facilita la incorporación de nuevas MCU de la arquitectura Cortex -M.

3. Obtención de archivos CMSIS-SVD.

En esta etapa se descargan o actualizan las descripciones CMSIS-SVD; estos archivos proporcionan un modelo estructurado de los periféricos, registros y campos internos de los dispositivos, y constituyen un repositorio general de descripciones que pueden ser utilizados por la herramienta para distintas MCU compatibles.

La figura 11.9 muestra el diagrama de flujo del submódulo de preparación del entorno de *software*, en donde se presentan las etapas de verificación del entorno, instalación de librerías, instalación del paquete de la MCU y obtención de los archivos CMSIS-SVD, asegurando que la herramienta quede lista para la FI.

11.2.3 Submódulo de análisis de archivos ELF

Un paso importante dentro del proceso de FI es conocer con precisión las direcciones de memoria asociadas a las instrucciones ejecutables del programa de la MCU. Por ello, resulta fundamental analizar el archivo ELF generado durante la compilación del *software* bajo prueba, el cual contiene información estructural del código, las secciones de memoria y los símbolos del sistema.

Con el objetivo de automatizar esta tarea y evitar la selección manual de direcciones, se desarrolló un submódulo de análisis de archivos ELF encargado de identificar las direcciones correspondientes al código ejecutable del programa. Esto permite construir un conjunto de puntos válidos para la colocación de *breakpoints* para la FI, de manera que la ejecución solo se detenga en instrucciones pertenecientes al flujo de ejecución del programa.

Descripción general del funcionamiento del submódulo

El submódulo de análisis de archivos ELF opera de manera secuencial y se apoya en herramientas estándar del *toolchain* de ARM, ampliamente utilizadas para el desarrollo, compilación, depuración y programación de aplicaciones que se ejecutan en MCU que integran núcleos basados en la arquitectura ARM Cortex-M.

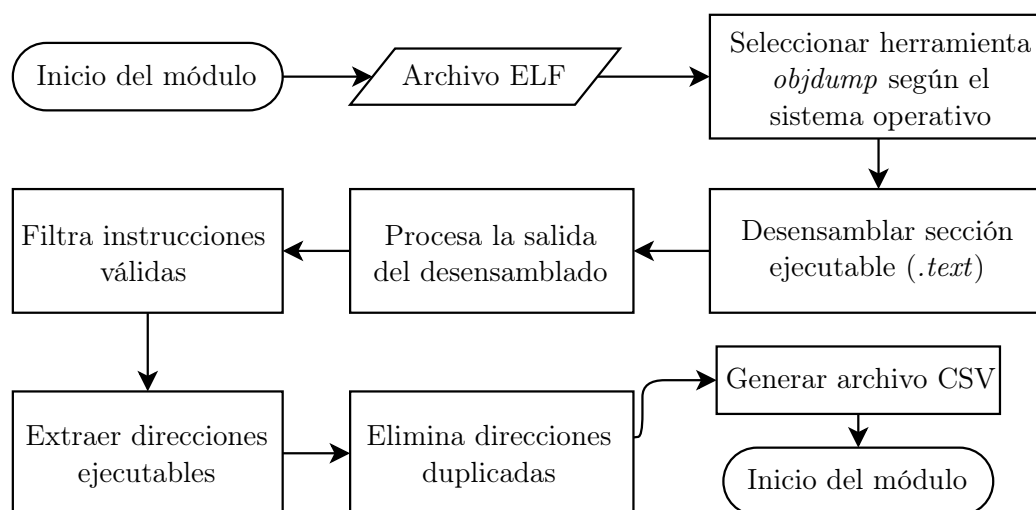


Figura 11.10. Diagrama de flujo del módulo de análisis de archivos ELF.

La figura 11.10 presenta el diagrama de flujo correspondiente, el cual resume la secuencia de procesamiento seguida para la obtención de direcciones ejecutables a partir del binario del DUT y su funcionamiento se compone de las siguientes etapas:

1. Validación del archivo ELF.

Se verifica la existencia y accesibilidad del archivo ELF proporcionado como entrada desde la GUI, asegurando que el análisis se realice únicamente sobre binarios válidos generados durante el proceso de compilación del *software* bajo prueba.

2. Desensamblado de la sección ejecutable.

Se emplea la herramienta *arm-none-eabi-objdump*, perteneciente al *toolchain* GNU para ARM, para desensamblar² la sección *.text*³ del archivo ELF y obtener las instrucciones en ensamblador junto con sus direcciones de memoria asociadas. El submódulo detecta automáticamente el sistema operativo del entorno de ejecución y ejecuta esta herramienta de forma compatible con dicho sistema, lo que permite realizar el análisis de manera transparente en distintos sistemas operativos y aporta portabilidad al submódulo.

3. Extracción de direcciones ejecutables.

A partir de la salida del desensamblado, se identifican y filtran únicamente las direcciones asociadas a instrucciones ejecutables, descartando directivas de datos u otros elementos no relevantes. Este proceso permite construir un conjunto depurado de puntos de interés para la FI, los cuales son utilizados como referencias temporales para la activación de los mecanismos de inyección durante la ejecución del *software* en la MCU.

4. Almacenamiento estructurado de resultados.

Las direcciones identificadas se almacenan en un archivo CSV, facilitando su integración con otros submódulos de la herramienta, como el submódulo generador de listas de fallas.

11.2.4 Submódulo de análisis de archivos SVD

El submódulo de análisis de archivos SVD tiene como finalidad automatizar y facilitar el proceso de FI en los registros de configuración de periféricos de la MCU; para ello, interpreta y procesa los archivos CMSIS-SVD asociados al DUT, los cuales describen la organización del mapa de registros y sus componentes internos.

Durante su funcionamiento, el submódulo identifica de forma automática los registros disponibles, así como los *bits* y campos válidos sobre los que es posible realizar la FI; este proceso se realiza sin intervención manual del usuario, reduciendo la probabilidad de errores y asegurando que únicamente se consideren registros oficialmente definidos por el fabricante.

Como resultado, el submódulo proporciona una base de información consistente y confiable, utilizada por los demás submódulos de la herramientas de inyección, lo que permite seleccionar de manera controlada las direcciones de inyección. Así, se garantiza que las campañas de FI se ejecuten de manera reproducible y alineada con la arquitectura de la MCU, evitando accesos inválidos o no documentados.

Descripción general del funcionamiento del submódulo

El submódulo de análisis de archivos SVD opera de manera secuencial, como se muestra en la figura 11.11, y utiliza los archivos CMSIS-SVD como fuente principal de información estructurada para el análisis del mapa de registros de la MCU.

²Proceso mediante el cual un archivo binario ejecutable es convertido a su representación en lenguaje ensamblador, permitiendo analizar las instrucciones y sus direcciones de memoria [69].

³Segmento del archivo ELF que contiene las instrucciones ejecutables del programa. Estas instrucciones corresponden al código ejecutable de la MCU y se almacenan en la memoria *flash* o en la RAM, según la configuración del sistema y la ubicación definida durante el proceso de compilación.

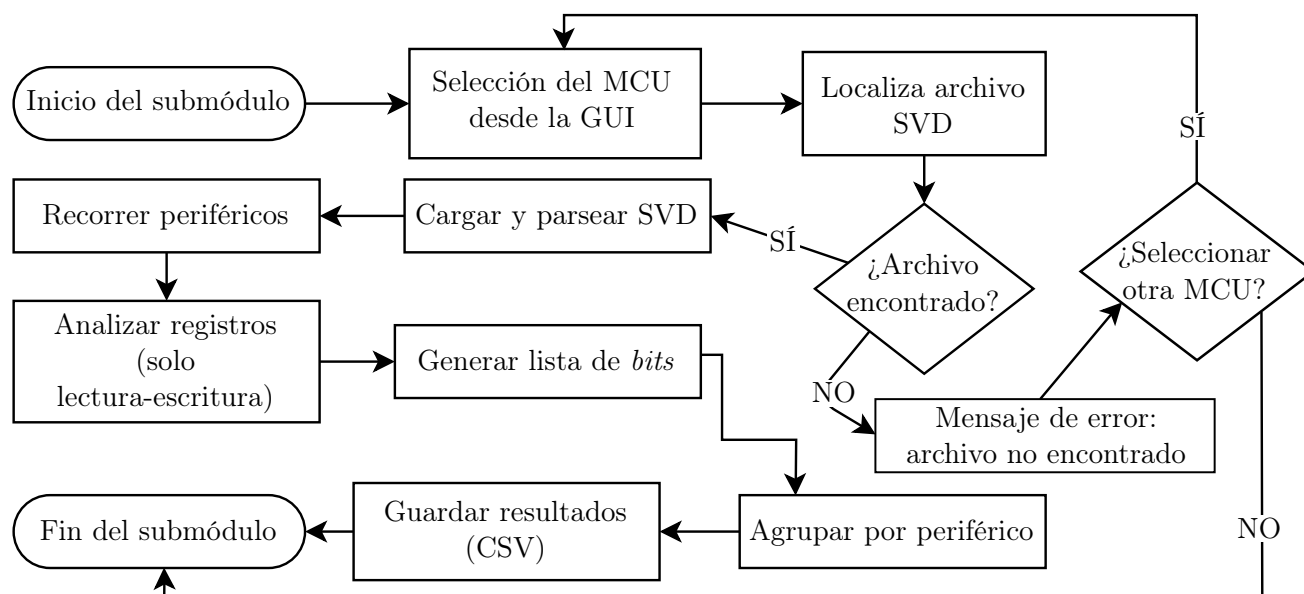


Figura 11.11. Diagrama de flujo del submódulo de análisis de archivos SVD.

A partir de esta información, el funcionamiento general del módulo se compone de las siguientes etapas:

1. Localización del archivo SVD.

Se identifica y localiza el archivo CMSIS-SVD correspondiente a la MCU seleccionada por el usuario dentro del repositorio local de archivos SVD, previamente obtenido por el submódulo de preparación del entorno de *software*, asegurando que el análisis se realice sobre la descripción asociada específicamente al DUT.

2. Parseo⁴ del archivo SVD y análisis de periféricos y registros.

El archivo SVD es parseado para extraer su estructura, la cual es posteriormente analizada recorriendo los periféricos definidos por el fabricante y los registros asociados a cada uno de ellos, junto con sus campos y permisos de acceso.

3. Identificación de campos válidos para inyección.

A partir de la información de permisos de acceso, se identifican únicamente los registros y campos definidos como de lectura-escritura, descartando aquellos de solo lectura, solo escritura o reservados. Con base en esta información, se determinan las posiciones de *bits* candidatas para la FI, evitando la inyección en ubicaciones que no pueden ser modificadas o cuya observación posterior no es posible.

4. Organización y almacenamiento de resultados.

Los *bits* identificados se organizan inicialmente en una lista global y posteriormente se agrupan por periférico, permitiendo su almacenamiento en archivos CSV para su uso posterior por otros submódulos dentro de la herramienta y su selección durante la generación de campañas de inyección.

11.2.5 Submódulo de análisis de RAM y flash

El submódulo de análisis de RAM y *flash* tiene como finalidad identificar los rangos de direcciones de memoria que pueden ser utilizados durante las campañas de FI. Para ello, se realiza el análisis de los archivos MAP y ELF generados durante la compilación del *software*, garantizando que las direcciones seleccionadas correspondan a regiones válidas y coherentes con la organización del código y los datos cargados en la MCU.

El submódulo distingue entre dos tipos de rangos de memoria. En primer lugar, obtiene los rangos totales de RAM y memoria *flash*, que abarcan toda la memoria disponible de la MCU. En segundo lugar, identifica los rangos

⁴Lectura e interpretación de un archivo estructurado para identificar y organizar sus componentes internos de acuerdo con un modelo formal, permitiendo su posterior procesamiento automático por *software*. [70]

efectivos de memoria, es decir, aquellas regiones que contienen secciones activas del *software* y que, por lo tanto, pueden considerarse como direcciones válidas para la FI.

Este análisis permite determinar automáticamente si el *software* se encuentra almacenado en memoria *flash* y RAM, o si reside únicamente en RAM, lo que corresponde a un escenario en el que la memoria *flash* está simulada en RAM. De esta manera, se restringen las campañas de inyección a regiones de memoria efectivas, evitando accesos a direcciones no ocupadas o fuera del mapa de memoria definido para el sistema.

Descripción general del funcionamiento del submódulo

El submódulo opera de manera secuencial a partir de los archivos MAP y ELF, utilizando la información contenida en ellos para identificar y clasificar las regiones de memoria de la MCU que serán consideradas durante las campañas de FI. El funcionamiento general del submódulo se muestra en la figura 11.12 y se compone de las siguientes etapas:

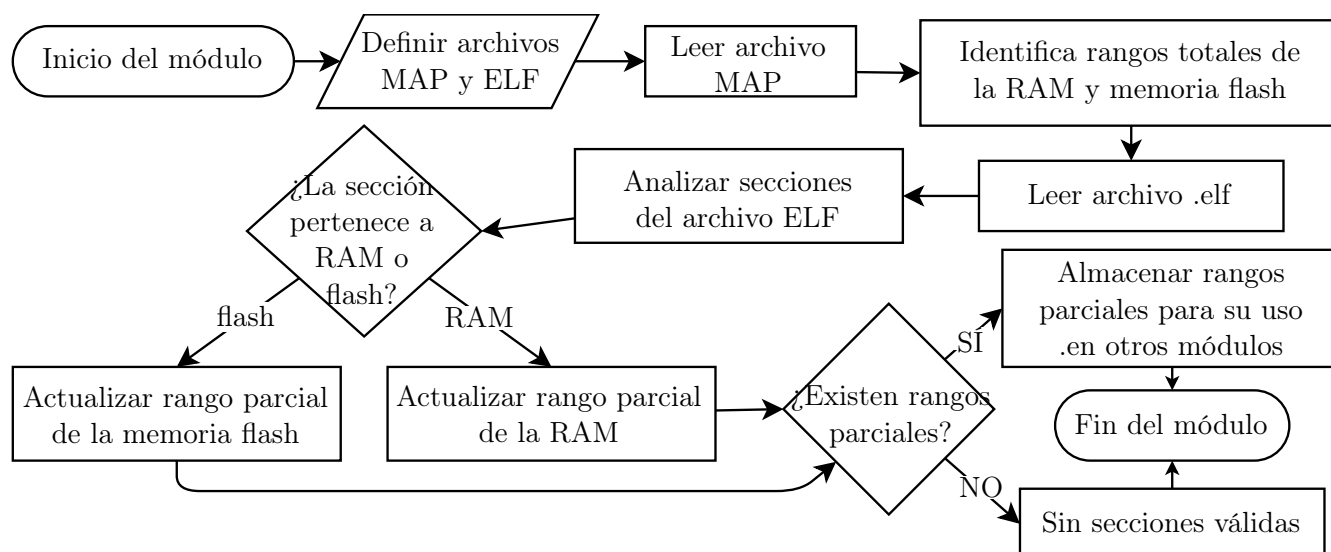


Figura 11.12. Diagrama de flujo del módulo de análisis de RAM y *flash*.

1. Identificación de rangos totales de memoria.

Se analiza el archivo MAP con el objetivo de identificar los rangos completos de direcciones correspondientes a la RAM y a la memoria *flash* de la MCU. Estos rangos representan el espacio total de direccionamiento disponible para cada tipo de memoria y sirven como referencia para el análisis posterior.

2. Lectura y análisis del archivo ELF.

Se procesa el archivo ELF para identificar las secciones que conforman el *software*, obteniendo sus direcciones de inicio y tamaños. Durante este proceso se descartan aquellas secciones que no contienen información relevante para la ejecución del programa.

3. Clasificación de secciones por tipo de memoria.

Las direcciones de las secciones del *software* se comparan con los rangos totales de RAM y memoria *flash* previamente identificados, determinando si cada sección se encuentra ubicada en RAM o en memoria *flash*. Este análisis tiene como fin detectar tanto la configuración habitual, en la que el código se ejecuta desde *flash*, como escenarios alternativos en los que el *software* se encuentra total o parcialmente ubicado en RAM.

4. Determinación de rangos efectivos de memoria.

A partir de las secciones válidas detectadas, se identifican los rangos parciales de RAM y memoria *flash* que contienen código o datos activos del *software*, definiendo sus límites inicial y final. Estos rangos representan las regiones de memoria efectiva utilizadas durante la ejecución del *software* en la MCU.

5. Generación de rangos de memoria utilizables.

Los rangos efectivos de RAM y memoria *flash* identificados se organizan como intervalos de direcciones, los cuales pueden ser utilizados posteriormente por otros submódulos de la herramienta durante la generación de campañas de FI.

11.2.6 Submódulo generador de lista de fallas

Dentro del proceso de generación de campañas de FI es necesario definir de manera precisa las características de cada falla inyectada en la MCU. Esto incluye la ubicación, la dirección de inyección, el tipo de falla y dirección de interrupción.

Con el objetivo de que la ejecución de las campañas de FI se lleve a cabo de forma automatizada, se diseñó un submódulo encargado de construir estructuras de datos que representan listas de fallas a partir de la información obtenida durante la compilación del *software* bajo prueba. Este submódulo integra los resultados del análisis de RAM y memoria *flash*, del análisis de archivos ELF y del análisis de archivos SVD, lo que garantiza que las fallas generadas correspondan a direcciones válidas y sean coherentes con la arquitectura de la MCU.

Descripción general del funcionamiento del submódulo

El funcionamiento del submódulo generador de lista de fallas se divide en las siguientes etapas:

1. Selección del tipo de falla y ubicación

El submódulo recibe como parámetros, a través de la GUI, la selección de la ubicación (registros, RAM o *flash* simulada sobre RAM), el tipo de falla (*bit-flip*, *stuck-at-0*, *stuck-at-1*) o la opción “todos”, así como el número total de fallas de la campaña.

2. Inicialización y análisis de memoria.

El submódulo emplea los resultados generados por el submódulo de análisis de RAM y *flash* para determinar los rangos efectivos de memoria del DUT, los cuales se utilizan como límites válidos para la generación de direcciones de FI.

3. Detección de secciones ejecutables.

El submódulo emplea los resultados generados por el submódulo de análisis de archivos ELF, almacenados y organizados en archivos CSV, para determinar las direcciones de interrupción (*breakpoints*) asociadas a instrucciones ejecutables del *software* bajo prueba, las cuales permiten interrumpir la ejecución en instantes precisos y sincronizar la aplicación de las inyecciones de fallas.

4. Incorporación de información de registros desde archivos SVD.

El submódulo emplea los resultados generados por el submódulo de análisis de archivos SVD, almacenados y organizados en archivos CSV, para incorporar información asociada a los registros de configuración de periféricos de la MCU, la cual es utilizada durante la generación de fallas sobre registros.

5. Generación de fallas. A partir de los rangos válidos de memoria, de las secciones ejecutables identificadas y de la información de registros cargada desde archivos CSV, el submódulo genera las direcciones de inyección y las máscaras de *bits* correspondientes a cada falla. Las direcciones se alinean en palabras de 32 *bits* para RAM y memoria *flash* simulada en RAM, mientras que para registros se seleccionan periféricos y *bits* válidos conforme a la información SVD procesada. Las máscaras permiten determinar los *bits* específicos sobre los que se aplicará la falla, de acuerdo con el tipo de inyección seleccionado. Adicionalmente, se seleccionan de manera aleatoria direcciones de interrupción asociadas a instrucciones ejecutables, las cuales definen los instantes de ejecución en los que se aplican las inyecciones de fallas, garantizando la consistencia técnica de cada falla generada.

6. Almacenamiento de resultados.

Finalmente, la lista de fallas generadas se almacena en un archivo CSV estructurado, el cual es utilizado posteriormente durante la ejecución de las campañas de inyección.

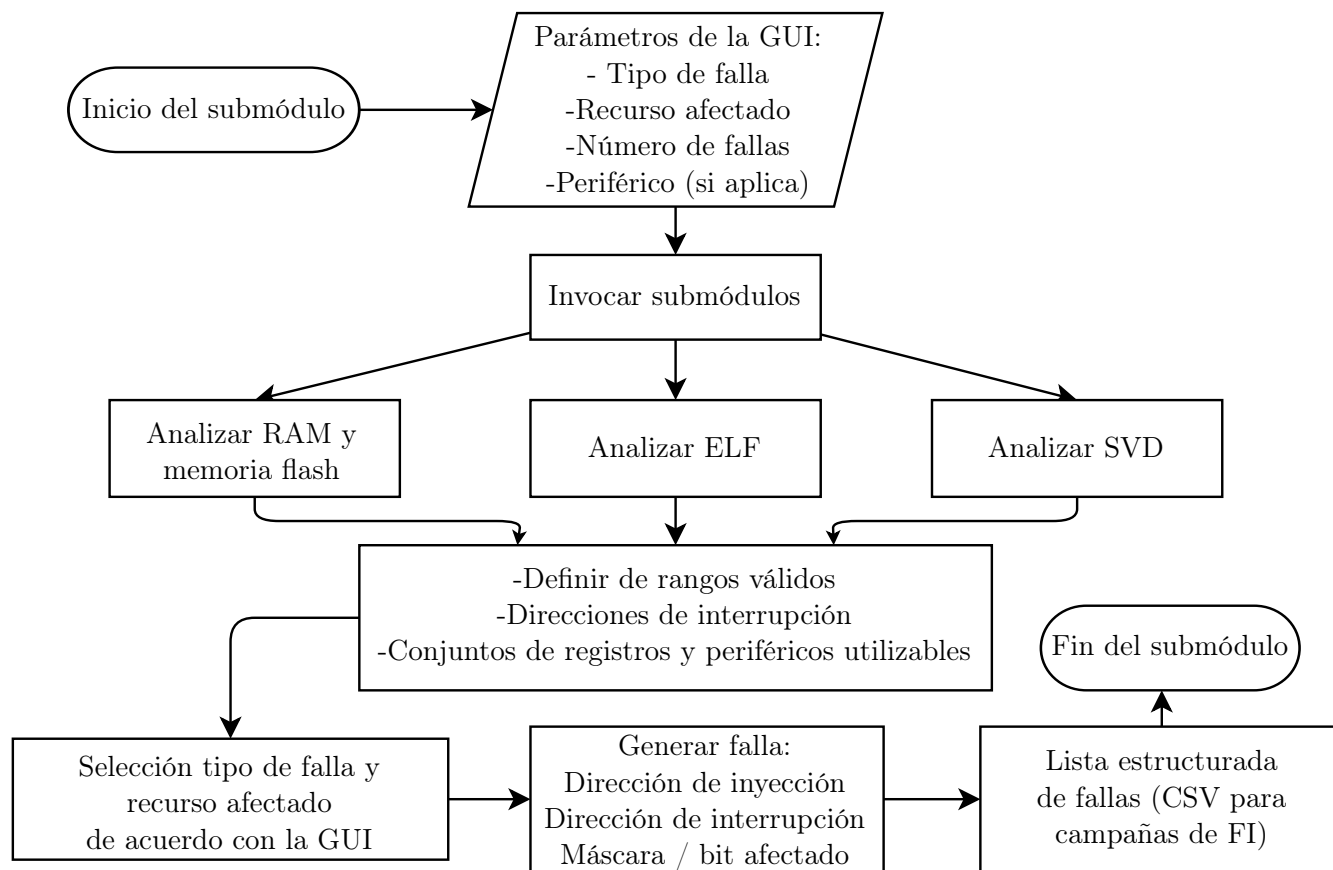


Figura 11.13. Diagrama de flujo del submódulo generador de lista de fallas.

La figura 11.13 presenta un diagrama de flujo que resume la secuencia general de operación del submódulo de generación de fallas, integrando las etapas previamente descritas y su relación con los parámetros definidos por el usuario.

11.2.7 Submódulo generador de lista de fallas manual

Si bien la herramienta de FI incorpora un submódulo para la generación automática de campañas a partir del análisis del *software* y del mapa de memoria de la MCU, también permite definir fallas de manera explícita mediante el modo de operación manual. Esta capacidad resulta especialmente relevante para garantizar la repetibilidad de eventos de falla específicos identificados durante campañas de inyección previas.

Con este propósito, se desarrolló el submódulo generador de lista de fallas manual, cuyo objetivo es permitir que el usuario, a través de la GUI, defina todos los parámetros asociados a cada evento de inyección. Adicionalmente, el submódulo verifica que los valores ingresados cumplan con las restricciones de formato y sean consistentes con la arquitectura de la MCU.

Descripción general del funcionamiento del submódulo

El submódulo generador de lista de fallas manual opera mediante un flujo guiado por el usuario, en el cual cada evento de inyección es definido de forma explícita a través de la interfaz. El funcionamiento general del submódulo se compone de las siguientes etapas:

1. Inicialización del proceso de definición.

A través de la GUI, el submódulo permite al usuario especificar el número total de fallas a generar, verificando que dicho valor se encuentre dentro de un rango permitido (entero válido mayor o igual a uno, descartando valores

nulos o negativos). Con base en esta información, se inicializan las estructuras de datos internas encargadas de almacenar, organizar y gestionar los parámetros asociados a cada evento de inyección.

2. Selección de parámetros de la falla.

Para cada falla, el usuario define de manera manual, mediante la interfaz gráfica, la ubicación (RAM o memoria *flash* simulada en RAM), el tipo de falla a inyectar (*bit-flip*, *stuck-at-0* o *stuck-at-1*) y las direcciones de control asociadas al evento de inyección, correspondientes a la dirección de interrupción del sistema objetivo y a la dirección de inyección, además de la máscara de la falla.

3. Validación de los datos ingresados.

Durante la captura de información a través de la GUI, el submódulo valida que las direcciones ingresadas correspondan a valores hexadecimales de 32 *bits*. Además, valida que la máscara ingresada por el usuario contenga exactamente un único *bit* en uno y, a partir de esta máscara, calcula automáticamente la posición del *bit* afectado, garantizando la coherencia de todos los parámetros antes de almacenarlos.

4. Almacenamiento estructurado de la falla.

Una vez validados y completados, los parámetros de cada falla se almacenan como entradas dentro de las estructuras de datos internas del submódulo, que representan la campaña de inyección definida por el usuario.

5. Exportación de la lista de fallas.

Al finalizar el proceso, la lista completa de fallas se exporta a un archivo CSV, siguiendo un formato predefinido y compatible con los submódulos encargados de la ejecución de las campañas de inyección.

La figura 11.14 muestra el diagrama de flujo que representa de manera general el funcionamiento del submódulo generador de fallas manual, incluyendo las etapas principales desde la inicialización hasta la exportación de la lista de fallas.

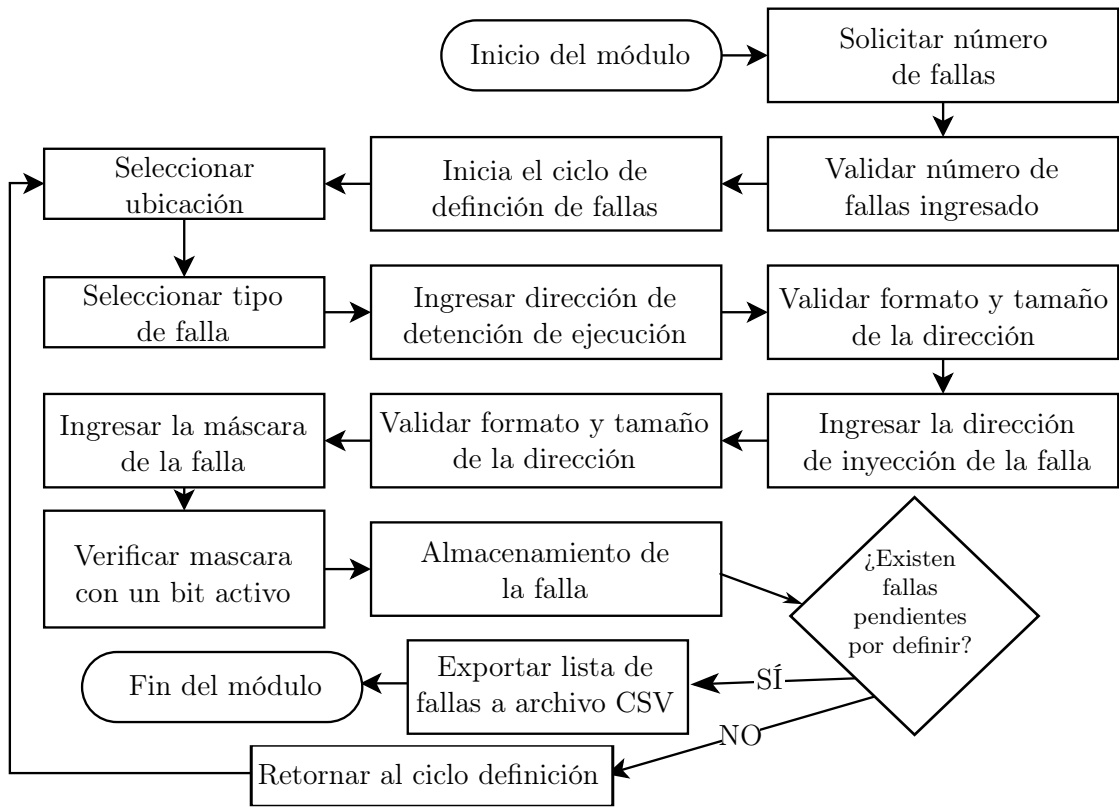


Figura 11.14. Diagrama de flujo del submódulo generador de lista de fallas manual.

11.2.8 Submódulo de detección de bucles

Durante la ejecución de las campañas de FI, resulta fundamental contar con un punto de interrupción que permita monitorear el comportamiento del dispositivo posteriormente a la inyección de una falla. Para este propósito, se desarrolló un submódulo cuyo objetivo es identificar automáticamente la instrucción asociada al bucle *while(1)* dentro de la función principal, ya que este representa el estado de ejecución estable del sistema una vez finalizada la fase de inicialización del *software*.

El punto de interrupción determinado por este submódulo se utiliza como un *breakpoint*, suspendiendo de forma controlada la ejecución del programa después de la inyección de la falla, facilitando la lectura del estado interno de la MCU, incluyendo registros y memoria.

Descripción general del funcionamiento del submódulo

A continuación, se describen las etapas del submódulo encargadas de configurar el punto de interrupción para la observación controlada de la MCU. La figura 11.15 presenta de manera esquemática el flujo de operación del submódulo.

1. Inicialización del análisis del *software*.

El submódulo recibe como entrada el archivo ELF generado durante la compilación del *software* y prepara el entorno necesario para realizar el análisis estático del programa.

2. Identificación de la función principal.

Se determina el rango de direcciones correspondiente a la función principal, delimitando la región del código sobre la cual se busca el bucle de ejecución estable.

3. Análisis de las instrucciones de salto.

Dentro del rango definido, se examinan las instrucciones de salto del código para identificar posibles bucles de ejecución, es decir, lugares en donde el flujo podría repetirse.

4. Detección de bucles de ejecución infinita.

De los bucles identificados, se seleccionan aquellos que no presentan condición de salida, característicos de un bucle infinito como *while(1)*.

5. Selección del punto de observación.

En caso de existir múltiples bucles detectados, se selecciona el bucle ubicado más cercano al final de la función principal, que normalmente representa el estado de ejecución estable del sistema. El submódulo devuelve la dirección de la instrucción de salto que regresa la ejecución al inicio del bucle, la cual se utiliza como punto de interrupción para la observación controlada de la MCU.

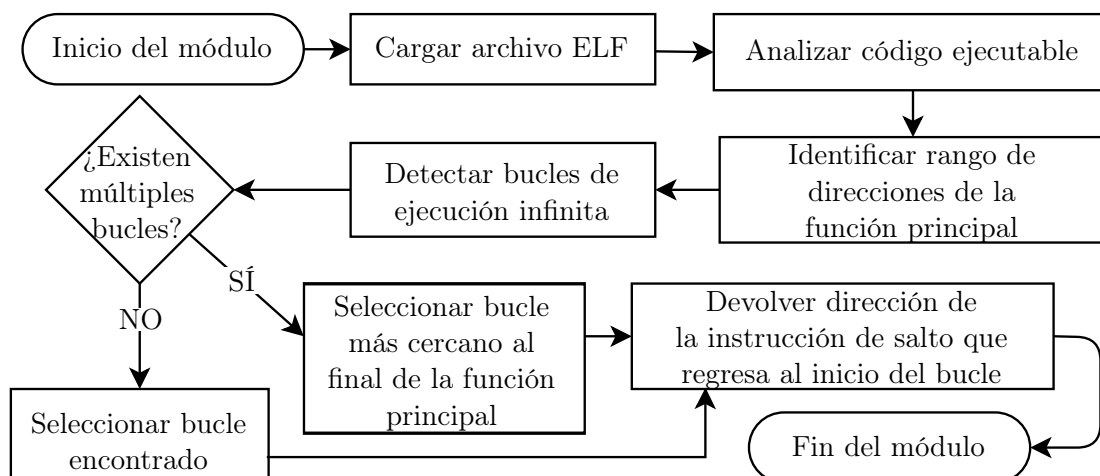


Figura 11.15. Diagrama de flujo del submódulo de detección de bucles.

11.2.9 Submódulo de gestión de sesión con la MCU

Para el desarrollo de las campañas de FI, es indispensable contar con un mecanismo que permita establecer, controlar y finalizar de manera segura la comunicación entre la computadora de control y el DUT. El objetivo principal de este submódulo es abstraer el proceso de conexión mediante una sonda de depuración, gestionar la programación del *software* en la MCU y permitir el acceso controlado al núcleo, actuando como una capa intermedia entre el módulo de inyección de fallas y el *hardware*.

Descripción general del funcionamiento del submódulo

El funcionamiento del submódulo de gestión de sesión con la MCU puede resumirse en las siguientes etapas:

1. Definición de parámetros de sesión de depuración.

El submódulo utiliza un conjunto de opciones de depuración predefinidas como parte del diseño de la herramienta, con el propósito de establecer una configuración consistente y controlada para la comunicación con la MCU. Estas opciones determinan los parámetros necesarios para la conexión, programación y control de la ejecución del dispositivo, evitando que el usuario tenga que configurarlos manualmente. Asimismo, esta configuración garantiza condiciones reproducibles durante la ejecución de las campañas de inyección de fallas, lo cual es fundamental para asegurar la validez y repetibilidad de los resultados experimentales. Adicionalmente, el submódulo recibe como entrada, desde la GUI, la ruta del archivo ELF que contiene el *software* bajo prueba del DUT.

2. Inicialización de la sesión de depuración con la MCU.

El submódulo crea y abre una sesión de depuración con la MCU utilizando una sonda de depuración compatible, deteniendo la CPU en un estado controlado para permitir las operaciones posteriores de depuración.

3. Selección de la MCU y su núcleo principal.

Una vez iniciada la sesión, el submódulo identifica la MCU objetivo y determina el núcleo principal que se utiliza para las operaciones de depuración. En MCU de un solo núcleo se emplea directamente ese núcleo, mientras que en sistemas multinúcleo se puede seleccionar el núcleo principal para control.

4. Programación del *software*.

El submódulo carga el *software* bajo prueba en la memoria de la MCU utilizando el archivo ELF proporcionado, dejando el dispositivo listo para su ejecución.

5. Disponibilidad del sistema objetivo para pruebas.

Tras la programación, la MCU queda listo para la ejecución de operaciones de depuración, tales como reinicios controlados, interrupción o reanudación de la CPU, lectura y escritura de memoria o registros, y ejecución del programa.

6. Finalización y cierre seguro de la sesión.

Al concluir las operaciones, el submódulo cierra la sesión de manera controlada y libera todos los objetos y conexiones establecidas, asegurando la correcta desconexión de la MCU y evitando estados inconsistentes en la sonda de depuración.

La figura 11.16 presenta el flujo de operación del submódulo de gestión de sesión MCU, resumiendo la secuencia de etapas necesarias para establecer la comunicación con la MCU, ejecutar las operaciones de depuración y finalizar la sesión de manera segura.

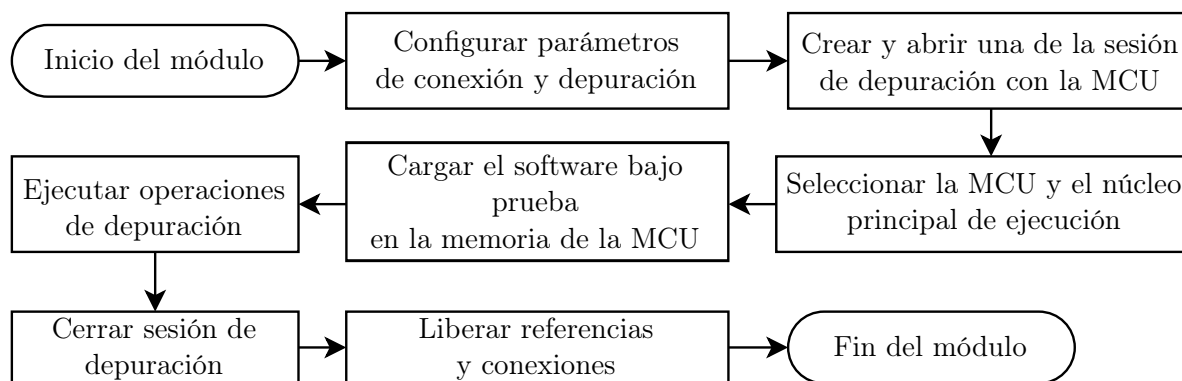


Figura 11.16. Diagrama de flujo del submódulo gestión de sesión con la MCU.

11.2.10 Submódulo de gestión de sesión con la MCU y programación en RAM

En este proyecto, la FI en memoria *flash* se realiza mediante su simulación en RAM, tal como se describió. Para soportar este mecanismo, se desarrolló un submódulo de gestión de sesión con la MCU, encargado de establecer, controlar y finalizar de manera segura la comunicación con la computadora de control, así como de programar el archivo ELF en el dispositivo.

Descripción general del funcionamiento del submódulo

El submódulo establece una sesión de depuración con el DUT y analiza el archivo ELF correspondiente al *software* bajo prueba para obtener la dirección base de la tabla de vectores. Con esta información, prepara la MCU para ejecutar dicho *software* desde la RAM, ajustando los registros del núcleo para iniciar correctamente la ejecución con base en la información contenida en la tabla de vectores.

De forma general, el funcionamiento del submódulo se divide en las siguientes etapas:

1. Definición de parámetros de sesión de depuración.

El submódulo utiliza un conjunto de opciones de depuración predefinidas para establecer la comunicación con el DUT y recibe desde la GUI la ruta del archivo ELF que se carga en la MCU.

2. Inicialización de la sesión de depuración.

Se establece una conexión con la MCU mediante una sonda de depuración compatible, empleando los parámetros definidos previamente para crear y abrir una sesión de depuración.

3. Programación del *software*.

El archivo ELF es programado en el DUT respetando las direcciones de memoria definidas durante la compilación, permitiendo cargar el *software* completo de acuerdo con la organización de memoria establecida en la RAM.

4. Análisis de la tabla de vectores del ELF.

El submódulo inspecciona el archivo ELF para localizar la tabla de vectores y obtener las direcciones base necesarias para definir el punto inicial de ejecución del *software*.

5. Configuración del estado inicial de ejecución.

Con base en la información obtenida de la tabla de vectores, el submódulo configura el estado inicial de ejecución del *software* en la MCU, estableciendo el punto de ejecución inicial del programa de forma controlada, sin aplicar un reinicio adicional del dispositivo.

6. Preparación del sistema para campañas de FI

Una vez iniciado el *software* desde la ubicación correcta, el submódulo mantiene habilitado el acceso a la MCU, permitiendo la observación y manipulación del comportamiento del DUT mediante operaciones de lectura y escritura de memoria, colocación de puntos de interrupción y ejecución controlada durante las campañas de FI.

7. Finalización de la sesión de depuración.

Al concluir las operaciones de control o inyección, el submódulo libera de manera segura la sesión de depuración, cerrando la comunicación con la MCU y finalizando la interacción con el DUT.

La figura 11.17 muestra el flujo de funcionamiento del submódulo de gestión de sesión con la MCU y programación en RAM, desde la definición de parámetros de depuración hasta la finalización de la sesión, incluyendo la programación del *software* en la MCU y la habilitación del acceso para realizar operaciones de depuración durante las campañas de FI.

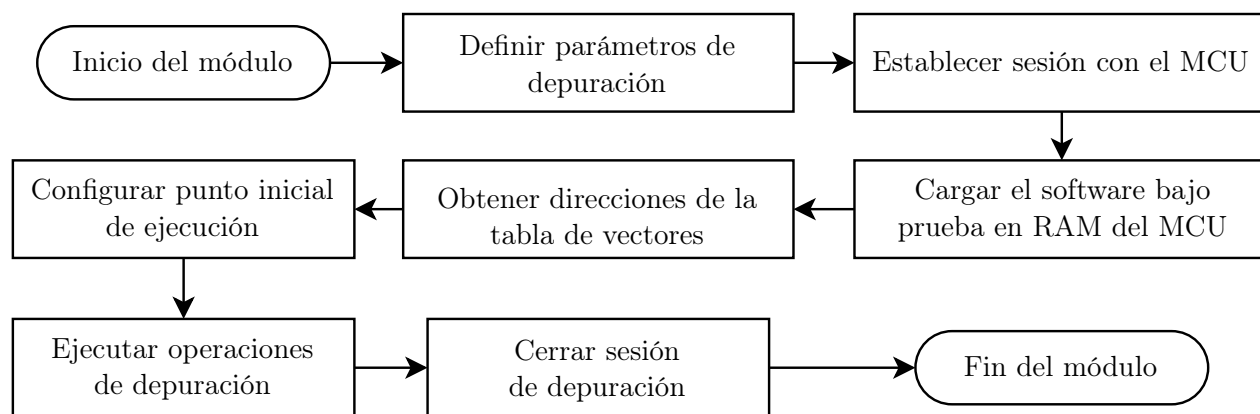


Figura 11.17. Diagrama de flujo del módulo de gestión de sesión con la MCU y programación en RAM.

11.2.11 Submódulo de ejecución de campañas de inyección de fallas

El submódulo de ejecución de campañas de inyección de fallas constituye el núcleo operativo de la herramienta desarrollada, ya que es el responsable de coordinar la aplicación sistemática de fallas sobre la MCU, así como la adquisición, el almacenamiento y la posterior clasificación de los resultados de cada experimento.

Este submódulo fue diseñado para soportar campañas extensas y repetibles de FI y permitir la automatización completa del flujo experimental; su arquitectura permite la inyección en ubicaciones internas del sistema, garantizando flexibilidad y escalabilidad en la definición y ejecución de los experimentos.

Descripción general del funcionamiento del submódulo

El submódulo opera a partir de una lista estructurada de fallas definida en un archivo CSV, en donde cada entrada describe los parámetros necesarios para la inyección, tales como la ubicación, el tipo de falla, la máscara asociada, la dirección de inyección y la dirección de interrupción.

De manera general, el submódulo realiza las siguientes funciones:

1. Inicialización del entorno de ejecución.

Antes de iniciar la campaña, el submódulo crea automáticamente un directorio de trabajo asociado a la ejecución actual, en donde se almacenan los archivos de resultados y cualquier información generada durante el proceso. Esta estructura permite aislar cada campaña, garantizar trazabilidad y evitar la sobrescritura de ejecuciones previas.

2. Carga y validación de la campaña de fallas.

El submódulo interpreta el archivo CSV de entrada y construye internamente una lista de fallas válidas, descartando aquellas entradas inconsistentes o incompletas. Esta información define el conjunto de experimentos que conforman la campaña, incluyendo contadores internos para seguimiento de fallas no aplicadas, errores de lectura/escritura y bloqueos del sistema.

3. Gestión de sesiones de depuración.

El submódulo gestiona la sesión de depuración de la MCU a lo largo de toda la campaña de inyección de fallas. Si existe una sesión activa, esta se reutiliza para la ejecución de la falla actual. En caso contrario, se inicia una nueva sesión antes de comenzar el experimento, asegurando que la MCU esté detenida y bajo control del depurador.

Para garantizar que cada inyección se realice desde un estado inicial reproducible, el submódulo restablece el *software* antes de cada experimento. Cuando la falla afecta regiones correspondientes a *flash* simulada en RAM, la MCU se reprograma para reconstruir correctamente el contenido en memoria. En los demás casos, se realiza un reinicio controlado de la MCU, preservando la imagen de *software* previamente cargada.

4. Sincronización mediante puntos de parada controlados.

Para garantizar que la inyección se realice en un instante de ejecución determinístico dentro del flujo del programa, el submódulo utiliza *breakpoints* ubicados en direcciones válidas del flujo de ejecución del *software*. Se establece un *breakpoint* temporal en la dirección definida para cada falla (direcciones de inyección), en donde se detiene la ejecución y se modifica el estado interno del sistema. Adicionalmente, se configura un *breakpoint* asociado a un bucle infinito previamente identificado, el cual actúa como referencia temporal fija y permite validar que el sistema alcanza un estado conocido tras la inyección.

5. Aplicación de la falla.

Una vez alcanzado la dirección de interrupción temporal asociado a la falla, el submódulo detiene la ejecución y modifica el estado interno del sistema de acuerdo con el tipo de falla definido en la campaña. La alteración se realiza mediante operaciones directas de lectura-modificación-escritura sobre la dirección de inyección, aplicadas tanto sobre registros como sobre regiones de memoria (RAM o *flash* simulada en RAM), según la ubicación especificada.

6. Captura del estado interno.

El submódulo adquiere *snapshots*⁵ del estado interno de la MCU en instantes de ejecución controlados, específicamente antes y después de cada inyección. Estos *snapshots* incluyen el contenido de los registros del núcleo y de las regiones de memoria previamente configuradas para monitoreo, permitiendo establecer una comparación directa entre el estado previo y el estado posterior a la aplicación de la falla.

7. Clasificación y registro de resultados.

Cada experimento es evaluado de manera sistemática con base en la comparación entre los estados capturados antes y después de la inyección, así como en el comportamiento observado durante la ejecución posterior. Este proceso permite identificar fallas toleradas (sin efecto observable), fallas no aplicadas, errores de escritura, errores de lectura y bloqueos del sistema (*hangs*).

Los resultados de cada inyección son almacenados en un archivo de salida en formato CSV, en donde cada entrada registra la información asociada a la falla (como la dirección de inyección, la dirección de interrupción, el tipo de falla aplicado y la clasificación obtenida) permitiendo su posterior procesamiento estadístico y la generación de reportes.

La figura 11.18 muestra el diagrama de flujo general del submódulo de ejecución de campañas inyección de fallas, desde la carga de la campaña hasta el registro y clasificación de los resultados.

⁵Captura del estado interno del sistema en un instante específico de ejecución, utilizada para el análisis comparativo y la evaluación del impacto de la falla.

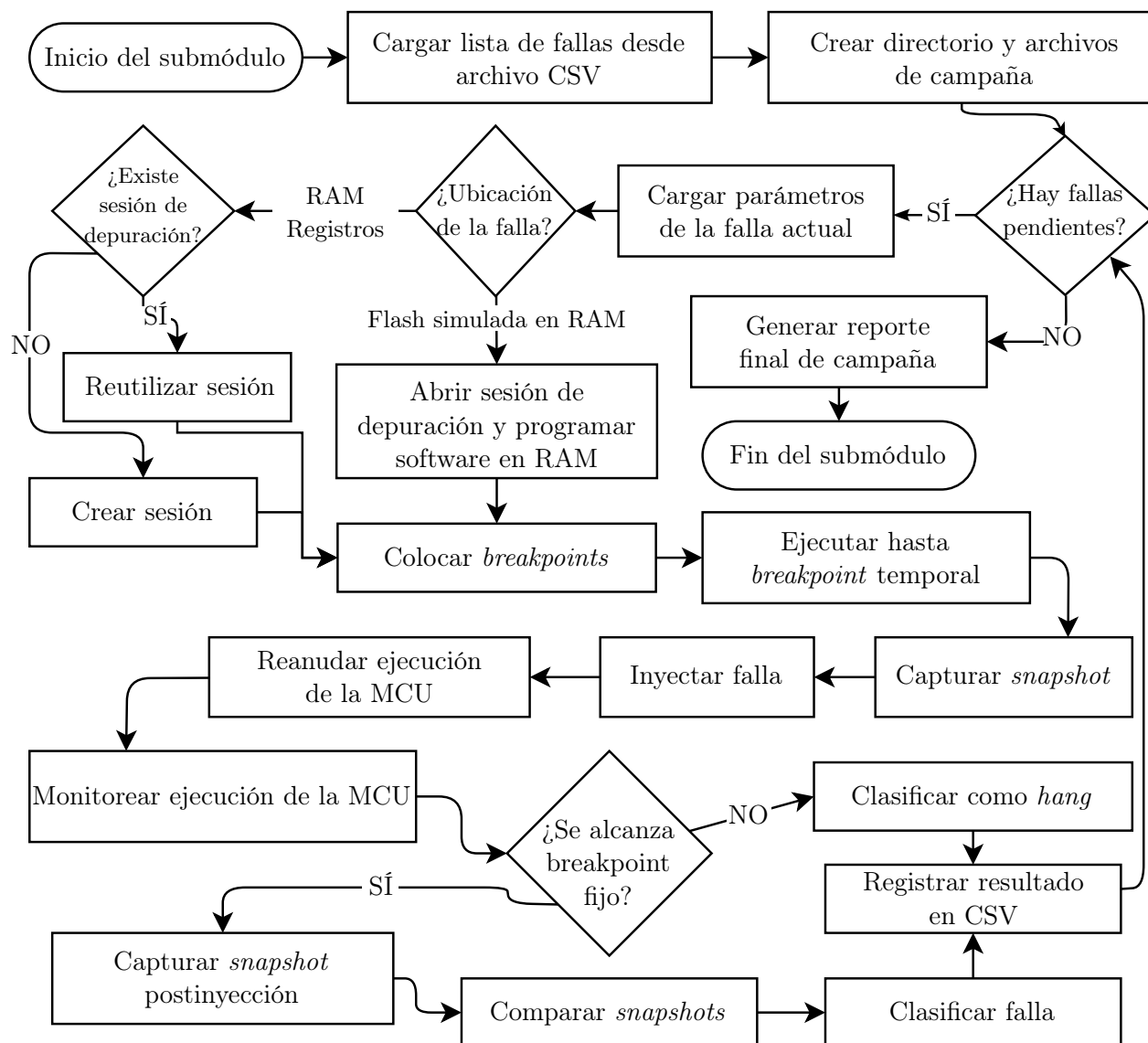


Figura 11.18. Diagrama de flujo del submódulo de ejecución de campañas de inyección de fallas.

11.2.12 Submódulo de detección de última campaña

Durante la etapa de análisis y procesamiento de los resultados de las campañas de FI, resulta fundamental contar con un mecanismo que permita acceder a los archivos generados durante la realización de cada campaña. Dado que cada campaña produce una nueva carpeta de trabajo identificada mediante una convención de nombres asociada a marcas temporales, la selección manual de dichas rutas puede introducir errores y afectar la trazabilidad de los resultados.

Descripción general del funcionamiento del submódulo

El submódulo de detección de última campaña se encarga de localizar automáticamente la carpeta correspondiente a la campaña de inyección de fallas más reciente dentro del directorio de trabajo del proyecto. Para ello, realiza una exploración del sistema de archivos y determina cuál de las carpetas disponibles corresponde a la campaña más reciente, tomando como referencia su información temporal.

El funcionamiento general del submódulo se compone de las siguientes etapas:

1. Determinación del directorio base de trabajo.

El submódulo identifica automáticamente el directorio en el que se encuentra el propio *script* del submódulo, el cual se utiliza como punto de partida para la búsqueda de campañas de FI.

2. Exploración de subcarpetas disponibles.

A partir del directorio base, se examinan las subcarpetas existentes con el fin de localizar aquellas asociadas a campañas experimentales, utilizando un prefijo identificador definido para las campañas.

3. Filtrado de carpetas válidas.

El submódulo selecciona únicamente las subcarpetas que cumplen con el patrón de identificación de campañas de FI, descartando cualquier otro elemento que no corresponda a resultados experimentales.

4. Verificación de disponibilidad de campañas.

Se comprueba la existencia de al menos una carpeta válida. En caso contrario, el submódulo genera una excepción que detiene la ejecución y notifica la ausencia de campañas disponibles.

5. Selección de la campaña más reciente.

Las carpetas válidas se ordenan según su fecha de modificación, permitiendo identificar automáticamente la campaña más reciente y obtener su ruta absoluta como resultado del submódulo

La figura 11.19 muestra el diagrama de flujo del submódulo de detección de última campaña.

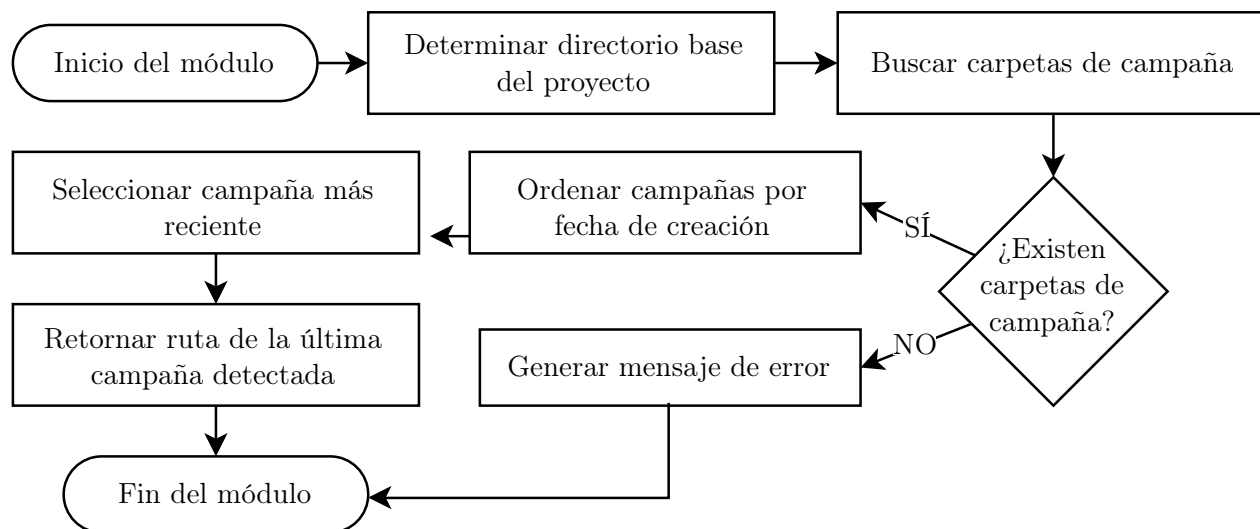


Figura 11.19. Diagrama de flujo del submódulo de detección de última campaña.

11.2.13 Submódulo de ejecución de referencia

Durante las campañas de inyección, es fundamental contar con un registro del comportamiento del DUT sin fallas para evaluar el efecto de las inyecciones sobre la MCU. El submódulo de ejecución de referencia tiene como objetivo principal ejecutar el *software* sin modificaciones intencionales, capturando el estado interno del sistema en un punto de ejecución definido, de manera que los datos obtenidos sirvan como base para clasificar los efectos generados por la falla.

Durante las campañas de inyección, es fundamental contar con un registro del comportamiento del DUT sin fallas para evaluar el efecto de las inyecciones sobre la MCU. El submódulo de ejecución de referencia tiene como objetivo ejecutar el *software* sin modificaciones intencionales y capturar el estado interno del sistema en un punto fijo de observación, correspondiente a la dirección asociada al bucle infinito *while(1)* de la función principal, identificada previamente por el submódulo de detección de bucles. Esta misma dirección se utiliza durante las ejecuciones con inyección, permitiendo comparar el estado nominal y el estado posterior a la falla en una misma posición del flujo de ejecución.

La figura 11.20 presenta el diagrama de flujo del submódulo de ejecución de referencia.

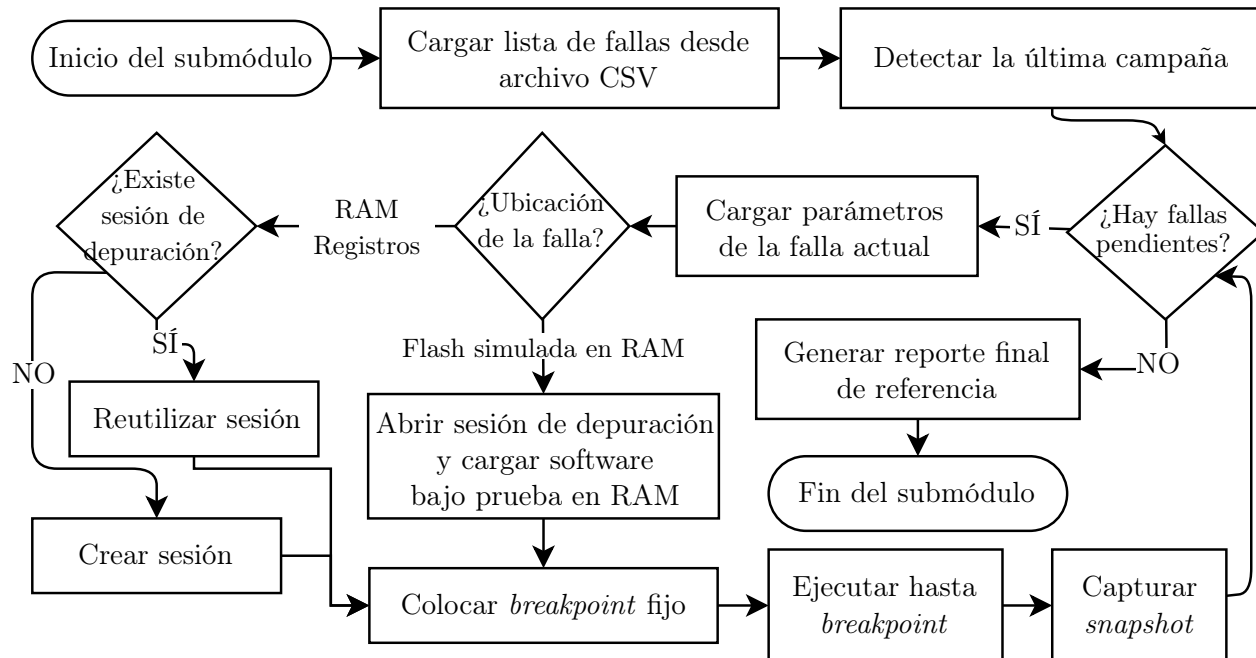


Figura 11.20. Diagrama de flujo del submódulo de ejecución de referencia.

Descripción general del funcionamiento del submódulo

El funcionamiento del submódulo se organiza en las siguientes etapas, orientadas a supervisar la ejecución del *software* y recopilar el estado de la MCU de manera controlada durante las campañas de referencia:

1. Inicialización del entorno experimental.

El submódulo recibe como entrada la lista de fallas de la campaña y las rutas del archivo ELF, preparando el entorno de depuración necesario para la ejecución controlada de la MCU; según la ubicación de la falla, la sesión puede requerir reprogramar la MCU con el *software* bajo prueba en RAM, como ocurre con las fallas en *flash* simulada en RAM, o simplemente supervisar la ejecución de la MCU sin reprogramarla, como sucede con las fallas en RAM o registros.

2. Detección de la campaña activa.

Antes de iniciar la ejecución de referencia, se identifica la carpeta correspondiente a la última campaña activa mediante el submódulo de detección de última campaña. Esta carpeta se utiliza para organizar y almacenar la información capturada, asegurando que los datos de referencia queden asociados correctamente a la campaña correspondiente.

3. Preparación de la campaña de referencia

Se interpreta el archivo de la campaña para identificar las fallas definidas y sus ubicaciones en el sistema; aunque durante la ejecución de referencia no se aplican fallas, esta información permite determinar qué elementos deben ser observados. Para esta ejecución se utiliza una única dirección de interrupción fija como punto de interrupción controlada al final del flujo de ejecución.

4. Inicialización de la sesión de depuración y colocación de la interrupción.

Se abre y configura la sesión de depuración en función la ubicación de la falla, asegurando que la MCU esté lista para la ejecución controlada. Posteriormente, se establece la dirección de interrupción fija al final del flujo de ejecución para detener la MCU de manera controlada y permitir la captura de su estado.

5. Ejecución y captura del comportamiento de referencia.

Con la sesión preparada y la interrupción fija colocada, se ejecuta el *software* sin aplicar ninguna falla, permitiendo que la MCU alcance de forma segura el punto de interrupción; en ese momento se realiza un *snapshot* de los registros y de la memoria, obteniendo los valores que reflejan el comportamiento normal del sistema.

6. Almacenamiento de la información de referencia.

La información capturada se organiza de forma estructurada dentro de la carpeta de la campaña activa, constituyendo la referencia del sistema que será utilizada posteriormente para analizar y clasificar los efectos observados durante las campañas de FI.

11.2.14 Submódulo de análisis de resultados

Una vez concluida la ejecución de una campaña de inyección de fallas, es necesario realizar un análisis de los resultados con el fin de evaluar el impacto real de las fallas sobre el comportamiento del DUT. Este análisis debe considerar tanto los efectos observables como el resultado del proceso de inyección de fallas, permitiendo diferenciar entre las fallas que produjeron alteraciones en el DUT y aquellas que no generaron cambios o no llegaron a aplicarse.

Con este propósito, se desarrolló el submódulo de análisis de resultados, el cual se encarga de procesar la información generada durante la campaña de FI, correlacionar los distintos archivos de resultados y clasificar cada falla en función de su efecto técnico en el DUT. Este submódulo constituye la etapa final del flujo experimental y permite convertir los datos experimentales en resultados claros y medibles.

Descripción general del funcionamiento del submódulo

El submódulo de análisis de resultados opera como un proceso de postprocesamiento totalmente automatizado que toma como entrada los archivos generados durante la campaña de FI. Su funcionamiento se basa en la comparación entre el comportamiento libre de fallas (referencia) de la MCU y el comportamiento observado después de la aplicación de cada falla, así como en la interpretación de los estados reportados por el inyector. De manera general, el funcionamiento del submódulo se divide en las siguientes etapas:

1. Detección automática de la campaña activa.

El submódulo identifica la carpeta correspondiente a la campaña de inyección de fallas más reciente, lo que garantiza que el análisis se realice siempre sobre los resultados actuales.

2. Carga de archivos de resultados.

Una vez localizada la campaña activa, el submódulo carga los archivos que contienen los *snapshots* de la MCU correspondientes al estado de referencia y al estado posterior a la inyección, así como el registro de eventos generado por el inyector durante la ejecución experimental, estructurando esta información para su procesamiento y análisis posterior.

3. Resolución del estado final de cada falla.

El submódulo de análisis de resultados agrupa los eventos registrados por el inyector de acuerdo con el identificador de cada falla. A partir de estos registros, asigna un estado final que describe lo ocurrido durante el proceso de inyección y la ejecución posterior.

La tabla 11.2 presenta los estados finales considerados, el código utilizado para representarlos en la implementación y su significado. Estos estados permiten distinguir entre las fallas que no fueron inyectadas, aquellas en las que el valor leído después de la escritura no coincidió con el valor esperado, los casos en los que no fue posible realizar la lectura posterior a la escritura, los bloqueos ocurridos después de la inyección y las ejecuciones completadas correctamente.

Los estados finales y sus descripciones se presentan en español, mientras que los códigos de estado conservan su denominación en inglés para mantener correspondencia con la implementación.

Tabla 11.2. Estados finales y códigos de estado del proceso de inyección de fallas.

Estado final	Código de estado	Descripción
Falla no inyectada	STOP_ADDRESS_NOT_REACHED	La MCU no alcanzó el punto de inyección dentro del tiempo establecido. Por lo tanto, no se intentó modificar el recurso seleccionado.
No escrita	WRITE_FAILED	La MCU alcanzó el punto de inyección y la herramienta ejecutó la orden de escritura. Sin embargo, el valor leído inmediatamente después no coincidió con el valor modificado esperado.
No leída	READBACK_FAILED	Después de ejecutar la orden de escritura, no fue posible leer el contenido del recurso seleccionado. Por lo tanto, no pudo determinarse el valor final almacenado en el recurso.
<i>Hang</i> después de la falla	POST_FAULT_HANG	El valor leído después de la escritura coincidió con el valor modificado esperado, confirmando que la falla fue inyectada. Sin embargo, después de reanudar la ejecución, la MCU no alcanzó el punto de observación dentro del tiempo establecido.
Falla inyectada ("OK")	SUCCESS	El valor leído después de la escritura coincidió con el valor modificado esperado y la MCU alcanzó el punto de observación dentro del tiempo establecido.

4. Comparación del estado interno del sistema.

El submódulo compara el estado de referencia de la MCU con el estado posterior a cada falla, revisando los registros críticos del núcleo del procesador, los registros de propósito general y los bloques de memoria o registros definidos como relevantes para la campaña. Esto permite identificar cambios inesperados provocados por la falla y evaluar su impacto en el funcionamiento interno del DUT.

5. Clasificación técnica de las fallas.

A partir de la comparación realizada y del estado final reportado por el inyector, cada falla se clasifica en categorías como fallas silenciosas, fallas propagadas o fallas no aplicadas. Las fallas silenciosas son aquellas que no producen efectos observables sobre el sistema, y la MCU continúa su funcionamiento normal; las fallas propagadas generan cambios que se reflejan en el estado interno de la MCU, afectando registros, memoria u otro comportamiento observable; mientras que las fallas no aplicadas son aquellas que no lograron implementarse correctamente y, por lo tanto, no tuvieron impacto en el DUT. Esta clasificación permite determinar de manera clara cómo afectó cada falla al sistema, facilitando un análisis técnico más preciso del comportamiento de la MCU frente a la campaña de inyección.

6. Generación de resultados consolidados.

Los resultados del análisis se almacenan en un archivo estructurado que resume la clasificación de cada falla y proporciona información detallada sobre los efectos observados, como los registros y bloques de memoria modificados y la correcta inyección de las fallas. Este archivo constituye la salida principal del submódulo y sirve como base para análisis posteriores, comparaciones entre diferentes campañas de inyección y evaluación cuantitativa del comportamiento de la MCU.

7. Visualización de resultados.

Finalmente, el submódulo genera representaciones gráficas que resumen tanto la clasificación técnica de las fallas como los estados finales reportados por el submódulo de inyección de campaña de FI, proporcionando una visión global del comportamiento de la MCU frente a la campaña de inyección y facilitando la interpretación de los datos.

La figura 11.21 muestra el diagrama de flujo del submódulo de análisis de resultados, describiendo el proceso de carga, comparación y clasificación de las fallas, así como la generación y visualización de los resultados consolidados.

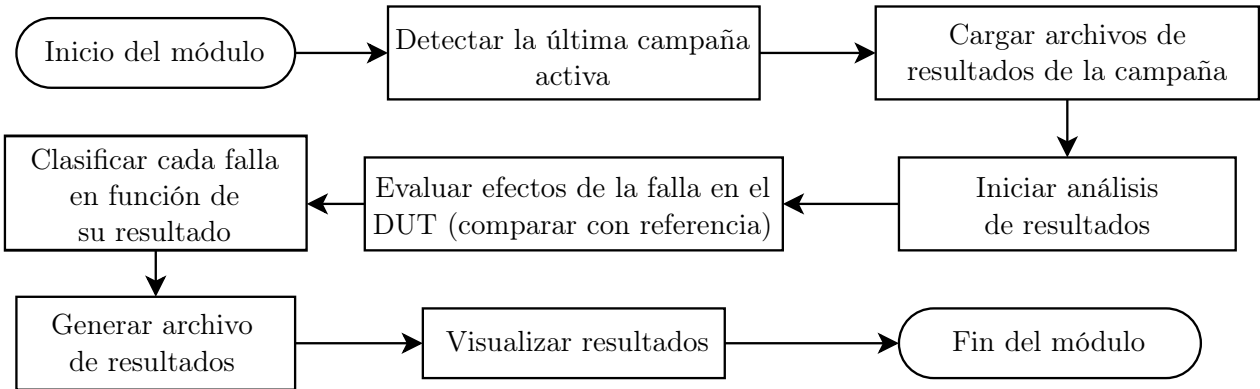


Figura 11.21. Diagrama de flujo del submódulo de análisis de resultados.

11.2.15 Submódulo de visualización y exploración de campañas

Una vez realizado el análisis de los resultados de una campaña de FI, es necesario contar con una herramienta que permita al usuario inspeccionar, validar e interpretar de forma interactiva la información generada. Dado el volumen de datos producido durante las campañas experimentales, la revisión manual mediante archivos individuales puede resultar poco eficiente y propensa a errores.

Con esto en mente, se desarrolló el submódulo de visualización y exploración de campañas, el cual proporciona una interfaz gráfica interactiva para la consulta estructurada y eficiente de los resultados experimentales. Este submódulo permite explorar campañas completas, visualizar métricas agregadas, consultar tablas de datos y analizar el comportamiento del sistema objetivo para cada falla de manera detallada e individual.

Descripción general del funcionamiento del submódulo

El submódulo de visualización y exploración de campañas opera como una interfaz que procesa y muestra los archivos de resultados generados durante la ejecución y el análisis de una campaña de FI. Su funcionamiento se basa en la detección automática de campañas disponibles, la selección interactiva de la campaña de interés y la presentación dinámica e interactiva de la información asociada.

De manera general, el funcionamiento del submódulo se divide en las siguientes etapas:

1. Detección automática de campañas disponibles.

El submódulo explora el directorio base y detecta automáticamente todas las carpetas asociadas a campañas de FI. Las campañas detectadas se ordenan por fecha, permitiendo acceder tanto a campañas históricas como a la más reciente.

2. Selección interactiva de la campaña.

A través de la interfaz, el usuario selecciona la campaña que desea analizar mediante un menú interactivo. Por defecto, el sistema propone la última campaña detectada, facilitando el flujo de trabajo en campañas consecutivas.

3. Carga de archivos de resultados.

Una vez seleccionada la campaña, el submódulo carga automáticamente los archivos de resultados, preparando los datos para su correcta visualización.

4. Resumen estadístico de la campaña.

El submódulo presenta un resumen de las fallas, mostrando el total de fallas analizadas y su distribución según la clasificación y el estado reportado. Esto permite al usuario tener una visión general del comportamiento de la campaña de manera rápida y sencilla.

5. Visualización de resultados gráficos.

Se integran representaciones gráficas que muestran la clasificación técnica de las fallas y los estados finales reportados, permitiendo una evaluación rápida del impacto global de la campaña.

6. Exploración tabular de los datos experimentales.

Se presentan tablas interactivas que permiten inspeccionar con detalle todos los conjuntos de datos de la campaña, facilitando la exploración completa de la información.

7. Análisis detallado por identificador de falla.

El submódulo permite seleccionar un identificador de falla específico y visualizar de forma comparativa el estado del sistema antes y después de la inyección, así como la clasificación y el estado final asignado, facilitando el análisis individual de cada experimento.

La figura 11.22 muestra el diagrama de flujo del submódulo de visualización y exploración de campañas, describiendo el proceso desde la selección de la campaña hasta la carga, procesamiento y presentación interactiva de los resultados.

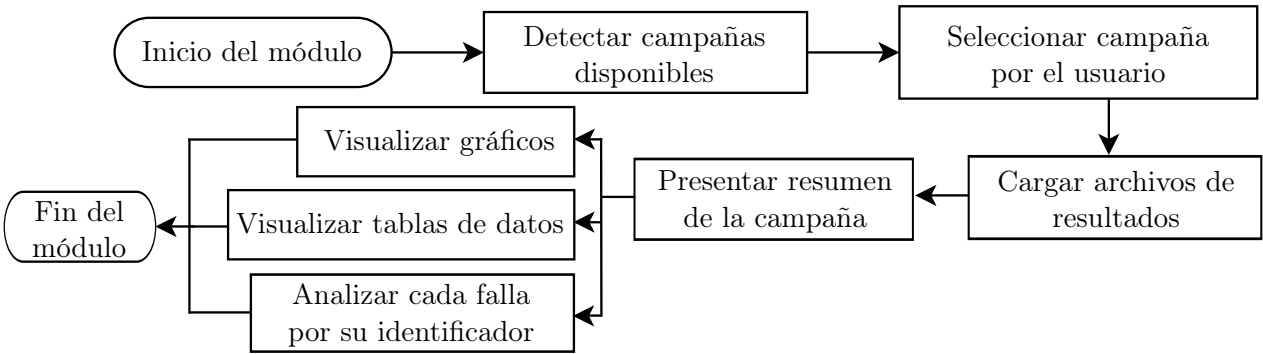


Figura 11.22. Diagrama de flujo del submódulo de visualización y exploración de campañas.

11.2.16 Submódulo de ejecución integral

El proceso de FI en una MCU requiere la coordinación precisa de múltiples etapas, que van desde el análisis estático del *software* hasta la ejecución de campañas experimentales, la adquisición del estado interno de la MCU y el análisis posterior de los resultados. La ejecución manual y aislada de cada uno de estos pasos incrementa la complejidad operativa de la herramienta y dificulta la repetibilidad de los experimentos.

Con el fin de integrar de manera coherente todas las funcionalidades desarrolladas, se diseñó el submódulo de ejecución integral, el cual actúa como un orquestador central, encargado de coordinar la ejecución secuencial de los distintos submódulos que conforman la herramienta, permitiendo la ejecución automatizada de campañas completas bajo distintos modos de operación.

Descripción general del funcionamiento del submódulo

El submódulo de ejecución integral funciona como una capa de control que gestiona el flujo de ejecución de la herramienta y asegura la coordinación adecuada entre los distintos submódulos involucrados en el proceso de inyección de fallas, organizando su funcionamiento en las siguientes etapas:

1. Inicialización del entorno de ejecución.

El submódulo prepara el entorno necesario para la campaña, validando los archivos del *software* y los parámetros definidos por el usuario, y estableciendo las condiciones necesarias para la ejecución coordinada de los submódulos y la interacción con la MCU objetivo.

2. Configuración del flujo de ejecución.

A partir del modo de operación y los parámetros definidos, se establece la secuencia de etapas de la campaña

experimental, comenzando con el análisis del *software* para identificar las direcciones válidas y los registros de la MCU a partir del archivo SVD. Con base en esta información, se procede a generar o cargar la lista de fallas correspondiente y a organizar el flujo general de ejecución.

3. Ejecución de la campaña y registro del comportamiento.

El submódulo coordina la ejecución de la campaña de inyección de fallas mediante el control del flujo entre los submódulos encargados de la ejecución experimental, asegurando la correcta aplicación de las fallas y el registro del comportamiento del sistema objetivo.

4. Análisis y organización de resultados.

El submódulo coordina el procesamiento y la organización de los datos generados durante la campaña, asegurando su disponibilidad para su visualización y el análisis del comportamiento del sistema frente a las fallas inyectadas.

Las etapas descritas anteriormente se representan de forma secuencial en la figura 11.23, que muestra el flujo de control gestionado por el submódulo de ejecución integral y la coordinación de las fases principales de la campaña experimental. El diagrama incluye únicamente los submódulos que intervienen directamente en el flujo operativo principal, mientras que otros, previamente descritos, permanecen encapsulados dentro de los bloques representados al formar parte de su lógica interna.

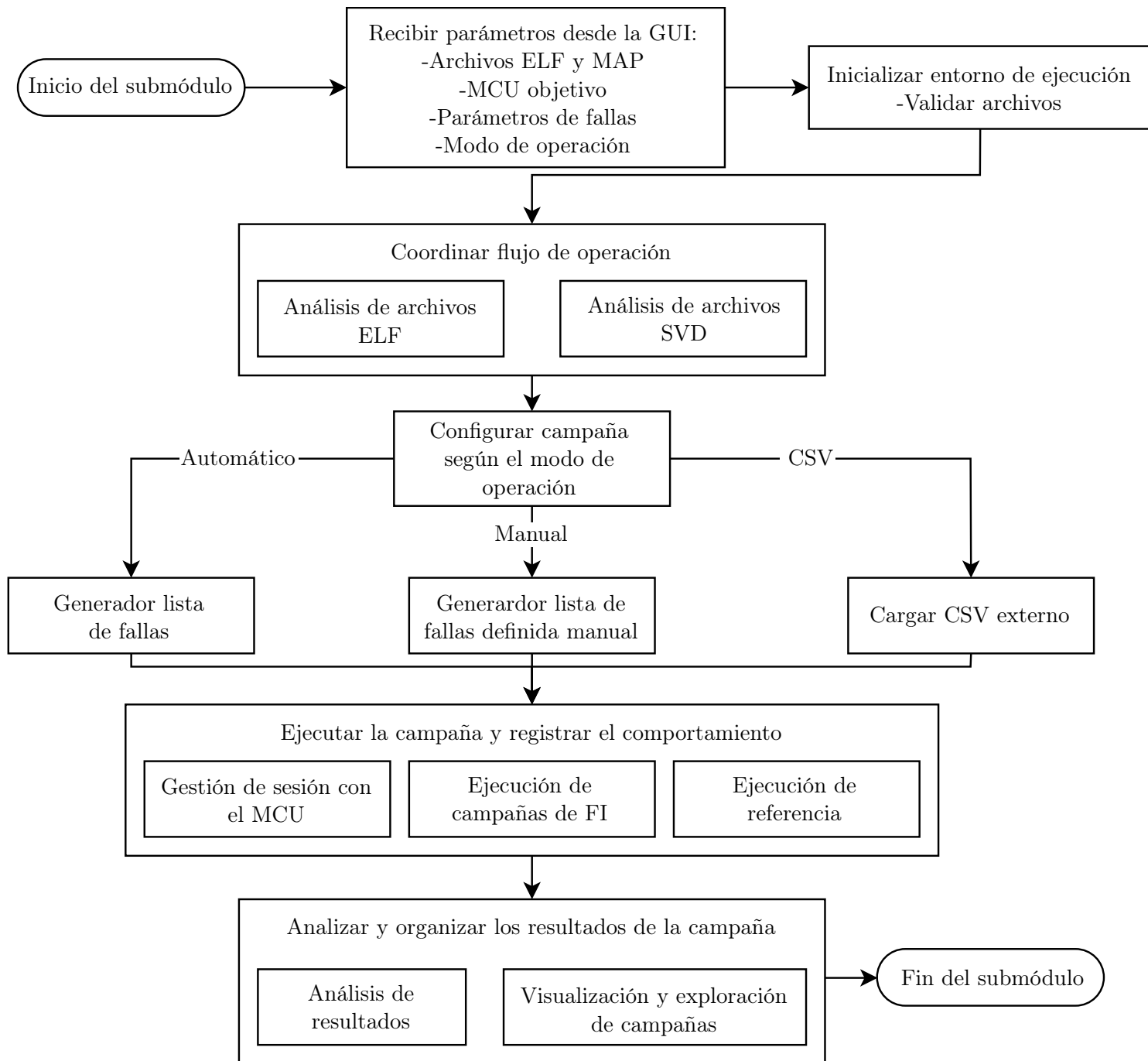


Figura 11.23. Diagrama de flujo del submódulo de ejecución integral

SECCIÓN IV
RESULTADOS Y CONCLUSIONES

12. RESULTADOS

Los resultados constituyen la evidencia empírica obtenida a partir de la ejecución de pruebas y experimentos durante el proceso de investigación, permitiendo observar el comportamiento del sistema bajo estudio en condiciones previamente definidas y proporcionando la base para su análisis y la formulación de conclusiones.

En el presente trabajo, los resultados se obtuvieron mediante la ejecución de campañas de FI sobre una MCU basada en la arquitectura Cortex-M4, empleando la herramienta desarrollada, automatizando la ejecución de las campañas, el control de la interacción con el DUT y la recopilación estructurada de la información generada durante cada experimento.

En este capítulo se presentan los resultados, organizados de acuerdo con los distintos modos de operación implementados en la herramienta de inyección. Asimismo, se describe el entorno experimental y las condiciones en que se realizaron las pruebas, con el fin de proporcionar el contexto necesario para su correcta interpretación.

12.1 *Entorno experimental*

Las campañas de inyección de fallas se ejecutaron sobre el DUT descrito en el capítulo de diseño de detalle, empleando la herramienta de inyección desarrollada y un *software* de aplicación previamente cargado en la MCU.

La herramienta permitió configurar y ejecutar las campañas, así como supervisar el estado del DUT y registrar los datos generados durante cada experimento. Este entorno proporcionó las condiciones necesarias para la ejecución controlada de las pruebas y la recopilación de los datos experimentales utilizados en el análisis de resultados.

12.1.1 *Software bajo prueba*

El *software* bajo prueba fue diseñado para utilizar distintos recursos de la MCU, incluyendo la RAM y los periféricos. Para lograrlo, se implementó el Juego de la Vida de Conway, tomando como referencia su uso como carga de trabajo en MiFIT, en donde fue seleccionado debido al elevado número de instrucciones requeridas por cada ejecución [52]. Este algoritmo es un autómata celular propuesto por John Conway que evoluciona en pasos discretos sobre una matriz bidimensional almacenada en memoria. En este algoritmo, el estado de cada celda se actualiza iterativamente en función del estado de sus celdas vecinas, lo que implica operaciones periódicas de lectura y escritura en memoria. Esta dinámica no solo produce accesos frecuentes a la memoria, sino que también genera cargas de trabajo periódicas sobre la MCU.

La interacción con los periféricos se implementó mediante GPIO, seleccionados porque no requieren *hardware* adicional. Su funcionamiento puede observarse a través de LED, que actúan como indicadores visuales de las tareas del sistema, permitiendo confirmar de manera inmediata que los periféricos responden durante la ejecución.

Además, se implementó el cálculo de un código de verificación de redundancia cíclica (CRC, por sus siglas en inglés) sobre los datos de memoria utilizados por el algoritmo. Aunque el CRC se almacena junto con los datos, su ubicación exacta no es determinante para la detección de fallas: cualquier alteración provocada por la inyección modifica el contenido de memoria y, por consiguiente, cambia el valor del CRC. Esto permite identificar de manera rápida si se ha producido algún cambio en la memoria, funcionando como un indicador complementario de integridad antes de analizar los *snapshots* completos.

12.1.2 *Configuración de las campañas de FI*

Con el propósito de evaluar el comportamiento del DUT ante distintas condiciones de inyección, se definió un conjunto de campañas experimentales considerando los modos de operación previamente descritos para la herramienta

desarrollada. Estas campañas permitieron realizar inyecciones de fallas sobre la memoria RAM, incluyendo tanto las variables del *software* de aplicación como el código ejecutable, el cual fue cargado y ejecutado directamente desde esta memoria. Asimismo, se realizaron inyecciones sobre registros asociados a periféricos, particularmente el periférico GPIO D, utilizado en el *software* de aplicación para el control de indicadores visuales mediante LED.

El uso del puerto GPIOD resulta relevante para este trabajo debido a que, al igual que los demás puertos GPIO de la MCU, su funcionamiento se controla mediante registros internos de configuración y estado, los cuales constituyen objetivos válidos para la inyección de fallas. En particular, GPIOD fue utilizado en el *software* bajo prueba, lo que permitió extender el análisis más allá de la memoria de datos y evaluar también los efectos de alteraciones en registros asociados a periféricos.

El número de fallas por campaña se estableció mediante un criterio experimental, considerando el objetivo de cada prueba y el tiempo requerido para su ejecución. La literatura muestra que el tamaño de las campañas puede variar en función del propósito del estudio; por ejemplo, MiFIT empleó 1000 inyecciones por registro, mientras que GOOFI-2 reportó que campañas con menos de 100 inyecciones fueron suficientes para identificar vulnerabilidades en el sistema evaluado [50, 52].

Con base en estos antecedentes, en las campañas automáticas sobre la memoria RAM y los registros de periféricos se emplearon 1000 fallas, tomando este valor como una referencia experimental para disponer de un número amplio de observaciones. En el caso de la memoria *flash* simulada en RAM, el número se redujo a 500 fallas debido al mayor tiempo requerido por este tipo de prueba, limitando a la mitad la duración aproximada de la campaña y manteniendo un conjunto de resultados para su análisis descriptivo. Por su parte, en el modo manual se emplearon campañas de 100 fallas, ya que su objetivo fue verificar la ejecución de inyecciones cuyos parámetros fueron definidos explícitamente por el usuario, y no realizar una exploración extensa de las posibles condiciones de falla. Adicionalmente, cada campaña fue reproducida mediante el modo CSV, lo que permitió repetir las condiciones de inyección y verificar la consistencia de los resultados.

La tabla 12.1 resume la configuración general de las campañas de inyección de fallas ejecutadas en el presente trabajo.

Tabla 12.1. Resumen de las campañas de inyección de fallas realizadas.

Campaña	Modo de operación	Ubicación	Número de fallas	Tipo de falla	Objetivo
1	Automático	RAM	1000	Todos (<i>bitflip</i> , <i>stuck-at-1</i> , <i>stuck-at-0</i>)	Evaluar el efecto de fallas en RAM durante la ejecución.
2	Automático	<i>Flash</i> simulada en RAM	500	Todos	Evaluar el efecto de fallas en memoria tipo no volátil simulada.
3	Automático	Registros (GPIO D)	1000	Todos	Evaluar el efecto de fallas en registros de periféricos.
4	Manual	RAM, <i>flash</i> simulada en RAM, registros (GPIO D)	100	Definido por el usuario	Evaluar fallas con parámetros de inyección completamente definidos.
5	CSV	RAM	1000	Predefinido	Verificar la reproducibilidad de las campañas.
6	CSV	<i>Flash</i> simulada en RAM	500	Predefinido	Verificar la reproducibilidad en memoria simulada.

Continúa en la siguiente página

Tabla 12.1. (Continuación) Resumen de las campañas de inyección de fallas realizadas.

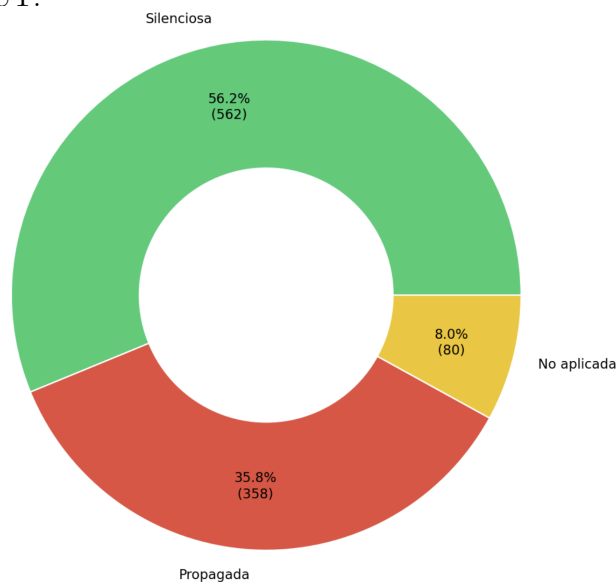
Campaña	Modo de operación	Ubicación	Número de fallas	Tipo de falla	Objetivo
7	CSV	Registros GPIO D	1000	Predefinido	Verificar la reproducibilidad en registros de configuración de periféricos.
8	CSV	Lista manual	100	Predefinido	Verificar la reproducibilidad de configuraciones de inyección previamente definidas.

Resultados de la campaña de inyección de fallas en RAM

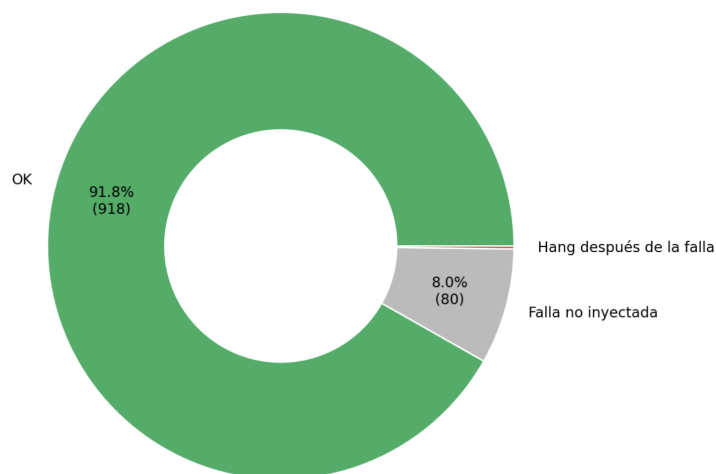
La figura 12.1a muestra la clasificación de los resultados de la campaña de FI en modo automático sobre la memoria RAM, en la cual se ejecutaron 1000 inyecciones considerando los tres tipos de falla. De las 1000 fallas inyectadas, el 56.2 % fueron clasificadas como fallas silenciosas, es decir, que no produjeron una modificación en el estado interno monitoreado del DUT; por otro lado, el 35.8 % fueron clasificadas como fallas propagadas, lo que implica que la alteración logró afectar el estado interno del sistema.

Finalmente, el 8.0 % fueron catalogadas como “no aplicadas”, correspondientes a casos en los que la inyección no llegó a ejecutarse efectivamente, ya sea porque la escritura de la falla no se realizó correctamente o porque la ejecución del programa no alcanzó la dirección de interrupción definida para la inyección, impidiendo que la alteración fuera aplicada sobre el DUT.

Estos resultados evidencian que más de la mitad de las fallas inyectadas en la RAM no generaron un cambio detectable en el estado interno evaluado, lo cual puede asociarse a múltiples factores, tales como variables que no participan en la lógica activa en el instante de la inyección, datos que son sobrescritos por el flujo normal del programa o regiones de memoria que no influyen directamente en las estructuras críticas del sistema. Esto indica que no toda alteración en RAM necesariamente impacta las variables o marcadores internos utilizados para determinar el comportamiento del DUT.



(a) Clasificación de los resultados de inyección de fallas en RAM.



(b) Estado final de ejecución del DUT tras la inyección en RAM.

Figura 12.1. Resultados de la campaña 1.

Sin embargo, un porcentaje considerable (35.8 %) logró propagarse, demostrando que la memoria RAM constituye un elemento sensible dentro del sistema, ya que más de un tercio de las alteraciones produjo un cambio en el estado

interno monitoreado. Este resultado confirma que la corrupción de datos en memoria volátil puede afectar variables activas durante la ejecución, alterando el comportamiento lógico del sistema sin requerir necesariamente un colapso completo.

En la figura 12.1b se observa que el 91.8 % finalizaron en estado “OK”, lo que indica que el sistema continuó su ejecución sin presentar un bloqueo total. Dentro de este conjunto se incluyen tanto las fallas silenciosas como aquellas fallas propagadas que no derivaron en un estado de *hang*. El 8.0 % corresponde a fallas no inyectadas, consistente con lo mostrado en la figura 12.1a, mientras que la fracción asociada a “*hang* después de la falla” es mínima (2 fallas), lo que indica que, aunque una proporción relevante de fallas se propagó, pocas condujeron a un estado de bloqueo completo del sistema.

Resultados de la campaña de inyección de fallas en memoria *flash* simulada en RAM

La figura 12.2a muestra los resultados de la campaña en modo automático realizada sobre la totalidad del espacio de memoria RAM utilizado durante la ejecución del DUT, abarcando tanto el *software* bajo prueba cargado en memoria como las variables de datos y las estructuras internas del programa.

En esta campaña se ejecutaron 500 inyecciones considerando los tres tipos de falla. Del total, el 48.0 % fueron clasificadas como fallas propagadas, el 43.6 % como fallas silenciosas y el 8.4 % como no aplicadas, correspondientes a casos en los que la inyección no se materializó o no tuvo efecto durante la ejecución observada.

A diferencia de la campaña realizada únicamente sobre la memoria de datos (campaña 1), en este caso la proporción de fallas propagadas supera ligeramente a la de fallas silenciosas. Este comportamiento es consistente con el hecho de que la inyección abarcó regiones críticas del sistema, en donde la alteración de instrucciones o direcciones de retorno puede comprometer directamente el flujo de control y la estabilidad de la ejecución.

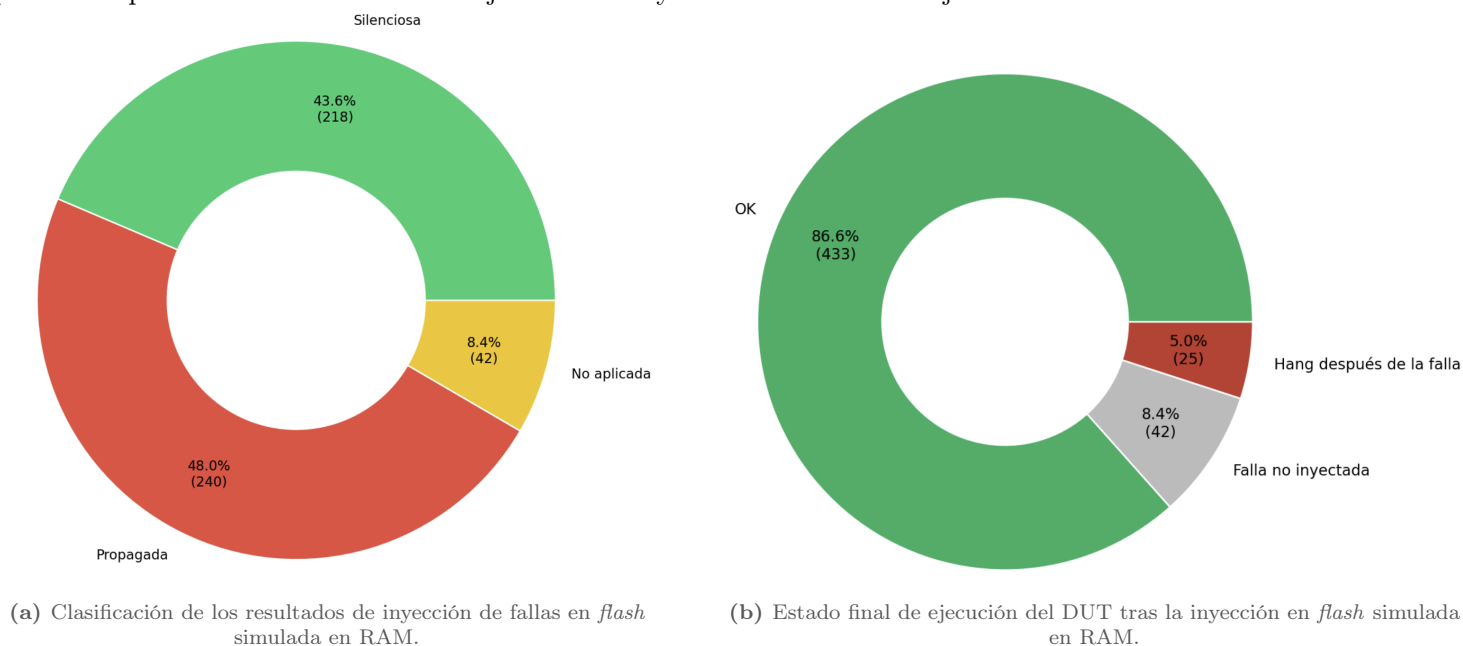


Figura 12.2. Resultados de la campaña 2.

La figura 12.2b presenta la clasificación del estado final del sistema posterior a la inyección. Se observa que el 86.6 % de las ejecuciones finalizaron en estado “OK”, mientras que el 5.0 % derivaron en estado de “*hang* después de la falla”. El 8.4 % restante corresponde a fallas no aplicadas, en concordancia con la clasificación.

En comparación con la campaña 1, se aprecia una mayor incidencia de comportamientos críticos, lo que confirma que la afectación del espacio completo de memoria activa incrementa la vulnerabilidad del sistema frente a fallas inyectadas.

Resultados de la campaña de inyección de fallas en regsitros de configuración de periféricos

La figura 12.3a muestra la clasificación general de los resultados de la campaña en modo automático sobre los registros de configuración asociados al periférico GPIO D. De las 1000 fallas inyectadas, el 52.4 % fueron clasificadas como fallas propagadas, el 30.0 % como fallas silenciosas y el 17.6 % como no aplicadas.

A diferencia de las campañas realizadas sobre memoria RAM y *flash* simulada en RAM, en este caso se observa una proporción aún mayor de fallas propagadas. Más de la mitad de las alteraciones inducidas en los registros del periférico generaron un efecto funcional observable, lo que evidencia la alta sensibilidad de estos elementos frente a modificaciones inducidas.

Este comportamiento puede explicarse por la función de los registros de periféricos, los cuales controlan directamente parámetros de operación tales como configuración de pines, modos de salida o estados lógicos. Una alteración en estos registros impacta de manera inmediata el comportamiento externo del sistema.

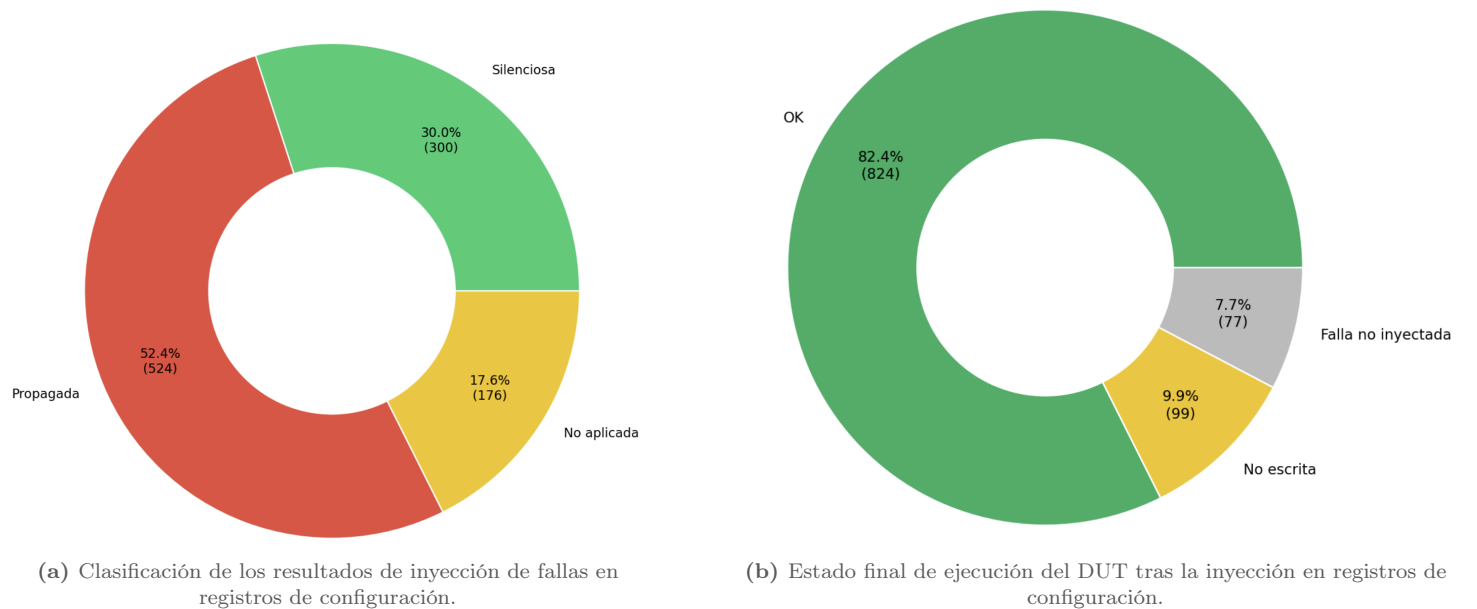


Figura 12.3. Resultados de la campaña 3.

La figura 12.3b presenta la clasificación del estado final del sistema posterior a la inyección. Se observa que el 82.4 % de las ejecuciones finalizaron en estado “OK”. Asimismo, el 9.9 % fueron clasificados como “no escrita”, indicando que la modificación del registro no logró efectuarse correctamente. El 7.7 % restante corresponde a fallas no aplicadas, en concordancia con la clasificación.

En comparación con la campaña realizada sobre memoria *flash* simulada en RAM, no se observa una incidencia significativa de estados de bloqueo total del sistema. Sin embargo, la elevada tasa de fallas propagadas confirma que los registros de periféricos constituyen un punto crítico desde el punto de vista funcional, ya que alteraciones en estos elementos se reflejan directamente en el comportamiento observable del DUT.

Resultados de la campaña de inyección de fallas en modo manual

La figura 12.4a muestra la clasificación de los resultados de la campaña ejecutada en modo manual. En esta campaña se realizaron 100 inyecciones definidas explícitamente por el usuario, incluyendo alteraciones sobre la RAM, memoria *flash* simulada en RAM y registros de configuración del periférico GPIO D.

El propósito de esta campaña no fue evaluar estadísticamente la vulnerabilidad del sistema, sino verificar el correcto funcionamiento del modo manual de la herramienta, confirmando que las fallas especificadas pudieran aplicarse de forma controlada y dirigida.

De las 100 fallas configuradas manualmente, el 33.0 % fueron clasificadas como fallas propagadas y el 17.0 % como fallas silenciosas. El 50.0 % restante fue catalogado como no aplicado.

La elevada proporción de fallas no aplicadas no representa una limitación del DUT, sino que responde al carácter controlado de la campaña, en la cual se seleccionaron deliberadamente distintas direcciones y tipos de fallas con fines de validación funcional del modo manual.

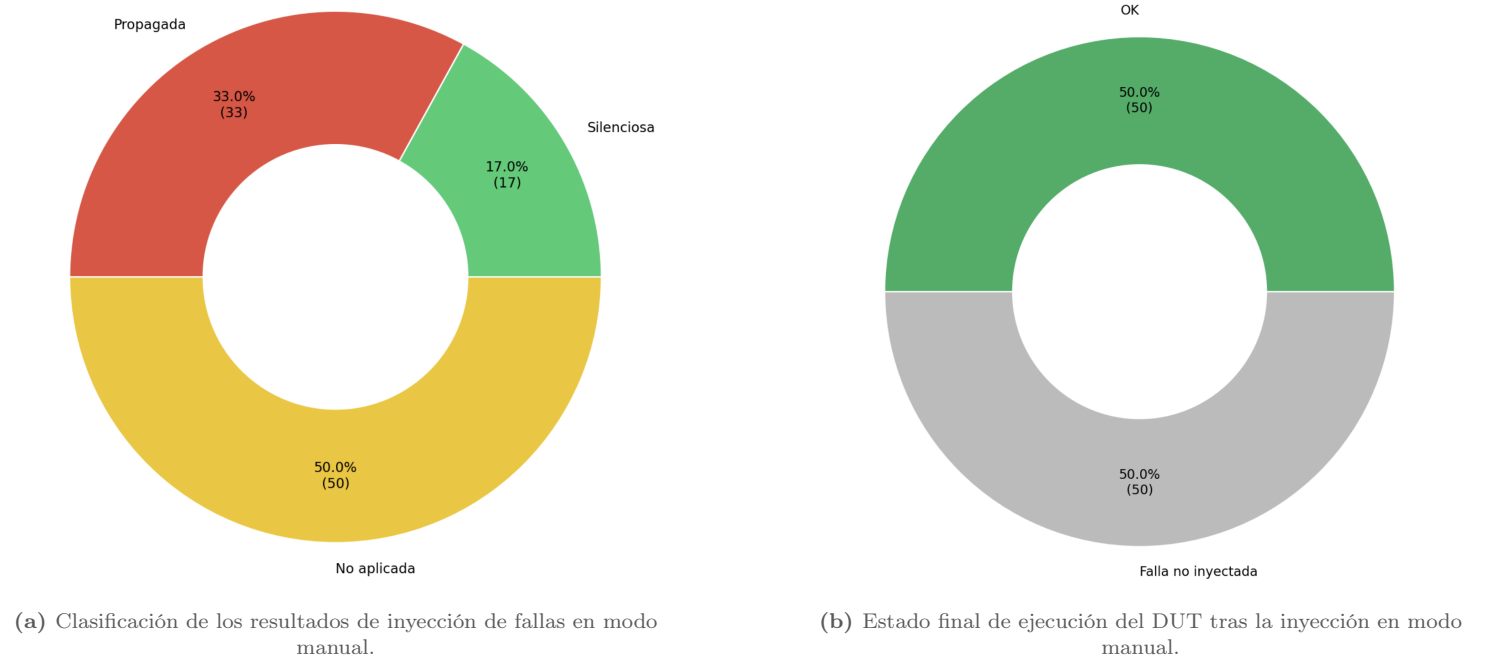


Figura 12.4. Resultados de la campaña 4.

La figura 12.4b presenta el estado final del sistema posterior a las inyecciones manuales. Se observa que el 50.0 % finalizaron en estado “OK”, mientras que el 50.0 % restante corresponde a fallas no aplicadas. No se registraron estados de bloqueo total del sistema durante esta campaña.

Estos resultados confirman que el modo manual permite ejecutar inyecciones específicas sin comprometer la estabilidad general del entorno experimental. Asimismo, se verifica que las fallas definidas por el usuario pueden aplicarse de manera selectiva y reproducible, validando el funcionamiento del módulo correspondiente dentro de la arquitectura de la herramienta desarrollada.

Análisis de reproducibilidad

Con el fin de evaluar la reproducibilidad de los experimentos, cada una de las campañas automáticas y manuales fue repetida mediante el modo CSV. Para ello, se utilizaron los archivos CSV generados durante las pruebas anteriores, los cuales conservan la dirección de inyección, el tipo de falla, el orden de ejecución y los parámetros asociados. Al repetir las campañas con las mismas condiciones de inyección, se esperan resultados idénticos a los registrados inicialmente.

En todos los casos (campañas 5, 6, 7 y 8), los resultados fueron idénticos a los registrados en sus respectivas campañas originales (campañas 1, 2, 3 y 4), tanto en la clasificación de las fallas como en su distribución porcentual y comportamiento final del sistema. No se observaron variaciones en el número de fallas silenciosas, propagadas, no aplicadas ni en los estados finales del DUT.

Esta coincidencia confirma que la herramienta desarrollada ejecuta las inyecciones de manera determinista bajo condiciones controladas, garantizando que una campaña previamente definida pueda reproducirse sin alteraciones en los resultados, lo que constituye un aspecto clave para asegurar la validez y confiabilidad de la herramienta desarrollada. La reproducibilidad observada valida no solo el mecanismo de inyección, sino también el módulo de exportación e importación de campañas mediante archivos CSV, consolidando así la solidez de la arquitectura implementada.

12.1.3 Validación de la herramienta en una MCU Cortex-M3

Con el propósito de evaluar la portabilidad, se realizaron pruebas adicionales sobre una tarjeta de desarrollo basada en el STM32F103C8T6, cuyo procesador corresponde a un ARM Cortex-M3. Para esta evaluación se utilizó el mismo flujo general de configuración, generación, inyección, registro y análisis empleado con la plataforma Cortex-M4.

Las pruebas se realizaron manteniendo los mismos modos de operación y el mismo número de inyecciones establecidos para las campañas principales. La principal modificación correspondió al periférico utilizado durante las pruebas sobre registros de configuración. Mientras que en la plataforma Cortex-M4 se emplearon registros asociados al GPIO D, en el STM32F103C8T6 se utilizaron registros correspondientes al GPIO B.

En la tabla 12.2 se resumen los resultados de las campañas realizadas mediante los modos automático y manual.

Tabla 12.2. Resultados en las campañas de FI sobre una MCU Cortex-M3.

Campaña / ubicación	Fallas	Silenciosas	Propagadas	No aplicadas
RAM	1000	562 (56.2 %)	370 (37.0 %)	68 (6.8 %)
Flash simulada en RAM	500	248 (49.6 %)	213 (42.6 %)	39 (7.8 %)
Registros GPIO B	1000	330 (33.0 %)	428 (42.8 %)	242 (24.2 %)
Modo manual	100	59 (59.0 %)	34 (34.0 %)	7 (7.0 %)

Los resultados muestran que las campañas pudieron ejecutarse sobre la plataforma Cortex-M3 utilizando los mismos modos de operación empleados durante la evaluación principal. En RAM predominaron las fallas silenciosas, con un 56.2 %, mientras que en la *flash* simulada en RAM se obtuvo una distribución más equilibrada entre fallas silenciosas (49.6 %) y propagadas (42.6 %). En los registros de configuración del GPIO B se observó una mayor proporción de fallas propagadas (42.8 %), además de un incremento en las fallas no aplicadas (24.2 %). Por su parte, la campaña manual presentó un 59.0 % de fallas silenciosas, un 34.0 % de fallas propagadas y un 7.0 % de fallas no aplicadas.

Posteriormente, las cuatro campañas fueron ejecutadas nuevamente mediante el modo CSV, obteniéndose los mismos resultados que en las campañas originales correspondientes. Esto permitió verificar la reproducibilidad de la herramienta de inyección.

12.2 Análisis de resultados

Los resultados de las campañas experimentales permiten identificar diferencias claras en el comportamiento del DUT en función de la ubicación de una falla inyectada, diferentes niveles de sensibilidad entre RAM, memoria *flash* simulada en RAM y registros de periféricos.

Las inyecciones realizadas sobre memoria RAM mostraron una mayor proporción de fallas silenciosas, superando el 50 % de los casos. Este comportamiento sugiere que una fracción significativa de las variables almacenadas no impacta directamente en el comportamiento observable del sistema o es sobrescrita durante la ejecución normal del programa. No obstante, el porcentaje de fallas propagadas confirma que la RAM constituye un elemento vulnerable cuando las alteraciones afectan variables activas dentro del flujo de ejecución.

En contraste, las inyecciones sobre memoria *flash* simulada en RAM evidenciaron un incremento en la proporción de fallas propagadas y la aparición de estados de bloqueo del sistema. Esto confirma que la corrupción del código ejecutable

tiene un impacto estructural más severo que la alteración de datos, ya que una modificación en instrucciones puede afectar directamente el flujo de control, generar saltos inválidos o alterar la lógica del programa, incrementando la probabilidad de colapso del sistema.

Por su parte, la inyección sobre los registros del periférico GPIO D presentó la mayor proporción de fallas propagadas entre las campañas automáticas, resultado consistente con la función de los registros de configuración, ya que estos controlan directamente la interacción del sistema con el entorno. No obstante, el impacto de este tipo de inyección también depende del periférico seleccionado y de su organización interna, dado que no todos presentan la misma estructura de configuración ni el mismo nivel de interdependencia entre registros, por lo que la sensibilidad observada puede variar según la arquitectura específica del periférico y el rol que desempeñe dentro del sistema.

De manera general, los resultados muestran una tendencia clara: la sensibilidad de una región no depende únicamente de su ubicación en memoria, sino del rol funcional que desempeña dentro del sistema. Mientras los datos pueden no ser determinantes en todo momento, el código define la estructura de ejecución y los registros de periféricos gobiernan la interfaz con el entorno, aumentando la visibilidad y el impacto de las alteraciones.

En cuanto al modo manual, los resultados confirmaron que las fallas definidas por el usuario pueden aplicarse de manera controlada sobre distintas regiones del DUT. Esta campaña no tuvo como objetivo evaluar la vulnerabilidad estadística, sino validar el funcionamiento del mecanismo de inyección manual, verificando que el módulo correspondiente opera conforme a lo esperado y permite inyecciones dirigidas y reproducibles.

Finalmente, el análisis de reproducibilidad constituye un aspecto clave del presente trabajo. La repetición de todas las campañas mediante el modo CSV arrojó resultados idénticos en cada caso, tanto en la clasificación de fallas como en su distribución porcentual y comportamiento final del sistema. Esta coincidencia confirma el carácter determinista de la herramienta desarrollada y respalda la solidez de la arquitectura implementada para la generación, almacenamiento y ejecución de campañas de inyección de fallas.

En conjunto, los resultados demuestran que la herramienta desarrollada no solo permite ejecutar campañas de inyección de fallas en distintas regiones del microcontrolador, sino que además proporciona un entorno controlado, determinista y reproducible para el análisis de vulnerabilidad en sistemas basados en ARM Cortex-M.

CONCLUSIONES

El presente trabajo tuvo como objetivo el desarrollo e implementación de una herramienta de inyección de fallas orientada a MCU ARM Cortex-M, empleando técnicas de OCD para la alteración controlada del estado interno del DUT. Durante su desarrollo se definió una arquitectura modular que integra la generación de campañas, su ejecución automatizada, la inyección sobre distintas regiones del microcontrolador y el análisis sistemático de los resultados obtenidos.

Los resultados experimentales confirman el cumplimiento de los objetivos planteados, permitiendo la inyección efectiva de fallas en memoria RAM, memoria tipo *flash* simulada y registros de configuración de periféricos. Las campañas ejecutadas evidenciaron comportamientos diferenciados según la región objetivo, demostrando que el rol funcional de cada componente influye directamente en la propagación y manifestación de las alteraciones inducidas. Mientras la memoria RAM presentó una mayor proporción de fallas silenciosas, la alteración del código ejecutable incrementó la probabilidad de propagación y afectación estructural del sistema, y los registros de periféricos reflejaron con mayor inmediatez las modificaciones debido a su interacción directa con el entorno.

Uno de los aportes centrales del trabajo es la verificación de reproducibilidad determinista en todas las campañas. La repetición mediante el modo CSV produjo resultados idénticos a los originales, validando la consistencia del mecanismo de inyección y la solidez del entorno experimental. Esta capacidad garantiza la repetición de escenarios bajo condiciones controladas, requisito esencial en estudios de tolerancia a fallas.

El modo manual confirmó además la posibilidad de realizar inyecciones dirigidas con parámetros definidos explícitamente, ampliando la flexibilidad de la herramienta y verificando el funcionamiento integral de su arquitectura.

La validación en un MCU distinto y en un sistema operativo diferente al empleado durante el desarrollo inicial demostró la portabilidad y adaptabilidad de la herramienta desarrollada, evidenciando que su funcionamiento no depende de una configuración específica de *hardware* o *software*, sino del cumplimiento de los requisitos establecidos en su diseño.

Desde una perspectiva metodológica, el trabajo no solo presenta una herramienta funcional, sino que establece una plataforma experimental estructurada para el análisis de vulnerabilidad y robustez en sistemas embebidos basados en ARM Cortex-M, con potencial de extensión hacia evaluaciones más avanzadas.

En el contexto de aplicaciones críticas, como plataformas *CubeSat*, la posibilidad de evaluar de manera controlada el impacto de alteraciones internas constituye un aporte relevante para el diseño de sistemas más robustos y confiables.

En síntesis, se consolida una herramienta de inyección de fallas funcional, determinista, reproducible y portable, que cumple con los requisitos definidos en su diseño y proporciona un entorno experimental sólido para el estudio de tolerancia a fallas en MCU ARM Cortex-M.

TRABAJO A FUTURO

En este trabajo se presentó el desarrollo de una herramienta de inyección de fallas mediante técnicas de depuración basadas en OCD, capaz de modificar de manera controlada los valores almacenados en RAM y en registros de configuración de periféricos de un microcontrolador con arquitectura ARM Cortex-M. Esta herramienta cumple con los objetivos planteados y surge de la necesidad de contar con una plataforma experimental para la simulación de los efectos *upset*, asociados a fenómenos de radiación espacial en sistemas embebidos.

No obstante, el trabajo realizado abre diversas oportunidades de mejora y expansión que pueden fortalecer su alcance experimental y ampliar sus capacidades de análisis. En este sentido, se proponen las siguientes líneas de trabajo a futuro:

- Extender los modelos de falla implementados, incorporando efectos como SET, SEFI y MBU, con el fin de aproximar el modelo lógico de inyección a los fenómenos físicos reales producidos por la radiación espacial.
- Desarrollar mecanismos de inyección en tiempo real que permitan inducir fallas sin detener completamente la ejecución de la MCU, permitiendo representar de manera más fiel las condiciones operativas de un sistema embebido expuesto al entorno espacial.
- Fortalecer el análisis de resultados, dado que actualmente la evaluación se basa en la comparación estática entre un estado libre de fallas y un estado con fallas. Como mejora, se propone incorporar mecanismos de análisis dinámico para estudiar la propagación temporal de los errores y su impacto durante la ejecución del sistema.
- Implementar mecanismos para la detección de SDC, mediante la validación de salidas esperadas o el uso de firmas de comportamiento del *software* bajo prueba, permitiendo identificar casos en los que el sistema continúa operando, pero produce resultados incorrectos.
- Integrar modelos probabilísticos de tasa de eventos *upset* asociados a distintos entornos orbitales, permitiendo generar campañas de inyección basadas en condiciones representativas de misiones espaciales reales y evaluar la confiabilidad del sistema en función del entorno de operación.
- Evaluar la herramienta utilizando un *software* bajo prueba con una aplicación espacial, como rutinas asociadas al control de actitud, adquisición de sensores, gestión de telemetría o procesamiento de datos a bordo. Esto permitirá analizar el efecto de las fallas inyectadas sobre funciones más cercanas a las condiciones de operación de una misión espacial.

Estas mejoras permitirán ampliar las capacidades de la herramienta desarrollada, consolidándola como una plataforma más completa para el análisis de confiabilidad y tolerancia a fallas en sistemas embebidos, particularmente en aplicaciones espaciales en donde la exposición a radiación representa un factor crítico.

BIBLIOGRAFÍA

- [1] «COTS Components in Spacecraft Systems: Understanding the Risk,» inglés, National Aeronautics y Space Administration, Technical Report NASA/TM-2014-218261, 2015.
- [2] NASA. «Sputnik 1,» visitado 5 de jun. de 2025. dirección: <https://www.nasa.gov/image-article/sputnik-1/>
- [3] W. J. Larson y J. R. Wertz, *Space Mission Analysis and Design* (Space Technology Library), inglés, 3.^a ed. El Segundo, California: Microcosm Press y Kluwer Academic Publishers, 2005, Edited by Wiley J. Larson and James R. Wertz; coordination by Douglas Kirkpatrick and Donna Klungle.
- [4] E. S. Agency, *Los satélites*, https://www.esa.int/kids/es/Aprende/Tecnologia/El_espacio_util/Los_satelites, 2016. visitado 18 de oct. de 2025.
- [5] J. J. Sellers, W. J. Astore, R. B. Giffen y W. J. Larson, *Understanding Space: An Introduction to Astronautics*, inglés, 3.^a ed. New York: McGraw-Hill Education, 2016.
- [6] A. Aeroespacial, *Gestión de los desafíos de diseño de sistemas de comunicaciones satelitales para órbita terrestre baja*, <https://actualidadaeroespacial.com/gestion-de-los-desafios-de-diseno-de-sistemas-de-comunicaciones-satelitales-para-orbita-terrestre-baja/>, 2025. visitado 18 de oct. de 2025.
- [7] C. D. Muñoz. «Introducción a los Satélites de Comunicaciones,» visitado 5 de jun. de 2025. dirección: <https://haciaelespacio.aem.gob.mx/revistadigital/articul.php?interior=61>
- [8] P. Fortescue, G. Swinerd y J. Stark, eds., *Spacecraft Systems Engineering*, inglés, 4.^a ed. Chichester, UK: John Wiley & Sons, 2011, ISBN: 978-0-470-75012-0.
- [9] Wikipedia, *CubeSat*, <https://es.wikipedia.org/wiki/CubeSat>, 2025. visitado 18 de oct. de 2025.
- [10] Administración Nacional de Aeronáutica y del Espacio (NASA). «Introducción al estado del arte de la tecnología de pequeñas naves espaciales,» visitado 7 de jun. de 2025. dirección: <https://www.nasa.gov/smallsat-institute/sst-soa/introduction/>
- [11] A. C. Tribble, *The Space Environment: Implications for Spacecraft Design*, inglés, 2.^a ed. Princeton University Press, 2017.
- [12] European Space Agency (ESA). «Atmospheric density changes with altitude,» visitado 19 de ago. de 2026. dirección: https://www.esa.int/ESA_Multimedia/Images/2009/06/Atmospheric_density_changes_with_altitude
- [13] Kintek Solution, *Why Can't Convection and Conduction Occur in Vacuum?* Accessed: 2026-05-14, 2024. dirección: <https://kindle-tech.com/faqs/why-cant-convection-and-conduction-occur-in-vacuum>
- [14] N. Oceanic y S. W. P. C. Atmospheric Administration, *Satellite Drag [Imagen]*, inglés, <https://www.swpc.noaa.gov/impacts/satellite-drag>. visitado 18 de oct. de 2025.
- [15] K. L. Bedingfield y R. D. Leach, «Spacecraft System Failures and Anomalies Attributed to the Natural Space Environment,» National Aeronautics y Space Administration, Marshall Space Flight Center, Huntsville, Alabama, NASA Reference Publication 1390, ago. de 1996.
- [16] National Geographic en Español, «Basura espacial: qué es y qué problemas puede generar,» *National Geographic en Español*, sep. de 2022. visitado 12 de jun. de 2025. dirección: <https://www.nationalgeographicla.com/espacio/2022/09/basura-espacial-que-es-y-que-problemas-puede-generar>
- [17] J. Espinosa y R. Paz, *La basura espacial podría convertirse en un problema sin control*, <https://www.gaceta.unam.mx/la-basura-espacial-podria-convertirse-en-un-problema-sin-control/>, feb. de 2022. visitado 19 de oct. de 2025.
- [18] I. N. de Investigaciones Nucleares, *Radiación cósmica de fondo y radiación cósmica*, <https://www.gob.mx/ini/n/articulos/radiacion-cosmica-de-fondo-y-radiacion-cosmica?idiom=es>, 2024. visitado 12 de oct. de 2025.

- [19] A. E. E. (ESA), *Radiación de fondo de microondas cósmica (CMB)*, https://www.esa.int/Science_Exploration/Space_Science/Cosmic_Microwave_Background_CMB_radiation, 2020. visitado 24 de oct. de 2025.
- [20] D. V. Reames, *Solar Energetic Particles: A Modern Primer on Understanding Sources, Acceleration and Propagation* (Lecture Notes in Physics), inglés, 2.^a ed. Springer, 2021, vol. 978, Open Access, <https://doi.org/10.1007/978-3-030-66402-2>, ISBN: 978-3-030-66401-5. DOI: 10.1007/978-3-030-66402-2
- [21] L. Gil, *Cosmic Radiation: Why We Should Not Be Worried*, inglés, <https://www.iaea.org/newscenter/news/cosmic-radiation-why-we-should-not-be-worried>, 2021. visitado 19 de oct. de 2025.
- [22] G. F. Knoll, *Radiation Detection and Measurement*, 4.^a ed. Wiley, 2010.
- [23] Shutterstock, *Earth's Magnetic Field – cross-section illustration*, inglés, <https://www.shutterstock.com/es/image-vector/earths-magnetic-field-cross-section-van-2526531269>, Ilustración vectorial bajo licencia Shutterstock, 2025. visitado 19 de oct. de 2025.
- [24] A. SINC, *La anomalía magnética del Atlántico Sur parece dividirse*, <https://www.agenciasinc.es/Noticias/La-anomalia-magnetica-del-Atlantico-sur-parece-dividirse>, Licencia CC 4.0, mayo de 2020. visitado 19 de oct. de 2025.
- [25] NASA Space Radiation Analysis Group. «What is Space Radiation?» Visitado 19 de ago. de 2026. dirección: <https://srag.jsc.nasa.gov/SpaceRadiation/What/What.cfm>
- [26] Shutterstock. «Defecto Frenkel: ilustraciones e imágenes de stock.» Consultado el 16 de mayo de 2026. dirección: https://www.shutterstock.com/es/search/defecto-frenkel?image_type=illustration
- [27] V. A. J. Van Lint, J. P. Raymond, A. R. Hart y M. L. Price, «Radiation Design Book,» NASA, inf. téc. NASA CR-152011, 1977. visitado 19 de ago. de 2026. dirección: <https://ntrs.nasa.gov/citations/19770081915>
- [28] Y. Q. de Aguiar, F. Wrobel, J.-L. Autran y R. G. Alía, *Single-Event Effects, from Space to Accelerator Environments: Analysis, Prediction and Hardening by Design*. Cham, Switzerland: Springer Nature Switzerland AG, 2025, ISBN: 978-3-031-71722-2. DOI: 10.1007/978-3-031-71723-9 dirección: <https://doi.org/10.1007/978-3-031-71723-9>
- [29] R. Velazco, S. Rezgui y R. Ecoffet, «Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection,» inglés, *IEEE Transactions on Nuclear Science*, vol. 47, n.º 6, págs. 2045-2411, 2000.
- [30] A. Aponte-Moreno, J. Isaza-González, A. Serrano-Cases, A. Martínez-Álvarez, S. Cuenca-Asensi y F. Restrepo-Calle, «Evaluation of fault injection tools for reliability estimation of microprocessor-based embedded systems,» *Microprocessors and Microsystems*, vol. 93, pág. 104723, 2022. DOI: 10.1016/j.micpro.2022.104723 dirección: <https://doi.org/10.1016/j.micpro.2022.104723>
- [31] A. Benso y P. Prinetto, eds., *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation* (Frontiers in Electronic Testing), inglés. Boston, MA: Springer US, 2003, vol. 23, ISBN: 978-1-4020-7589-6.
- [32] OpenOCD.org, *OpenOCD Manual (PDF)*, <https://openocd.org/doc/pdf/openocd.pdf>, Accedido: 03-ene-2026.
- [33] pyocd.io, *pyOCD Documentation*, <https://pyocd.io/docs/>, Accedido: 03-ene-2026.
- [34] I. Koren y C. M. Krishna, *Fault-Tolerant Systems*, inglés. Boston, MA: Morgan Kaufmann / Elsevier, 2007, ISBN: 978-0-12-0885251.
- [35] IBM. «¿Qué es un microcontrolador?» Visitado 20 de ago. de 2026. dirección: <https://www.ibm.com/mx-es/think/topics/microcontroller>
- [36] J. W. Valvano, *Embedded Systems: Real-Time Interfacing to ARM[®] Cortex[™]-M Microcontrollers, Volume 2*, Fourth. University of Texas, Austin, TX, USA: Self-published, 2014, ISBN: 978-1463590154. dirección: <http://users.ece.utexas.edu/~valvano/>

- [37] S. P. Gimenez, *8051 Microcontrollers: Fundamental Concepts, Hardware, Software and Applications in Electronics*. Cham, Switzerland: Springer International Publishing AG, 2019, ISBN: 978-3-319-76438-2. DOI: 10.1007/978-3-319-76439-9 dirección: <https://doi.org/10.1007/978-3-319-76439-9>
- [38] J. H. Davies, *MSP430 Microcontroller Basics*. Amsterdam, Netherlands: Newnes, Elsevier, 2008, ISBN: 978-0750682763.
- [39] Arm. «Arm CPU Architecture,» visitado 16 de ago. de 2026. dirección: <https://www.arm.com/architecture/cpu>
- [40] Arm Limited, *ARMv6-M Architecture Reference Manual*, DDI 0419, Consultado en 2026, Arm Limited. dirección: <https://developer.arm.com/documentation/ddi0419/latest/>
- [41] Arm Limited, *ARMv7-M Architecture Reference Manual*, DDI 0403, Consultado en 2026, Arm Limited. dirección: <https://developer.arm.com/documentation/ddi0403/latest/>
- [42] Arm Limited, *Armv8-M Architecture Reference Manual*, DDI 0553, Consultado en 2026, Arm Limited. dirección: <https://developer.arm.com/documentation/ddi0553/latest/>
- [43] Arm Limited, *Arm Debug Interface Architecture Specification ADIv5.0 to ADIv5.2*, IHI 0031, Consultado en 2026, Arm Limited. dirección: <https://developer.arm.com/documentation/ihi0031/latest/>
- [44] Arm Limited. «Understanding CoreSight.» Document ID 102551_0100_01_en. Consultado en 2026. dirección: <https://support.arm.com/documentation/102551/0100/Understanding-CoreSight>
- [45] ARM Ltd., *CMSIS-DAP: Debug Access Port Implementation*, https://arm-software.github.io/CMSIS_5/DAP/html/index.html, Accessed: Mar. 2026.
- [46] Arm Limited, *ARMv7-M Architecture Reference Manual*, DDI 0403, Consultado en 2026, Arm Limited. dirección: <https://developer.arm.com/documentation/ddi0403/latest/>
- [47] Tool Interface Standards Committee, *Executable and Linking Format (ELF) Specification*, Tool Interface Standard, Linux Foundation.
- [48] Arm Limited, *CMSIS System View Description*, Arm Limited.
- [49] J. Aidemark, J. Vinter, P. Folkesson y J. Karlsson, «GOOFI: Generic Object-Oriented Fault Injection Tool,» inglés, *International Conference on Dependable Systems and Networks (DSN)*, 2001.
- [50] D. Skarin, R. Barbosa y J. Karlsson, «GOOFI-2: A Tool for Experimental Dependability Assessment,» inglés, *International Conference on Dependable Systems and Networks (DSN)*, 2010.
- [51] N. Stollon, «Nexus IEEE 5001,» inglés, en *Debugging Systems-on-Chip*, Springer, 2011. DOI: 10.1007/978-1-4419-7563-8_11 visitado 2 de jun. de 2025. dirección: https://www.researchgate.net/publication/251124321_Nexus_IEEE_5001
- [52] A. Aponte, F. Restrepo y C. Pedraza, «MiFIT: A Fault Injection Tool to Validate the Reliability of Microprocessors,» inglés, *Latin American Test Symposium (LATS)*, 2019.
- [53] X. Mallón. «¿Qué es Bash (Shell) y cómo funciona?» Visitado 2 de jun. de 2025. dirección: <https://keepcoding.io/blog/que-es-bash-shell-y-como-funciona/>
- [54] A. Ghani, A. Nasser y A. Jacob, «FERRARI: A Flexible Software-Based Fault and Error Injection System,» inglés, *IEEE Transactions on Computers*, vol. 44, págs. 248-260, 1995.
- [55] Dirección General de Cómputo y de Tecnologías de Información y Comunicación (DGTIC), UNAM. «Sistema operativo familia Unix,» visitado 2 de jun. de 2025. dirección: https://cursosenlinea.tic.unam.mx/cursos/Sistema_operativo_tipo_Unix.htm
- [56] D. Stott, B. Floering, Z. Kalbarczpk y R. Lyer, «NFTAPE: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors,» inglés, *Proceedings IEEE International Computer Performance and Dependability Symposium (IPDS)*, 2000.

- [57] V. Sih, O. Tschache y F. Balbach, «VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions,» inglés, *Proceedings of IEEE 27th International Symposium on Fault-Tolerant Computing*, 1997.
- [58] E. Jenn, J. Arlat, M. Rimen, J. Chisson y J. Karlsson, «Fault Injection into VHDL Models: The MEFISTO Tool,» inglés, *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, 1994.
- [59] G. Choi y R. Iyer, «FOCUS: An Experimental Environment for Fault Sensitivity Analysis,» inglés, *IEEE Transactions on Computers*, vol. 41, págs. 1515-1526, 1992.
- [60] A. Fidalgo, G. Alves y J. Ferreira, «A Modified Debugging Infrastructure to Assist Real Time Fault Injection Campaigns,» inglés, *Design and Diagnostics of Electronic Circuits and Systems*, 2006.
- [61] J. Isaza, A. Serrano, F. Restrepo, S. Cuenca y A. Martínez, «Dependability Evaluation of COTS Microprocessors via On-Chip Debugging Facilities,» inglés, *2016 17th Latin-American Test Symposium (LATS)*, 2016.
- [62] IONOS México, *¿Qué es QEMU? La herramienta de emulación de un vistazo*, <https://www.ionos.mx/digitalguide/servidores/configuracion/qemu/>, Consultado el 13 ene. 2026, 2025.
- [63] M. Portela, C. Lopez, M. Garcia y L. Entrena, «A Rapid Fault Injection Approach for Measuring SEU Sensitivity in Complex Processors,» inglés, *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, 2007.
- [64] XJTAG, *¿Qué es JTAG y cómo puedo usarlo? Una guía de XJTAG*, https://www.xjtag.com/wp-content/uploads/xjtag-ebook_what-is-jtag-es.pdf, Consultado el 13 ene. 2026, 2025.
- [65] I. Sommerville, *Ingeniería de software*, 9.^a ed., trad. por V. Campos Olguín. Pearson Educación de México / Addison-Wesley, 2011, Traducción autorizada de *Software Engineering*, 9th ed., ISBN: 978-607-32-0603-7.
- [66] C. Noeldeke et al., «Single Event Upset investigations on the “Flying Laptop” satellite mission,» *Advances in Space Research*, vol. 67, n.º 6, págs. 2000-2009, 2021. DOI: 10.1016/j.asr.2020.12.032
- [67] Agencia Espacial Mexicana, *Nuevo sistema satelital mexicano SCMI al espacio en 2025 con GuaraniSat-2*, <https://www.gob.mx/aem/articulos/nuevo-sistema-satelital-mexicano-scmi-al-espacio-en-2025-con-guaranisat-2-384061>, Consultado el 2 de agosto de 2026, 2024.
- [68] STMicroelectronics, *STM32F4DISCOVERY Discovery kit product image*, <https://www.st.com/en/evaluation-tools/stm32f4discovery.html>, Accessed: Mar. 2026.
- [69] Free Software Foundation, *GNU objdump Manual*, GNU Project, 2024. dirección: <https://sourceware.org/binutils/docs/binutils/objdump.html>
- [70] A. V. Aho, M. S. Lam, R. Sethi y J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Pearson Addison-Wesley, 2006.