MAESTRÍA EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Evaluador de microcontroladores para misiones espaciales

Autor:
Esp. Ing. Gonzalo Nahuel Vaca

Director:
Ing. Roberto Cibils (INVAP S.E.)

Codirector:
Ing. Damian Rosetani (INVAP S.E.)

Jurados:
Mg. Ing. Iván Andrés León Vásquez (INVAP S.E.)
Mg. Ing. Rodrigo Cardenas (FIUBA)
Esp. Ing. Pablo Almada (FIUBA-UTN)

Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires, entre mayo de 2021 y junio de 2022.

Resumen

Esta memoria explica el trabajo realizado para INVAP S.E. en el área de la tecnología aeroespacial. Se realizó una herramienta que simula los efectos de la radiación cósmica en un microcontrolador.

La herramienta sirve para abaratar el costo de la producción de satélites y aumentar su confiabilidad. Las simulaciones permiten evaluar técnicas de mitigación de errores y componentes no calificados para uso espacial. Para realizar este trabajo se valió de la teoría de arquitecturas de microcontroladores y sus protocolos de depuración.

Índice general

Índice de figuras

Índice de tablas

Dedicado a mi hija Helena

Capítulo 1

Introducción general

Este capítulo presenta de forma breve el contexto del trabajo realizado. Se explica la problemática solucionada y los conceptos necesarios para la lectura de la memoria.

1.1. El espacio como recurso estratégico

El espacio exterior es un recurso estratégico para el bienestar de la humanidad. Dado que, su explotación permite mejorar la producción de bienes y servicios. Además, provee datos de valor científico que incrementan la calidad de vida de las personas [1].

El marco jurídico que regula la explotación del recurso es el Tratado sobre el espacio exterior de 1967. Del cual, la República Argentina es miembro desde el 27 de enero de ese año. Los países pueden reclamar cualquiera de las órbitas disponibles pero tienen la obligación de hacer uso dentro de un margen de tiempo determinado. Luego, si el país no ocupó la órbita pierde el derecho a usarla [2].

El mercado de la explotación espacial está compuesto de estados y empresas. Siendo las últimas quienes adquirieron el protagonismo en el siglo XXI [3]. El marco jurídico y la competencia en este rubro empuja a las compañías a innovar de forma permanente. En particular, con nuevas técnicas y tecnologías que permitan abaratar el costo de las misiones.

El cliente de este trabajo es INVAP S.E. y es una empresa de la provincia de Río Negro. La compañía es modelo en su tipo y realiza proyectos tecnológicos complejos en áreas como: reactores nucleares, satélites y radares. En la figura 1.1 se puede observar un satélite realizado por la empresa.

FIGURA 1.1. Satélite SAOCOM[1].

1.2. Radiación cósmica y sus efectos

El sol produce partículas de luz e iones pesados que de forma conjunta se denominan viento solar. Este fenómeno es atenuado antes de llegar a la superficie del planeta gracias al campo magnético terrestre [5]. Como se puede ver en la figura 1.2, las partículas son desviadas por el campo. Luego, este queda deformado por el viento solar y se genera una magnetosfera asimétrica. En la tabla 1.1 se puede observar las características de la asimétria.

TABLA 1.1. Cinturón de Van Allen [5].

Cinturon	Frontera	Partícula dominante
Interior	1,2 - 2,5 radios terrestres	Protones de alta energía
Exterior	2,8 - 12 radios terrestres	Electrones de alta energía

La electrónica de los satélites tiene un alto grado de exposición al viento solar. Esto significa que la probabilidad de incidencia de una partícula cargada en el circuito es mayor. La incidencia de una partícula genera una traza densa de pares electrón-hueco en los semiconductores [6]. Además, es posible que esta ionización cause un pulso transitorio de corriente. En la figura 1.3 se puede ver el efecto de la radiación sobre los transistores de un integrado.

Los efectos de la radiación cósmica sobre el circuito pueden ser transitorios o permanentes. Los permanentes se deben a la destrucción de una parte del circuito. Esta destrucción es producto de: el disparo de componentes activos parásitos o la generación de plasma dentro del encapsulado [7]. Finalmente, en la tabla 1.2 se puede ver un resumen de los tipos de errores generados por la radiación cósmica.

[1]Imagen tomada de la página oficial de INVAP S.E. [4]

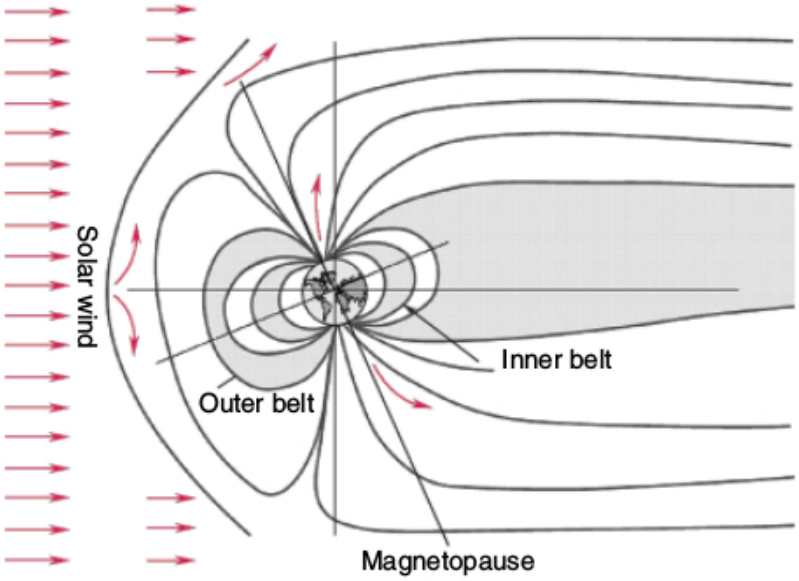


FIGURA 1.2. Capas magnéticas de la tierra y viento solar[2].

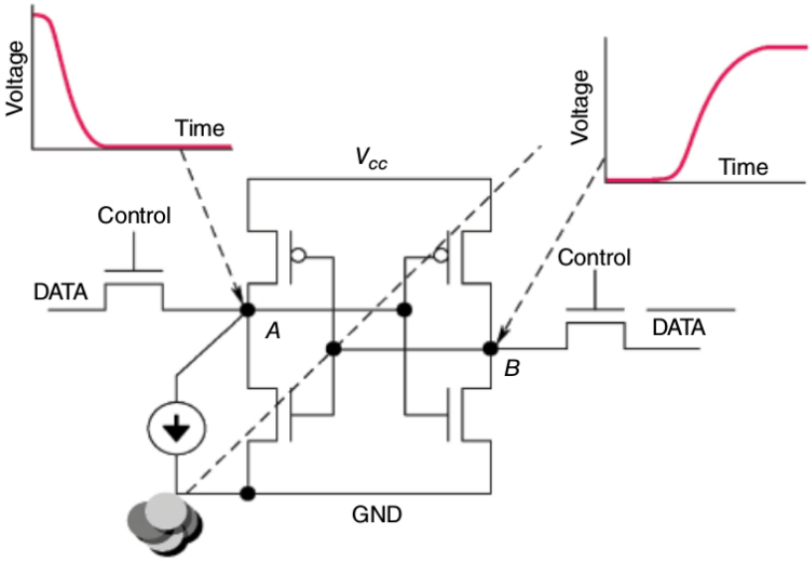


FIGURA 1.3. Ejemplo simplificado de *bit flip* en un bloque *SDRAM*[3].

[2]Imagen tomada del artículo *Structure of space radiation* [5]

[3]Imagen tomada del artículo *Effects of space radiation on electronic devices* [7]

TABLA 1.2. Efectos producidos por la radiación cósmica [5].

Evento	Acrónimo	Efecto
Latch-up	SEL	Pico de corriente
Upset	SEU	Alteración de datos
Funtional Interrupt	SEFI	Cambios en la configuración
Transient	SET	Pico de tensión
Burnout	SEB	Activación de transistores parásitos
Gate Rapture	SEGR	Generación de plasma de alta densidad

1.3. Calificación espacial e iniciativa *new space*

A los efectos vistos en la sección 1.2 se suman: el estrés mecánico del lanzamiento y los cambios de temperatura en la órbita. Este ambiente genera la necesidad de utilizar componentes con calificación espacial. Para que un componente alcance la calificación espacial se debe someter a un largo y costoso proceso de acreditación. Luego, estos componentes adolecen de un elevado precio y atraso tecnológico frente a los del mercado masivo [8].

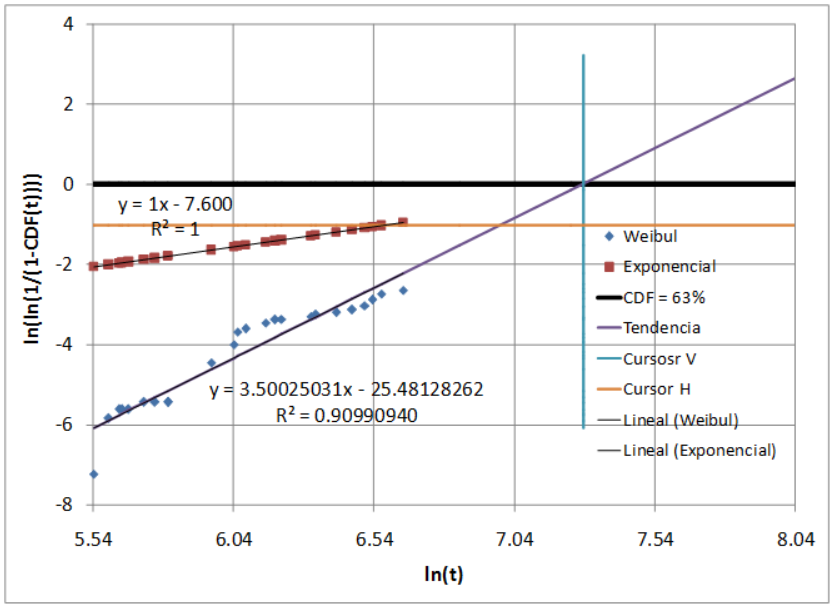
La irrupción del sector privado vista en la sección 1.1, trajo una nueva iniciativa comercial denominada *new space*. Esta iniciativa busca bajar los costos al utilizar componentes no calificados para su uso espacial. Además, existe la ventaja adicional de introducir tecnología de vanguardia.

El caso de *Starlink* es un ejemplo de *new space* particular. Su volumen de satélites lanzados permite realizar conclusiones estadísticas significativas. En particular, su poca capacidad para cumplir sus objetivos si se mantiene la actual tasa de mortalidad de sus satélites [9]. En las figura 1.4 se puede observar que la constelación no logrará alcanzar la población deseada.

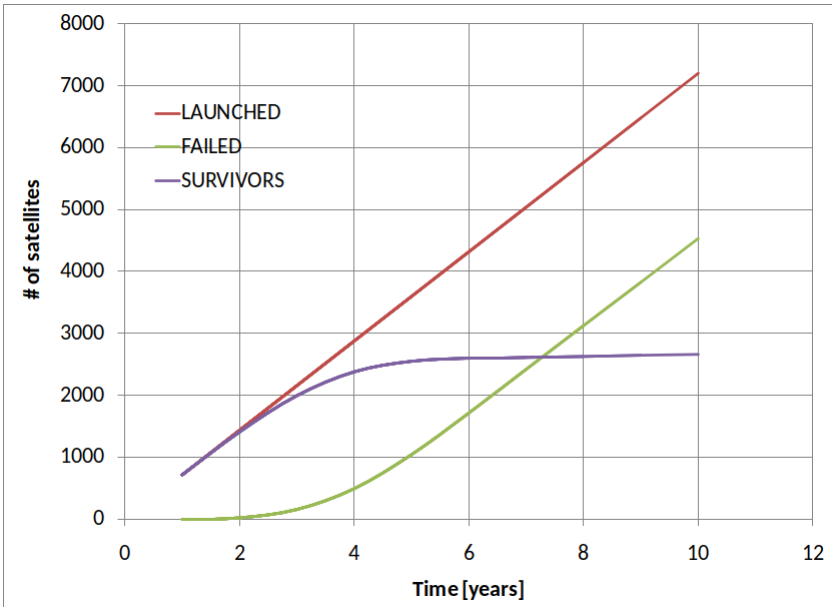
Al problema de población de *Starlink* se suma la gran cantidad de polución generada. Los satélites fuera de servicio no pueden ser desorbitados y persisten en forma de *debris*. Como se puede ver en la tabla 1.3, el volumen de basura generado es significativo.

TABLA 1.3. Proyección de *debris* de *Starlink* [9].

Lanzamientos	Satélites	Total lanzados	Población	Debris
12	60	7200	2704	4046
12	180	21600	8105	12146
12	400	48000	18007	26994
180	60	108000	40000	61200
60	180	108000	40000	61200
27	400	108000	40000	61200



(A) Gráfico Weibull de expectativa de vida *Starlink*.



(B) Proyección de la constelación *Starlink*.

FIGURA 1.4. Estadísticas de la constelación *Starlink*[4].

Las conclusiones del caso *Starlink* muestran la importancia de tener herramientas para simular el ambiente espacial. En particular, los efectos de la radiación cósmica para poder probar las técnicas de mitigación de errores seleccionadas. Finalmente, este trabajo agrega valor al cliente al incrementar la confiabilidad de los satélites y evitar los problemas de la competencia.

[4]Imagen tomada de la publicación de Roberto Cibils [9].

1.4. Estado del arte

Los ensayos de pruebas de radiación en tierra tienen las siguientes estrategias [6]:

- Ensayo por *software*: se introducen instrucciones espúreas en el código para generar errores.
- Ensayo por *hardware*: se conecta un dispositivo que introduce errores durante la ejecución del programa.
- Ensayo por radiación: se introduce el dispositivo bajo prueba en una cámara de iones pesados como se puede ver en la figura 1.5.

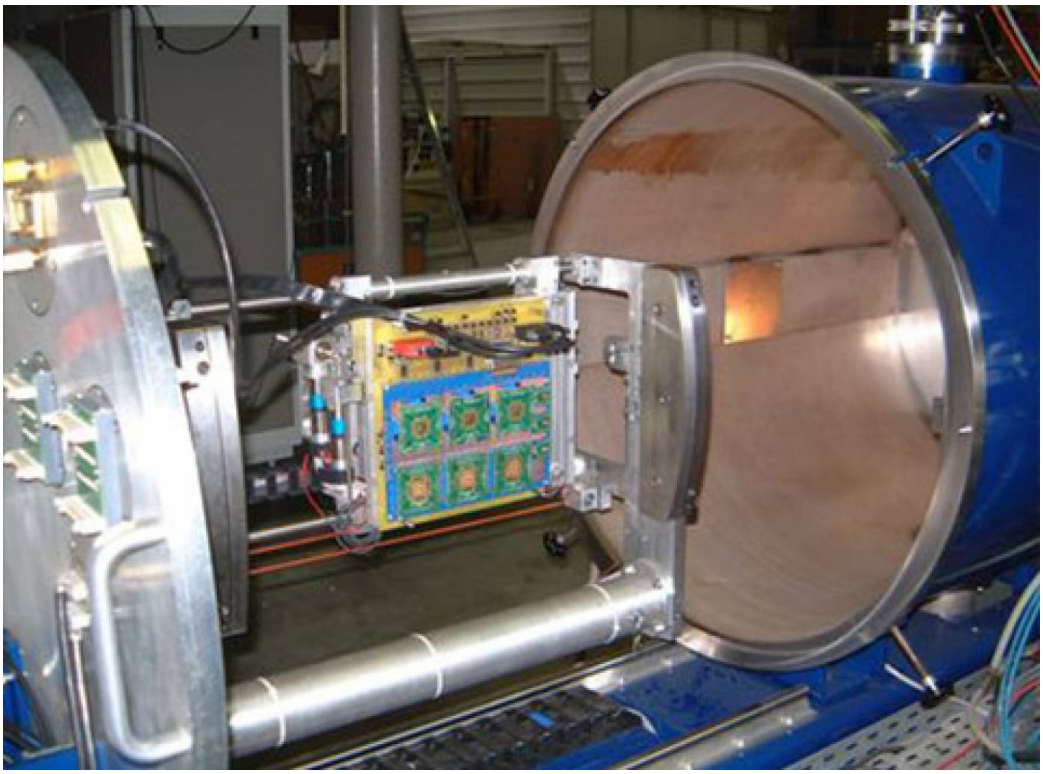


FIGURA 1.5. Cámara de pruebas de iones pesados[5].

Los métodos de ensayos mencionados presentan compromisos de diseño como se puede ver en la tabla 1.4. El trabajo realizado es una solución del tipo ensayo por *hardware*.

TABLA 1.4. Comparación de métodos de simulación [6].

Método	**Eficiencia**	**Costo**	**Limitación**
Software	Baja	Bajo	Ciclos de CPU
Hardware	Media	Medio	Acceso al integrado
Radiación	Alta	Alto	Control del ensayo

[5]Imagen tomada del sitio web ucl.ac.eu [10].

1.5. Alcance del trabajo

El trabajo realizado se divide en dos partes:

1. *Firmware* para el dispositivo bajo prueba.
2. Inyector de *soft-errors* por consola de comandos.

El *firmware* en el dispositivo bajo prueba tiene la misión de validar su funcionamiento. Esto se logró al verificar cada periférico de interés dentro del integrado. Luego, se generan reportes periódicos que se envían al inyector por consola de comandos. En la figura 1.6 se puede observar un diagrama en bloques simplificado.

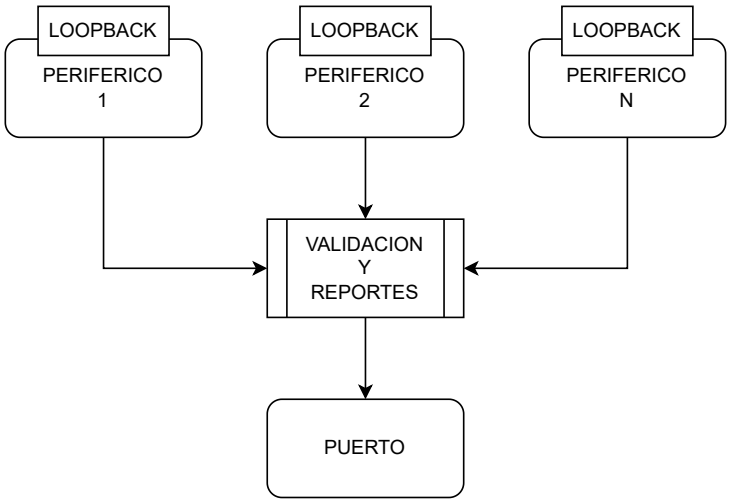


FIGURA 1.6. Diagrama simplificado del dispositivo bajo prueba.

El inyector por consola de comandos tiene la función de planificar los ensayos y gestionar la introducción de errores. En la figura 1.7 se puede ver como interactúan las partes del sistema.

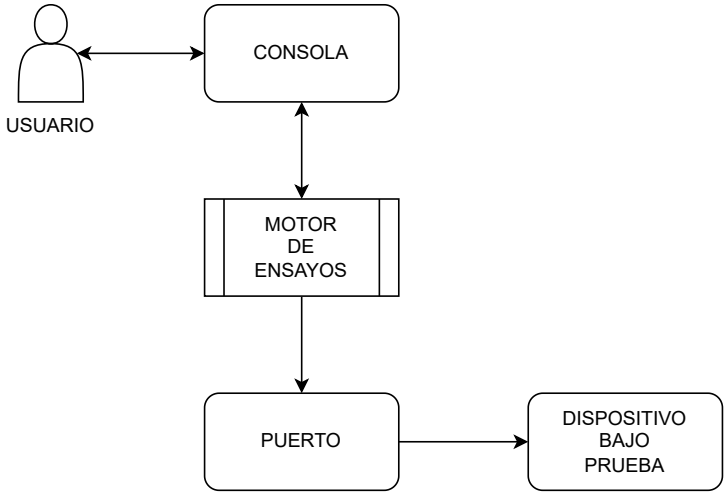


FIGURA 1.7. Diagrama simplificado del sistema de inyección de errores.

Capítulo 2

Introducción específica

En este capítulo se detallan las tecnologías que forman parte del trabajo. Son productos de terceros que se integran en las herramientas entregadas al cliente.

2.1. Arquitectura del dispositivo bajo prueba

El trabajo fue realizado para un tipo de microcontrolador específico. Su diseño forma parte de la familia *Cortex M* de la empresa *ARM*. En la figura 2.1 se puede observar un diagrama en bloques de la arquitectura.

Al fabricante del dispositivo bajo prueba se le impone respetar el mapa de memoria y registros del núcleo. Esto permitió construir un inyector de *soft-errors* genérico. Finalmente, la herramienta entregada funciona para cualquier integrado de la familia *Cortex M*.

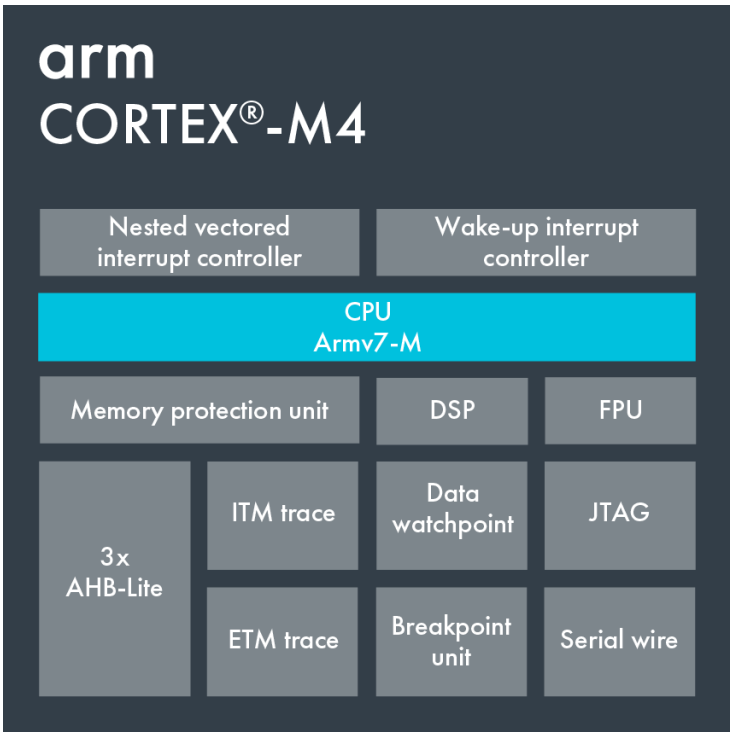


FIGURA 2.1. Diagrama de la arquitectura *Cortex M4*[1].

[1]Imagen tomada de la página oficial de *ARM Developers*. [11]

La arquitectura tiene un módulo que permite programar y depurar el integrado. Este módulo se denomina *CoreSight* y es propio de los dispositivos *ARM*. En la figura 2.2 se muestra un diagrama en bloques del módulo. Sus partes principales son:

- *Cross Triggering*: permite conectar y encaminar las señales que utilizan las sondas de depuración.
- *Debug Access Port*: es el puerto físico para conectar la sonda de depuración. Es una implementación de la interfaz de depuración *ARM*.
- *Embedded Trace Macrocells*: permite extraer información y controlar el núcleo del dispositivo.
- *Instrumentation Trace Units*: permite que una sonda de depuración se conecte con las *Embedded Trace Macrocells*.
- *ROM Tables*: sirven para que la sonda de depuración identifique al integrado.
- *Self Hosted Debug*: son instrucciones específicas de depuración controladas por un procesador secundario.
- *Trace Interconnect*: provee puentes para compartir señales de reloj, alimentación y otras señales comunes.

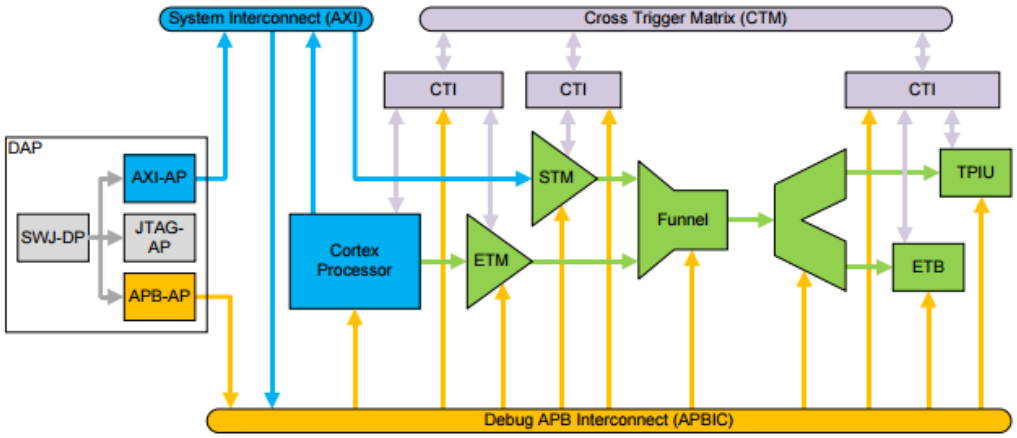


FIGURA 2.2. Diagrama del módulo *CoreSight*[2].

2.2. Servidores y sondas de depuración

Una sesión de depuración sirve para observar y modificar el estado de ejecución de un programa. Esto se logra al leer y modificar los valores en registros del procesador y periféricos. Además, se necesita de un sistema de disparos por eventos y supervisión de recursos. Finalmente, la sesión debe detener la ejecución del núcleo de ser necesario. En la figura 2.3 se puede observar un esquema simplificado de una sesión de depuración.

[2]Imagen tomada del artículo *How to debug: CoreSight basis*. [12]

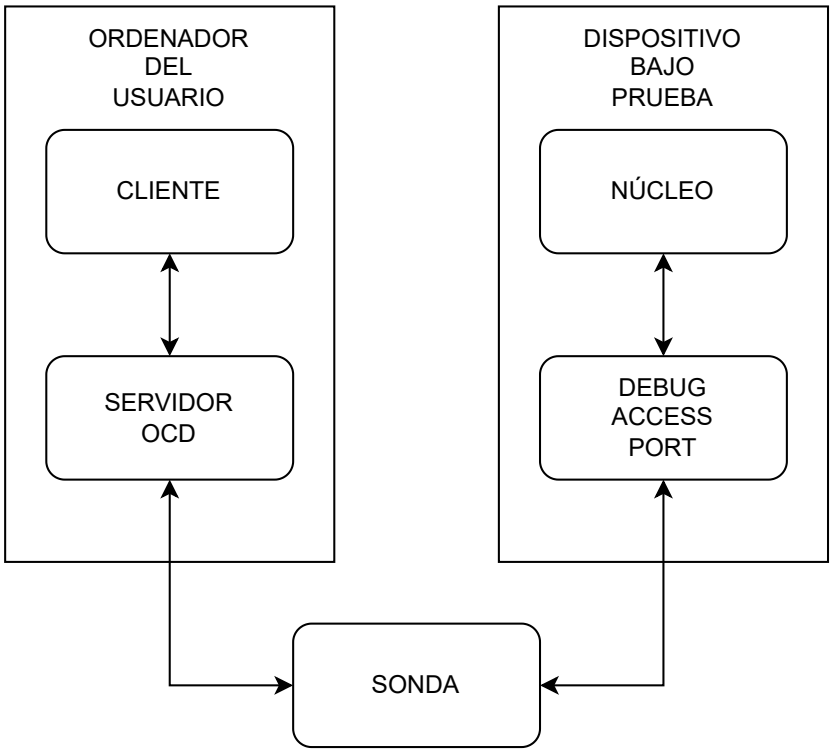


FIGURA 2.3. Conexión de una sesión de depuración.

Las sondas de depuración tienen el objetivo de conectar el *Debug Access Port* con el puerto del ordenador del usuario. Adaptan los niveles de tensión y los protocolos involucrados. Luego, permiten realizar una sesión de depuración, programar el dispositivo o verificar el estado de los componentes en la placa. En la figura 2.4 se puede ver la sonda provista por el cliente.

FIGURA 2.4. Sonda de depuración *Segger J-32*[3].

Un servidor *On-chip debugger* tiene la misión de abstraer la conexión la sonda de depuración. Además, facilita el manejo del ciclo de vida de la sesión y permite usar *software* como *GNU Project debugger*. Finalmente, es la base de una pila de tecnologías que permite el uso de herramientas como *Eclipse IDE*. En la tabla 2.1 se puede observar un resumen de los servidores evaluados en el trabajo.

[3]Imagen tomada de https://www.digikey.com/

TABLA 2.1. Comparativa entre servidores de depuración

Servidor	API	Acceso	Licencia
OpenOCD	tcl	Registros y SDRAM	MIT
PyOCD	Python 3	Registros y SDRAM	Apache-2.0

2.3. Periféricos de interés

El dispositivo bajo prueba ofrece una variedad de periféricos para el desarrollo de aplicaciones. Sin embargo, el cliente manifestó interés solo en los que se nombran a continuación:

- CAN: este periférico permite al microcontrolador ser el dispositivo principal en una *Controller Area Network*. La red es de grado industrial y fue diseñada para gestionar una red de sensores en un ambiente automotriz.
- PIO: es el puerto de entradas y salidas digitales de propósito general. En el caso del dispositivo bajo prueba, el periférico permite usar circuitos anti rebote, *pull-up* y *pull-down* internos.
- SPI: el periférico permite realizar una conexión del tipo *Serial Peripheral Interface*. Esta conexión es sincrónica y solo apta para distancias cortas.
- UART: es un periférico que permite conectarse a puertos y controlar dispositivos serie.
- Watchdog: el periférico sirve para detectar un error de ejecución y reiniciar el microprocesador.

En la tabla 2.2 se resume la funcionalidad de cada uno de ellos.

TABLA 2.2. Resumen de periféricos

Periférico	Funcionalidad
CAN	Bus de comunicación de grado industrial
PIO	Entradas y salidas digitales
SPI	Interfaz de comunicación sincrónica
UART	Puerto para dispositivos serie
Watchdog	Detección de errores y reinicio del integrado

2.4. Requerimientos del cliente

Se realizaron una serie de reuniones con el cliente y se pudo definir los requerimientos del trabajo. A continuación se enumeran los principales:

1. Referentes al inyector por consola de comandos:
   a) Generará de una interfaz de usuario.
   b) Permitirá configurar el ensayo a realizar.
   c) Observará la salida del dispositivo bajo prueba.
   d) Inyectará *soft-errors* en el dispositivo bajo prueba.
   e) Persistirá las operaciones, entradas y salidas.
   f) Generará informes del ensayo realizado.
2. Referentes al proceso del dispositivo bajo prueba:
   a) Verificará el estado de los periféricos del dispositivo bajo prueba.
   b) Detectará si el dispositivo bajo prueba perdió su secuencia.
   c) Generará reportes de estado de periféricos y secuencia.
   d) Permitirá que inyector por consola de comandos configure el alcance de la secuencia.
   e) Permitirá que inyector por consola de comandos maneje el flujo de su secuencia.

El cliente definió algunas restricciones para el desarrollo del sistema. Estas se enumeran a continuación:

- Utilización de un repositorio con control de versiones *Gitlab*.
- Documentación del código con *Doxygen*.
- Utilización exclusiva del lenguaje de programación *Python 3*.

Capítulo 3

Diseño e implementación

Este capítulo detalla la generación de contenido original del trabajo. Se explica su diseño y producción.

3.1. Autoevaluación del dispositivo bajo prueba

La construcción del *firmware* de autoevaluación del dispositivo bajo prueba requirió superar las siguientes etapas:

- Configuración de las señales de reloj del dispositivo bajo prueba.
- Selección y configuración de los periféricos del dispositivo bajo prueba.
- Selección y configuración de los terminales externos del dispositivo bajo prueba.
- Implementación de las estrategias de validación de periféricos.
- Integración de una secuencia de validación y reporte.

Para configurar las frecuencias de reloj se buscó obtener 150 MHz para suministrar al *Master CAN Bus*. Con esta condición satisfecha, se pudo configurar las frecuencias de reloj del resto de los periféricos. En la figura 3.1 se puede observar la utilización del *Programmable Clock Controller* número cinco.

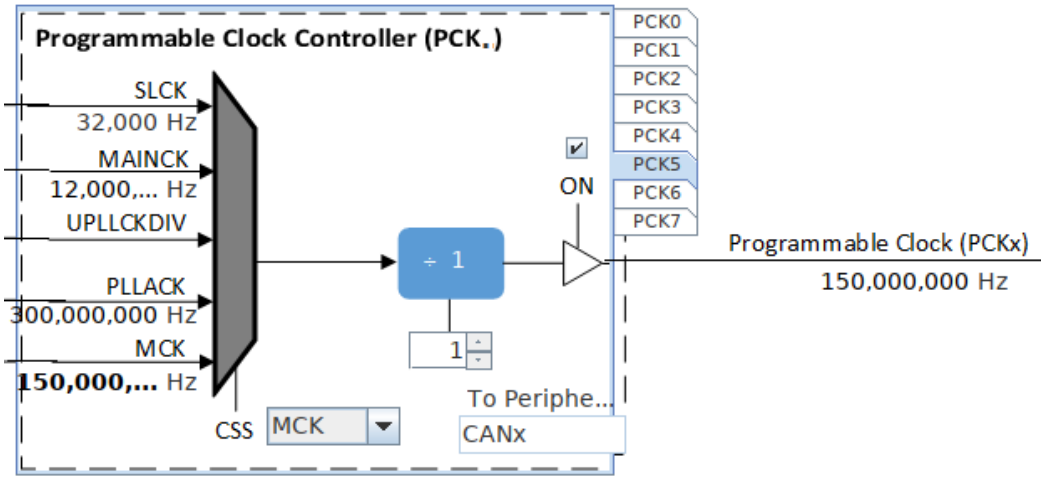


FIGURA 3.1. Diagrama de configuración de las señales de reloj.

El siguiente paso en la etapa de diseño fue la selección de las instancias de los periféricos del integrado. Es posible que dos periféricos compartan parte del circuito

interno o terminales del encapsulado. Esta situación puede generar una disminución en las funcionalidades o una total incompatibilidad. Finalmente, se seleccionaron instancias completamente disjuntas.

Luego de seleccionar las instancias de los periféricos, se configuraron para realizar un *loopback*. La configuración se realizó de la siguiente manera:

- *CAN*: se utilizó el *MCAN1* con una configuración de *loopback* interna, como se puede ver en la figura 3.2.
- *PIO*: se configuraron dos terminales del dispositivo bajo prueba. El primero como salida sin *latch* y el segundo como entrada sin circuito anti rebote.
- *SPI*: la configuración elegida fue por defecto ya que el *loopback* se logró conectando *TX* y *RX* con un cable.
- *UART*: se configuró el periférico con una velocidad de 9600 baudios, 8 bits de datos y sin bits de paridad.
- *Watchdog*: el disparo se configuró con un contador a 4095 cuentas. Este valor se estimó entre dos y cinco ejecuciones del *loop* principal.

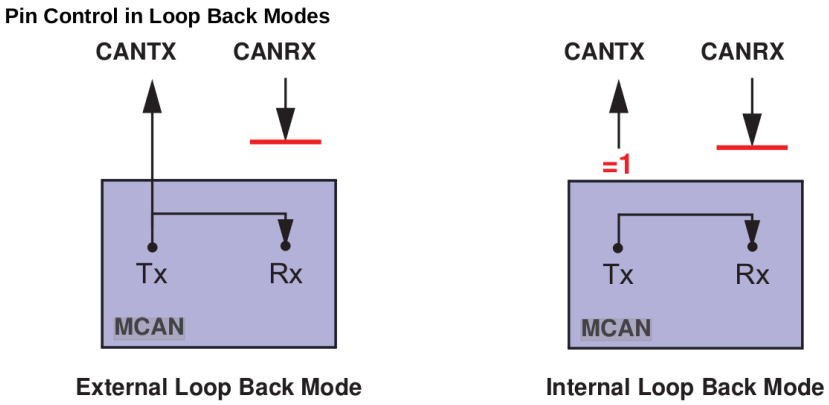


FIGURA 3.2. Diagrama de *loopback* del periférico *CAN*[1].

Como se puede ver en la figura 3.3, se priorizaron los *loopbacks* físicos externos. Cuando esta estrategia no fue posible, se optó por internos provistos por el fabricante. Finalmente, en los casos que las dos primeras opciones fuesen imposibles, se utilizó una estrategia de *software*. En la tabla 3.1 se puede ver un resumen de las estrategias aplicadas.

TABLA 3.1. Comparación entre estrategias de depuración

Periférico	Validación	Detección en un ciclo
CAN	Loopback interno	Si
PIO	Loopback externo	No
SPI	Loopback externo	Si
UART	Lógica en firmware	No
Watchdog	Lógica en inyector	No

Una vez configurados los componentes de *hardware* del dispositivo bajo prueba; se procedió a diseñar el *firmware*. Se comenzó con la estructura que define los

[1]Imagen tomada de la hoja de datos del dispositivo bajo prueba [13].

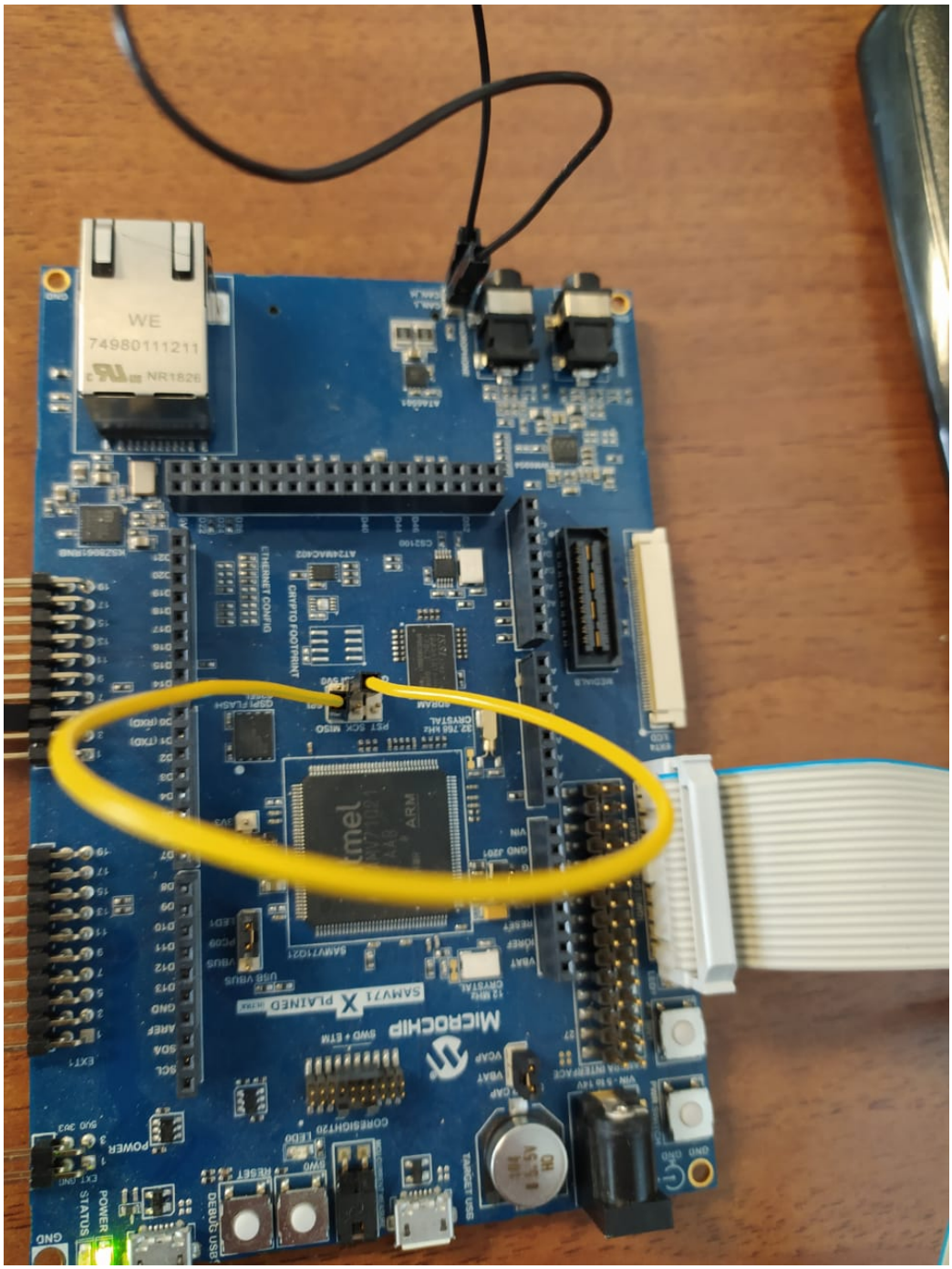


FIGURA 3.3. Fotografía del dispositivo bajo prueba.

reportes de estado del dispositivo bajo prueba. Los reportes están formados por 2 bytes, el primero es el carácter "F" y marca el inicio del reporte mientras que el segundo byte lleva la información del estado de los periféricos. En el código 3.1 se puede ver la implementación del segundo byte del reporte.

```
1  #define BIT 1
2
3  struct status_bitfield_t
4  {
5      uint8_t CAN:BIT;
6      uint8_t SPI:BIT;
7      uint8_t PIO:BIT;
8      uint8_t WATCHDOG:BIT;
9  }__attribute__((packed));
10
11 typedef union
12 {
13     struct status_bitfield_t status_of;
14     uint8_t packed;
15 }report_t;
```

CÓDIGO 3.1. Definición de la estructura de reportes.

En el código 3.2 se puede observar la implementación del lazo principal. Es importante notar que en la línea 10 se utilizó la *union* para transformar el reporte en caracteres legibles para una persona. En la figura 3.4 se puede observar el flujo completo del programa.

```
1  while ( true )
2  {
3      SYS_Tasks ( );
4      report.status_of.CAN = validate_CAN();
5      report.status_of.PIO = validate_PIO();
6      report.status_of.SPI = validate_SPI();
7      report.status_of.WATCHDOG = NORMAL;
8
9      buffer[FRAME_START] = 'F';
10     buffer[FLAGS_INDEX] = report.packed + 'A';
11     USART1_Write(&buffer[0], FRAME_SIZE);
12
13     WDT_Clear();
14 }
```

CÓDIGO 3.2. Lazo principal del *firmware* de autoevaluación.

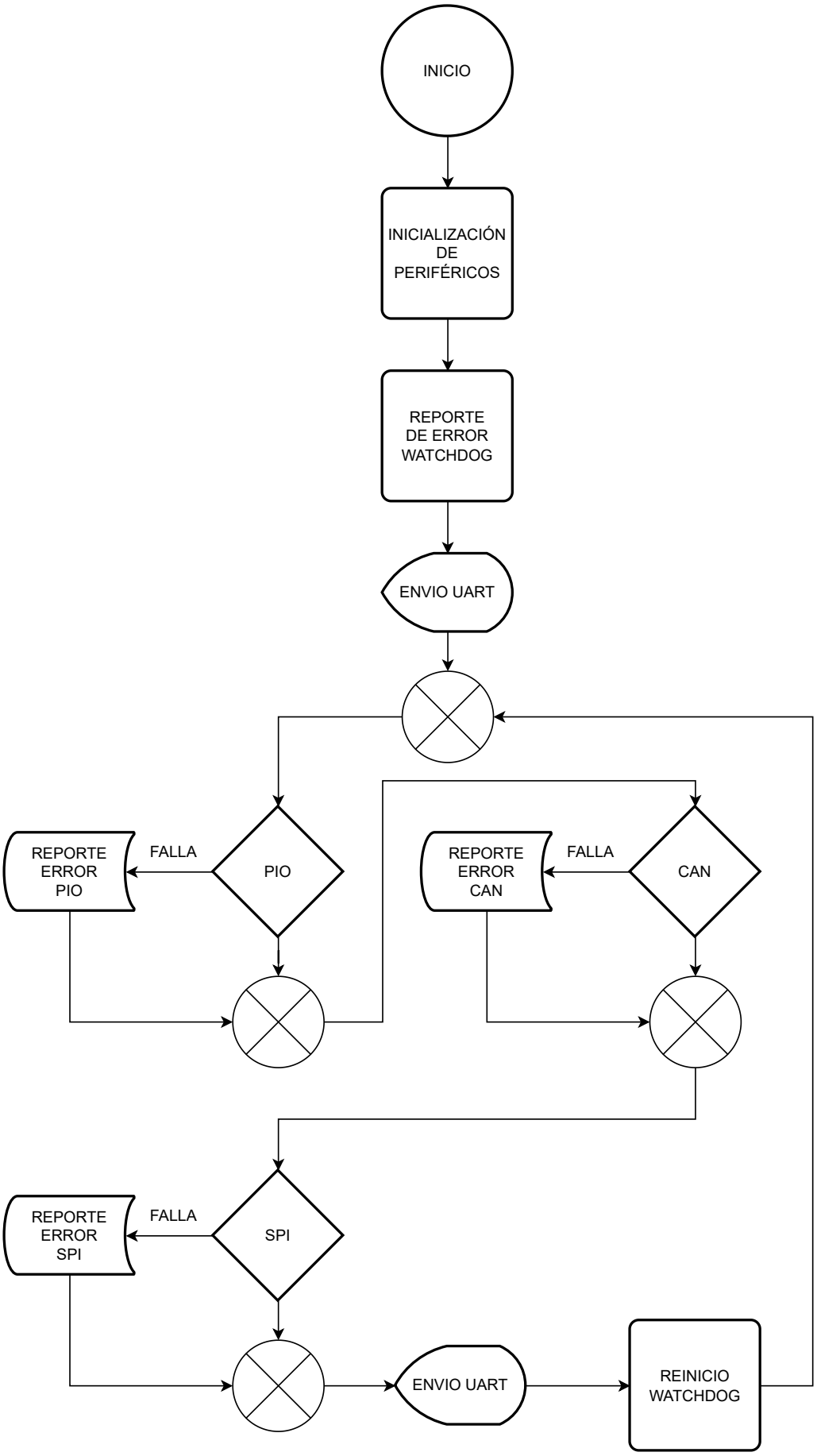


FIGURA 3.4. Flujo del *firmware* de autoevaluación.

3.2. Interfaz de programación de aplicaciones

La interfaz de programación de aplicaciones tiene la función de abstraer al inyector de errores del servidor *OCD*. Esto se logró usando los siguientes patrones de diseño:

- Programación orientada a objetos *(OOP)*: este patrón de diseño se basa en agrupar en una unidad lógica las funcionalidades y estados que tengan un alto grado de acoplamiento. Esto significa que la funciones que tienen efectos colaterales junto a los datos mutados se encapsulan dentro de una construcción denominada objeto. El principal objetivo de un objeto es contener dentro suyo los efectos colaterales. Finalmente, el lenguaje de programación *Python 3* permite, a través de una *class*, modelar un tipo de dato que permite instanciar a un objeto.
- *Resource Acquisition Is Initialization (RAII)*: este patrón de diseño consiste en modelar el ciclo de vida de un recurso al usar un objeto. Un recurso es todo aquello que requiera mantenimiento luego de su uso, como por ejemplo: liberar memoria, cerrar una conexión o unir dos hilos de programa. Esto se logra adquiriendo un recurso cuando se invoca el constructor de la *class* por ejemplo, la conexión con la sonda de depuración se realiza durante la instanciación de un objeto llamado conexión. Luego, cuando se desea cerrar la conexión se invoca al destructor del objeto. Dentro de esta función se encuentra el código para cerrar de forma ordenada la conexión con la sonda y el dispositivo bajo prueba. Finalmente, este patrón de diseño hace que el programa maneje de forma robusta los recursos ya que está garantizada la ejecución de los destructores.

En el código 3.3 se puede observar un ejemplo de *RAII* donde se maneja como recurso la detención del núcleo del dispositivo bajo prueba. Esto permite que frente una excepción del proceso que esté utilizando la interfaz de programación de aplicaciones, el núcleo pueda continuar operando. Finalmente, se posibilita recuperar el proceso sin tener que reiniciar el dispositivo bajo prueba.

```
class Halted():
    def __init__(self, target):
        self.target = target

    def __enter__(self):
        self.target.halt()

    def __exit__(self, exc_type, exc_val, traceback):
        self.target.resume()
```

CÓDIGO 3.3. Ejemplo de *Resource Acquisition Is Initialization (RAII)*.

Los patrones de diseño utilizados permiten escribir funciones expresivas y robustas. Como se puede ver en el código 3.4, es fácil comprender lo que sucede. Se detiene el núcleo para leer una posición de memoria, luego de la lectura, el núcleo continua en funcionamiento. Finalmente, se retorna el valor leído.

```
def readMemory(self, addr: int):
    with Halted(self.target):
        val = self.target.read_memory(addr)
    return val
```

CÓDIGO 3.4. Ejemplo de uso de *RAII*.

En la tabla 4.1 se puede observar un resumen de las funcionalidades y sus estrategias de abstracción.

TABLA 3.2. Funcionalidades abstraídas

Funcionalidad	**Patrón de diseño**	**Acceso**
Conexión al integrado	RAII	Público
Detener el núcleo	RAII	Privado
Registros CORE: read/write	OOP	Público
Memoria SDRAM: read/write	OOP	Público

Finalmente, se logró abstraer el ciclo de vida de la sesión de depuración como se muestra en la figura 3.5.

3.3. Sistema de inyección de emphsoft-errors

Una vez lograda la interfaz de programación de aplicaciones explicada en la sección 3.2, se pudo construir el inyector por consola de comandos.

3.4. Biblioteca para el desarrollo de ensayos

```
import sise.library as sise

dut = sise.Connection()

# Bit-flip en SDRAM
addr = 0x20400000
bit = 0
res = dut.bitFlipMemory(addr, bit)
print("res:", res)

del(dut)
```

CÓDIGO 3.5. Ejemplo de aplicación de la biblioteca.

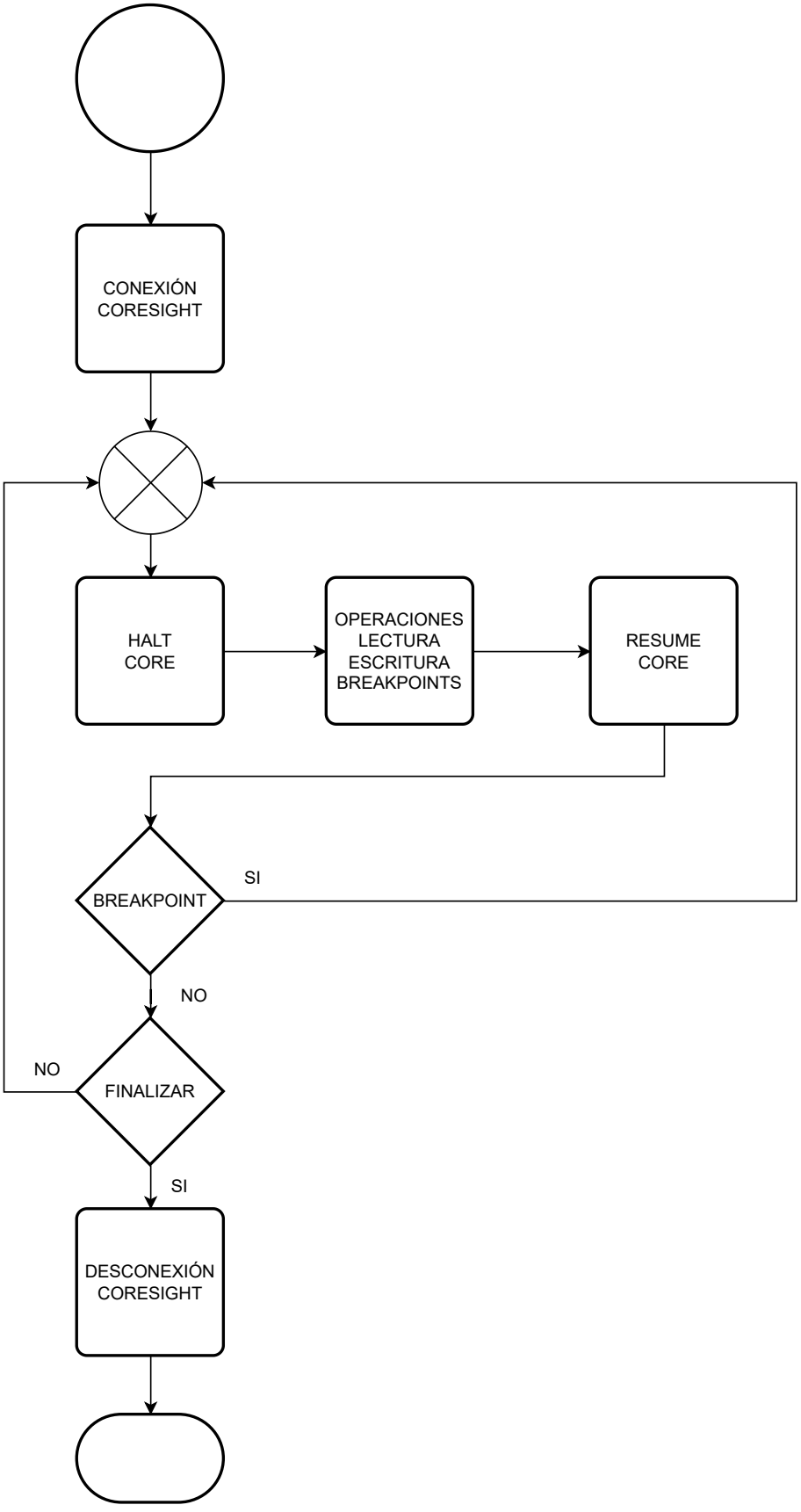


FIGURA 3.5. Flujo de una sesión de depuración.

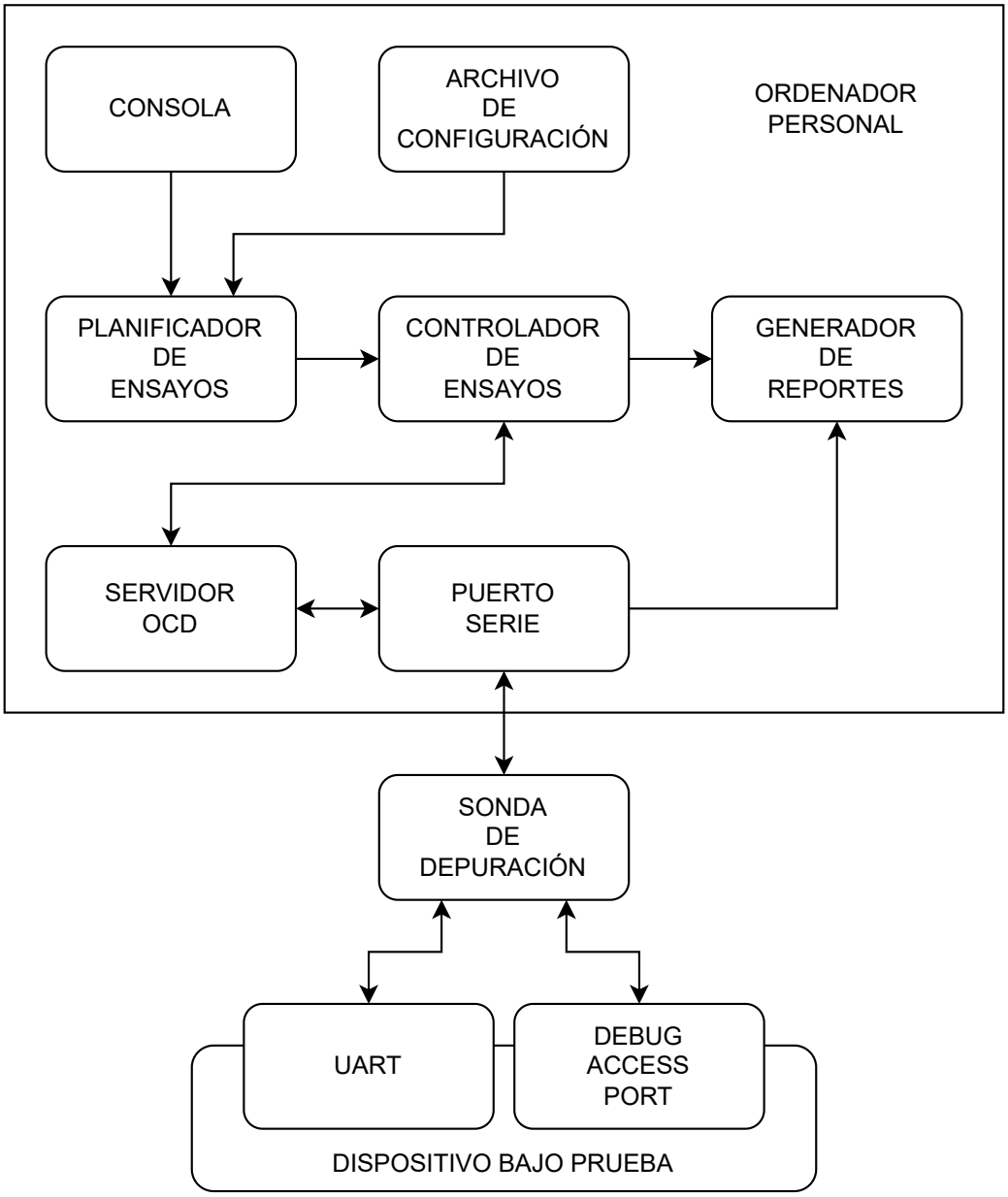


FIGURA 3.6. Diagrama en bloques del sistema de inyección de soft-errors.

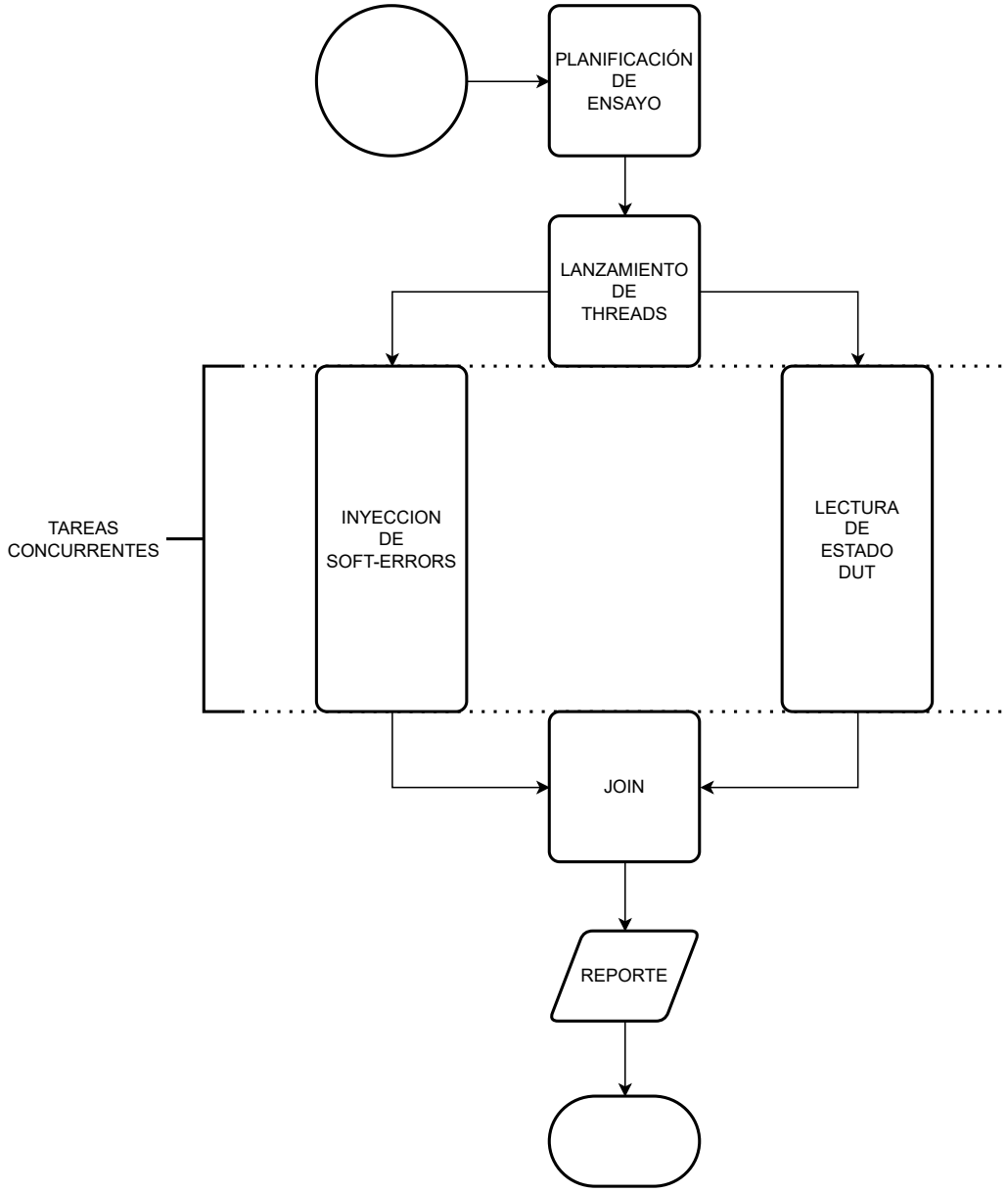


FIGURA 3.7. Flujo de tareas concurrentes.

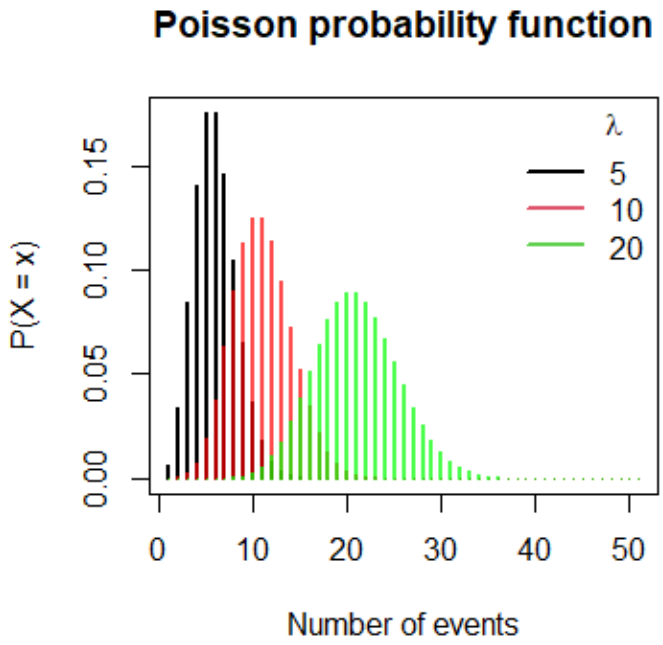


FIGURA 3.8. Gráfico de distribución Poisson[14].

Capítulo 4

Ensayos y resultados

4.1. Laboratorio remoto

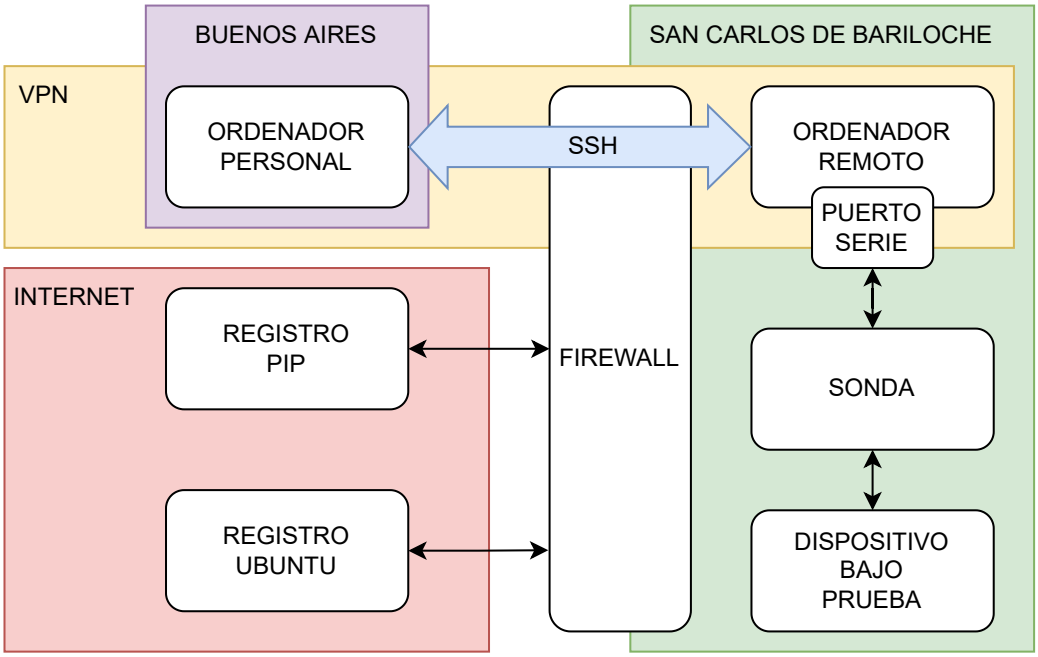


FIGURA 4.1. Diagrama en bloques del laboratorio remoto.

TABLA 4.1

Funcionalidad	Nivel de servicio
Carga de binarios en DUT	++
Comunicación con registro PIP	+++
Comunicación con registro Ubuntu	+++
Comunicación con debug access port	+++
Comunicación con UART	+

4.2. Ensayos de inyector

4.3. Validación con el cliente

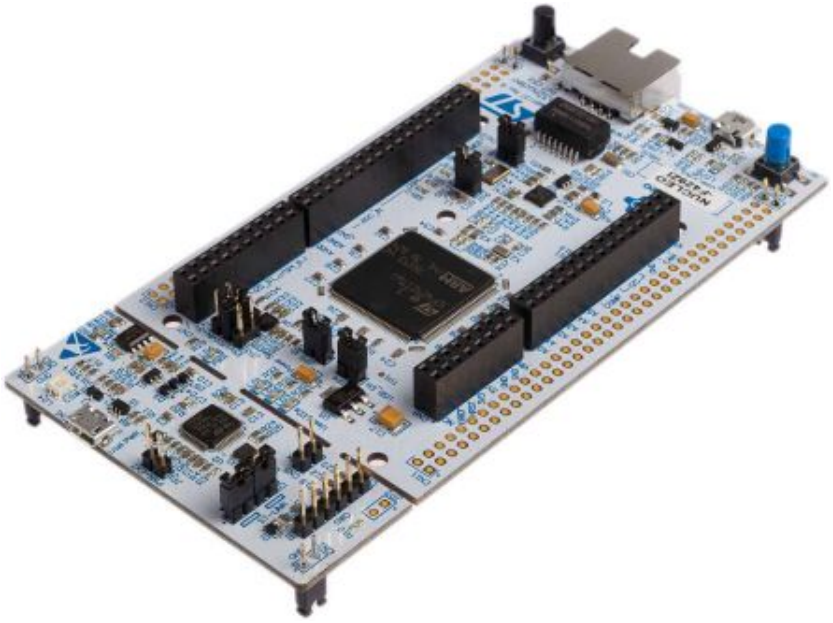


FIGURA 4.2. Dispositivo alternativo *NUCLEO-F429ZI*.

TABLA 4.2. Resumen de ensayos

Ensayo	**Lab. local**	**Lab. remoto**	**DUT alterno**
Escritura SDRAM	+++	+++	+++
Escritura registros CORE	+++	+++	+++
Funcionalidades extras	++	++	+++
Halt CORE	+++	+++	+++
Lectura SDRAM	+++	+++	+++
Lectura registros CORE	+++	+++	+++
Uso concurrente de puerto serie	+++	+	+++
Resume CORE	+++	+++	+++

TABLA 4.3. Resumen de la validación con el cliente

Expectativas	**Cumplimiento**
Acceso a memoria	+++
Acceso al CORE	+++
Biblioteca de ensayos	+++
Capacidad de bit-flip	+++
Configuración del sistema	+++
Distribución de errores	+++
Validación de periféricos	+++
Generación de reportes	+++

Capítulo 5

Conclusiones

Este capítulo explica de forma breve el cierre del trabajo realizado, sus logros y futuro.

5.1. Logros obtenidos

El primer logro a destacar es que se pudo mejorar el estado del arte actual. Su mejora es la capacidad de describir una simulación en términos de la radiación cósmica. Esto se logró al construir una capa de abstracción de aplicaciones. Esta, esconde la gestión de la sesión de depuración. De esta manera, se superaron las técnicas utilizadas en la actualidad. Luego, el desarrollador solo debe describir los errores a introducir en el dispositivo bajo prueba. Finalmente, se agrega valor al mejorar el tiempo, costo y claridad de los ensayos.

Un segundo logro a destacar es la recepción que obtuvo este trabajo. La cual fue muy positiva y con un impacto mayor al esperado. Ya que en la actualidad, INVAP S.E. se encuentra en proceso de integrar la herramienta en su ambiente de desarrollo de satélites. Además, el sistema realizado se utiliza dentro del marco de un proyecto final en la especialización en sistemas embebidos.

Como tercer logro a remarcar es que el trabajo pudo cumplir las expectativas y requerimientos del cliente. Además, se cumplieron sin desvíos respecto a la planificación de tiempo y recursos. Esto fue posible gracias los contenidos de gestión de proyectos que se impartieron durante el desarrollo de esta maestría. Finalmente, estos son los objetivos más destacados:

- Creación de un sistema de inyección de errores transitorios que permita evaluar técnicas de mitigación de errores.
- Acceso a los componentes de interés del dispositivo bajo prueba.
- Biblioteca para el diseño de ensayos de radiación cósmica en lenguaje *Python 3*.
- *Firmware* de autoevaluación del dispositivo bajo prueba que verifique los periféricos de interés.

5.2. Trabajo futuro

La investigación realizada durante la producción del trabajo sugiere que es posible agregar las siguientes funcionalidades:

- Conexión entre el código fuente del dispositivo bajo prueba y el inyector de errores transitorios. Esto se lograría a través del uso de los símbolos de depuración generados en el binario. Con estos símbolos es posible unir un valor del registro *program counter* con una línea en el código fuente. Esta funcionalidad permitiría analizar la vulnerabilidad de un segmento de código escrito en lenguaje *C/C++*.
- Creación de instrucciones específicas para la inyección de *single event functional interrupt*. Estas instrucciones lograrían alterar los registros que definen la configuración de un periférico. De esta manera, se podrían realizar pruebas sobre las técnicas de mitigación de este tipo de errores. La dificultad a superar es que los registros de los periféricos son propios del dispositivo en particular. Para lograr una abstracción genérica se requiere definir el comportamiento de las funciones en tiempo de ejecución.

Si se lograran introducir estas funcionalidades al trabajo realizado, el método de simulación de errores transitorio por *software* quedaría obsoleto. Esto tendría el efecto de modificar el estado del arte; ya que se evitaría el problema que tiene el método al intentar introducir un error cuando el integrado no se encuentra en modo privilegiado.

Como se mencionó en la sección 5.1, este trabajo forma parte de un proyecto final de la especialización en sistemas embebidos. Probablemente se obtengan sugerencias y nuevas ideas durante su desarrollo. Finalmente, se seguirá con interés la evolución del trabajo en curso.

Bibliografía

[1] Steven J. Dick. *Remembering the Space Age*. National Aeronautics y Space Administration, 2008.

[2] J. Hickman y E. Dolman. *Resurrecting the Space Age: A State-Centered Commentary on the Outer Space Regime*. National Aeronautics y Space Administration, 2002.

[3] National Aeronautics y Space Administration. «Human Space Flight Transition Plan». En: *NASA Archive* (2008).

[4] INVAP. *INVAP SE*. https://www.invap.com.ar/. Mayo de 2022. (Visitado 01-05-2022).

[5] spaceradiation.eu. *Structure of space radiation*. https://spaceradiation.eu/structure-of-space-radiation/. Mayo de 2022. (Visitado 01-05-2022).

[6] Raoul Velazco. «Inyección de fallos para el análisis de la sensibilidad a los errores transitorios, "soft errors", provocados por las radiaciones en circuitos integrados». En: *Architectures Robustes of Integrated circuit and systems, Grenoble - France* (2014).

[7] spaceradiation.eu. *Effects of space radiation on electronic devices*. https://spaceradiation.eu/effects-of-space-radiation-on-electronic-devices/. Mayo de 2022. (Visitado 01-05-2022).

[8] Michael D. Griffin. «NASA and the Business of Space». En: *American Astronautical Society 52 Annual Conference* (2005).

[9] Roberto Cibils. «Don't look up. Starlink project: bold venture or economic bubble?» En: *Mission Proyect Workshop 24/25 feb 2022* (2022).

[10] ucl.ac.eu. *Heavy ion latchup tests in louvain la neuve*. https://www.ucl.ac.uk/mssl/sites/mssl/files/styles/owl_carousel/public/heavy_ion_latchup_tests_in_louvain_la_neuve.jpg?itok=1dDe-TEo. Mayo de 2022. (Visitado 01-05-2022).

[11] developer.arm.com. *Cortex M4*. https://developer.arm.com/. Mayo de 2022. (Visitado 01-05-2022).

[12] How to debug: CoreSight basis (Part 3). *Cortex M4*. https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/how-to-debug-coresight-basics-part-3. Mayo de 2022. (Visitado 01-05-2022).

[13] ATMEL. «Atmel 44003 32 bit Cortex M7 Microcontroller SAMV71». En: *Atmel Smart ARM-based Flash MCU* (2016).

[14] r coder.com. *Poisson probability function in R*. https://r-coder.com/wp-content/uploads/2020/10/poisson-probability-function-r.png. Mayo de 2022. (Visitado 01-05-2022).