Índice de tablas

VII

Índice de tablas

1.1. Cinturón de Van Allen 2
1.2. Efectos de la radiación cósmica 4
1.3. Proyección de *debris* 4
1.4. Comparación de métodos de simulación 7

2.1. Servidores de depuración 12
2.2. Resumen de periféricos 14

3.1. Estrategias de depuración 19
3.2. Funcionalidades abstraidas 23

4.1. Resumen del laboratorio remoto 33
4.2. Resumen de ensayos 34
4.3. Resumen de la validación con el cliente 35

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.

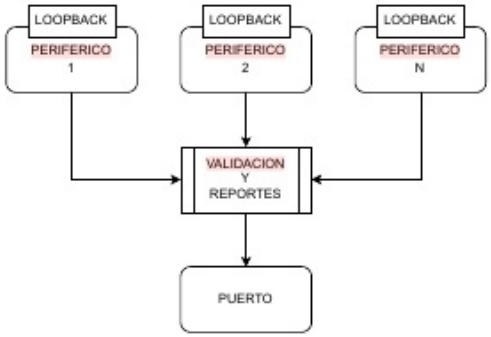

FIGURA 1.6. Diagrama simplificado del dispositivo bajo prueba.

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.

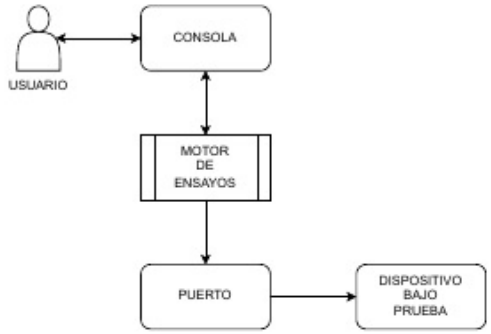

FIGURA 1.7. Diagrama simplificado del sistema de inyección de errores.

8 Capítulo 1. Introducción general

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.

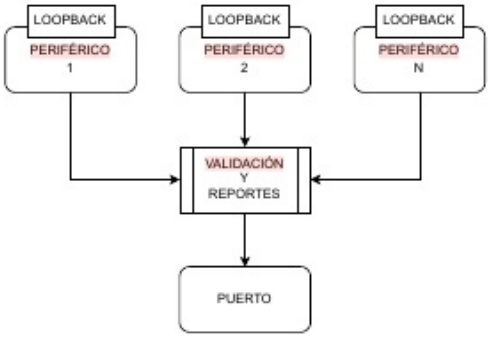

FIGURA 1.6. Diagrama simplificado del dispositivo bajo prueba.

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.

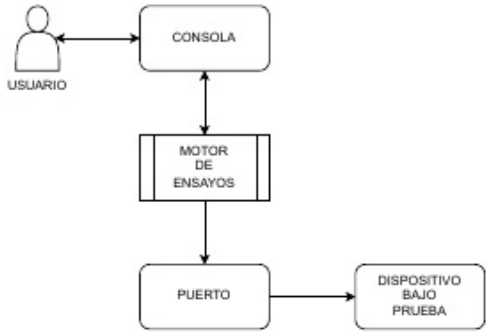

FIGURA 1.7. Diagrama simplificado del sistema de inyección de errores.

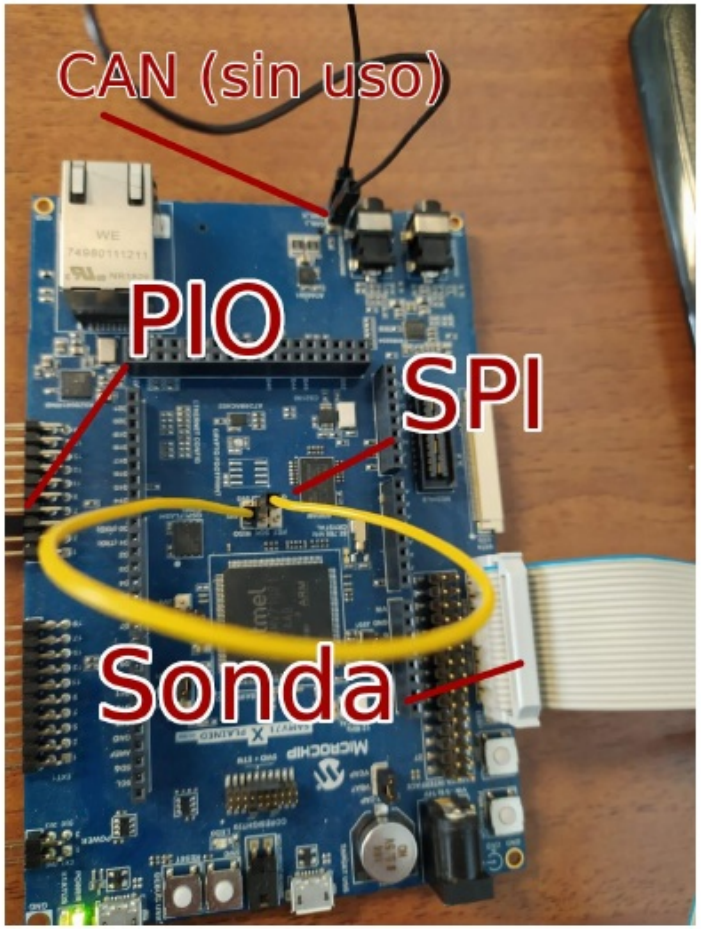

FIGURA 3.3. Fotografía del dispositivo bajo prueba.

3.1. Autoevaluación del dispositivo bajo prueba 19

TABLA 3.1. Comparación entre estrategias de depuración.

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

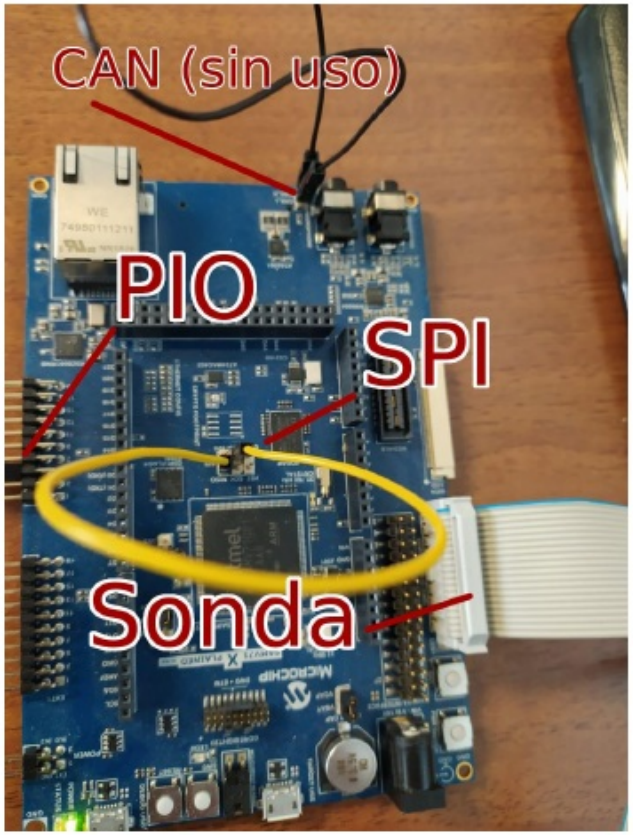

FIGURA 3.3. Fotografía del dispositivo bajo prueba.

TABLA 3.1. Comparación entre estrategias de depuración.

Periférico	Validación	Detección en un ciclo
CAN	Loopback interno	Sí
PIO	Loopback externo	No
SPI	Loopback externo	Sí
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 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.

20 Capítulo 3. Diseño e implementación

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 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.

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.

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

compilador consume una cantidad de memoria que supera el *hardware* disponible en placas como *Raspberry Pi 4B*. Se pudo verificar que este es el único módulo escrito en *Rust*, pero no es posible desacoplarlo del servidor OCD. Finalmente, PyOCD tiene una limitación de plataformas compatibles que podría ser sorteada con *cross* compilación.

La única dificultad en el uso del laboratorio remoto se presentó en las pruebas de la sonda de depuración. Muchas de las pruebas requirieron reiniciar la sonda y esto solo es posible al desconectar el cable *USB*. La operación debió ser realizada por el co-director de este trabajo.

En la tabla 4.1, se puede ver un resumen de las funcionalidades del laboratorio remoto. Las funcionalidades con tres marcas tienen el mismo nivel de servicio que el laboratorio local, las de dos marcas tienen un nivel algo inferior y las que tienen solo una marca tienen un nivel de servicio bajo.

TABLA 4.1. Resumen del laboratorio remoto.

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

El inyector de *soft-errors* se sometió a ensayos en los siguientes ambientes:

- Laboratorio remoto.
- Laboratorio local.
- Dispositivo alternativo *NUCLEO-F429ZI*. Este último ambiente se puede ver en la figura 4.2 y se utilizó para probar si el inyector es genérico. En particular, porque utiliza una sonda de depuración distinta.

Se generaron múltiples archivos de configuración para definir distintos casos de prueba. Luego, se corrieron los ensayos en los tres ambientes y se compararon los resultados. Además, se usó una variante adicional en el dispositivo alternativo. Se probó el comportamiento del inyector sobre un blanco con *mbedOS*. Finalmente, en la tabla 4.2 se puede observar un resumen de los resultados obtenidos. En la tabla se marca con tres cruces los ensayos que arrojaron resultados sobresalientes, con dos cruces los ensayos que mostraron inconvenientes mínimos y con una cruz los ensayos con resultados insatisfactorios.

El dispositivo alternativo arrojó los mejores resultados porque PyOCD tiene el *Device Family Pack*. Por otro lado, el laboratorio remoto tuvo malos resultados en las pruebas de concurrencia. Esto fue así ya que se debía pedir ayuda al personal de INVAP S.E. cada vez que la sonda necesitaba ser reiniciada.

4.2. Ensayos de inyector 33

compilador consume una cantidad de memoria que supera el *hardware* disponible en placas como *Raspberry Pi 4B*. Se pudo verificar que este es el único módulo escrito en *Rust*, pero no es posible desacoplarlo del servidor OCD. Finalmente, PyOCD tiene una limitación de plataformas compatibles que podría ser sorteada con *cross* compilación.

La única dificultad en el uso del laboratorio remoto se presentó en las pruebas de la sonda de depuración. Muchas de las pruebas requirieron reiniciar la sonda y esto solo es posible al desconectar el cable *USB*. La operación debió ser realizada por el co-director de este trabajo.

En la tabla 4.1, se puede ver un resumen de las funcionalidades del laboratorio remoto. Las funcionalidades con tres marcas tienen el mismo nivel de servicio que el laboratorio local, las de dos marcas tienen un nivel algo inferior y las que tienen solo una marca tienen un nivel de servicio bajo.

TABLA 4.1. Resumen del laboratorio remoto.

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

El inyector de *soft-errors* se sometió a ensayos en los siguientes ambientes:

- Laboratorio remoto.
- Laboratorio local.
- Dispositivo alternativo *NUCLEO-F429ZI*. Este último ambiente se puede ver en la figura 4.2 y se utilizó para probar si el inyector es genérico. En particular, porque utiliza una sonda de depuración distinta.

Se generaron múltiples archivos de configuración para definir distintos casos de prueba. Luego, se corrieron los ensayos en los tres ambientes y se compararon los resultados. Además, se usó una variante adicional en el dispositivo alternativo. Se probó el comportamiento del inyector sobre un blanco con *mbedOS*. Finalmente, en la tabla 4.2 se puede observar un resumen de los resultados obtenidos.

En la tabla se marca con tres cruces los ensayos que arrojaron resultados sobresalientes, con dos cruces los ensayos que mostraron inconvenientes mínimos y con una cruz los ensayos con resultados insatisfactorios.

El dispositivo alternativo arrojó los mejores resultados porque PyOCD tiene el *Device Family Pack*. Por otro lado, el laboratorio remoto tuvo malos resultados en las pruebas de concurrencia. Esto fue así ya que se debía pedir ayuda al personal de INVAP S.E. cada vez que la sonda necesitaba ser reiniciada.