
CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

# Emulador de microprocesador LEON3 para desarrollo de software satelital y simuladores

**Autor:**
**Ing. Nicolás Ezequiel Iriarte Fernandez**

Director:
Esp. Lic. Nicolás Eduardo Horro (INVAP)

Jurados:
Nombre del jurado 1 (pertenencia)
Nombre del jurado 2 (pertenencia)
Nombre del jurado 3 (pertenencia)

*Este trabajo fue realizado en la ciudad de Didcot, Oxfordshire, Reino Unido, entre agosto 2024 y diciembre de 2024.*

# *Resumen*

En la presente memoria se describe el diseño e implementación de un emulador de microprocesador LEON3 desarrollado para la empresa INVAP. SE, pensado para su uso en simulaciones espaciales. Este trabajo abarca las etapas de diseño, planificación, desarrollo y documentación, con la finalidad de ofrecer una herramienta robusta que se adapte a las necesidades de la industria espacial. Para su realización, se aplicaron conocimientos adquiridos en arquitectura de microprocesadores, programación de microcontroladores y protocolos de comunicación.

# *Agradecimientos*

Esta sección es para agradecimientos personales y es totalmente **OPCIONAL**.

# Índice general

# Índice de figuras

# Índice de tablas

***Dedicado a... [OPCIONAL]***

# Capítulo 1

# Introducción general

En este capítulo se presenta una introducción general al trabajo final, en el que se describen los antecedentes, la motivación, los objetivos y alcance del trabajo. Además, se presenta el estado del arte en emuladores de sistemas espaciales.

Durante el desarrollo del trabajo se utilizarán términos técnicos y acrónimos que se encuentran en el glosario del apéndice A.

## 1.1. Desarrollo de simuladores espaciales y emuladores

Para productos de ámbito espacial, como lo son los satélites, muchas veces es difícil, y en ocasiones imposible, generar escenarios realistas para pruebas de los elementos que los componen. Ya sea por no poder generar las mismas condiciones ambientales, o porque la naturaleza de la maniobra que se busca probar implicaría un daño a los equipos bajo revisión.

Otro desafío que se presenta es la escasez de hardware disponible para realizar dichas pruebas, ya que los componentes espaciales suelen ser costosos y no se encuentran en grandes cantidades, lo que limita el acceso a estos recursos.

En este contexto, es común replicar los elementos de interés de manera programada, cumpliendo con cierto grado de representatividad. De manera tal que se comporten de la manera más similar posible a su contraparte física. Estos elementos, íntegramente desarrollados en software, se los denominan emulados o simulados. Uno de los componentes que suele generar mayor interés para su simulación es el procesador de la computadora a bordo.

El término computadora a bordo (OBC, por las siglas en inglés de *On Board Computer*) suele referirse a la unidad en la que se ejecuta el software a bordo (OBSW, por sus siglas en inglés de *On Board Software*) y su rol principal es la orquestación del resto de subsistemas en el satélite. Esto incluye recolectar información de los periféricos conectados, analizarla y tomar las decisiones y acciones apropiadas cuando sea requerido.

En la figura 1.1 se muestra la relación entre simuladores, emuladores y software de vuelo. Los simuladores satelitales son más abarcativos e incluyen todos los subsistemas simulados del satélite, entre ellos suele encontrarse el emulador de microprocesador. El emulador, por otro lado, se enfocan en reproducir el comportamiento del microprocesador específico a la misión. Por último, el software de vuelo es el software que se ejecuta en la OBC del satélite, y es el software que se busca probar en los emuladores de microprocesadores.

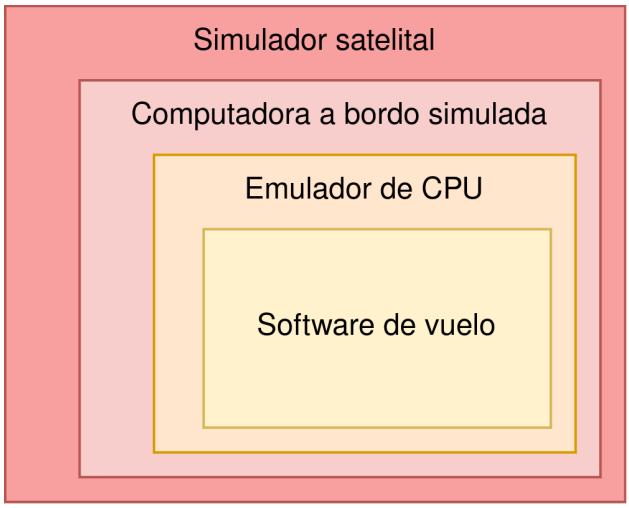
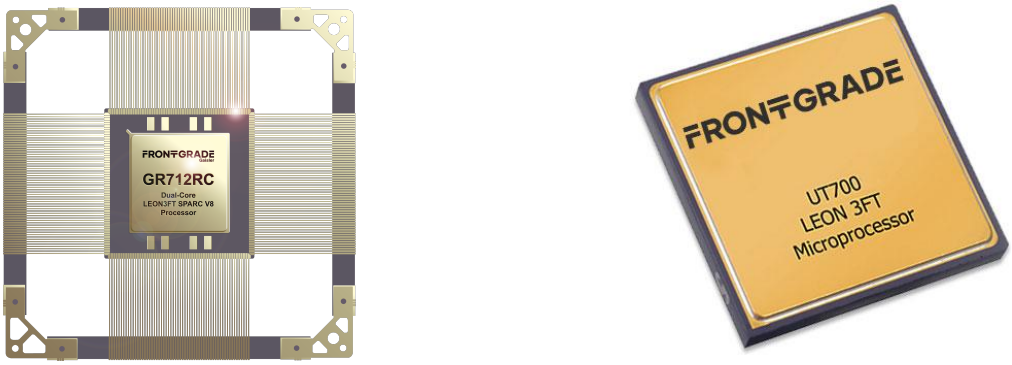



FIGURA 1.1. Relación entre simuladores, emuladores y software de vuelo.

En el caso de los satélites, la OBC suele estar compuesta por un microprocesador, memoria y periféricos, los últimos suelen variar dependiendo de la misión. El microprocesador es el encargado de ejecutar el OBSW, y es el componente que se busca emular en este trabajo. En la figura 1.2 se muestran dos microprocesadores de uso espacial, el GR712RC [1] y el UT700 [2]. Ambos utilizan la arquitectura SPARC V8 [3], que es la arquitectura que se desarrolló en este trabajo.




(A) Procesador GR712RC. (B) Procesador UT700.

FIGURA 1.2. Ejemplo de procesadores de uso espacial.[1]

[1]Imágenes tomadas de https://www.gaisler.com/

Cabe aclarar que el desarrollo de un emulador de microprocesador no es trivial, ya que se debe replicar el comportamiento del microprocesador físico ciclo a ciclo, incluyendo sus instrucciones, registros, y comunicación con los periféricos. Además, se debe tener en cuenta que el microprocesador real puede tener características específicas que no se encuentran en otros microprocesadores, lo que puede dificultar la tarea de emulación.

Dado que el presente desarrollo se realiza en el marco de un posgrado, no se buscó replicar todas las funcionalidades del microprocesador, sino que un subconjunto de las mismas. En particular, el desarrollo se enfocó en la emulación de las instrucciones básicas del microprocesador GR712RC, omitiendo múltiples instrucciones y periféricos.

## 1.2. Estado del arte

En la actualidad, existen dos grandes grupos de emuladores de microprocesadores: los emuladores de microprocesadores de propósito general y los de propósito específico.

Los emuladores de propósito general son aquellos que buscan emular una amplia variedad de microprocesadores de manera genérica, es decir, sin tener en cuenta las características específicas de cada microprocesador. Estos emuladores suelen ser muy complejos y suelen tener un rendimiento inferior a los emuladores de propósito específico, ya que su implementación debe ser lo más abarcativa posible. En particular, los emuladores de propósito general suelen ser utilizados en el desarrollo donde el rendimiento no es un factor crítico.

Un ejemplo de este tipo de emuladores es QEMU [4], un emulador de código abierto que permite emular múltiples arquitecturas de microprocesadores, incluyendo x86, ARM, SPARC y otros. QEMU es un emulador muy popular y es utilizado en múltiples proyectos de software libre y de código abierto. Teniendo una comunidad activa de desarrolladores y usuarios. Haciendo que sea una opción muy atractiva para su uso.

Por otro lado, los emuladores de propósito específico son aquellos que buscan emular un microprocesador específico. Estos emuladores suelen ser más simples que los emuladores de propósito general, ya que no buscan generalidad, sino que buscan emular un microprocesador específico de la manera más precisa y eficiente posible. Pudiendo utilizar optimizaciones específicas que podrían romper con la compatibilidad con otros microprocesadores.

Una clara ventaja de este tipo de emuladores, además de su rendimiento, es que suelen ser más fáciles de utilizar y de integrar con otros sistemas, porque están diseñados desde un inicio para ser embebidos en otros sistemas. Lo que es de suma importancia en el desarrollo de sistemas espaciales, ya que gran parte del desarrollo suele hacerse dentro de la misma organización.

A continuación, en la tabla 1.1 se muestra una tabla comparativa entre emuladores de propósito general y de propósito específico.

TABLA 1.1. Comparación entre emuladores de propósito general y de propósito específico.

| Emuladores específicos | Emuladores genéricos |
|---|---|
| Usualmente privativos y pagos | Sólidas opciones gratis disponibles |
| Alto rendimiento | No enfocado en el rendimiento |
| Alta integrabilidad | Baja integrabilidad |
| Soporte limitado | Comunidades activas y código abierto |

## 1.3. Objetivos y alcance

En esta sección se detallan los objetivos y el alcance del trabajo final, describiendo sus limitaciones y restricciones

### 1.3.1. Objetivos

El objetivo de este trabajo es el diseño e implementación de un emulador de microprocesador LEON3 desarrollado para la empresa INVAP. SE, pensado para su uso en simulaciones espaciales. El emulador debe ser capaz de emular todas las instrucciones necesarias de un *bootloader*, así como instrucciones de propósito general, tales como operaciones de adición, sustracción y acceso a la memoria. Además, debe incluir un modelo de memoria RAM y permitir la carga de binarios en dicha memoria.

Las operaciones de lectura y escritura deberán acceder a una memoria RAM emulada y deberán ser accesibles a través de la API expuesta para interactuar con el emulador.

Otro objetivo del presente trabajo es la creación de manual de usuario de la herramienta, que permita a futuros desarrolladores entender el uso de la herramienta y su integración con otros sistemas.

### 1.3.2. Alcance

El alcance del trabajo se limita a la emulación de un subconjunto de instrucciones del microprocesador LEON3, omitiendo múltiples instrucciones y periféricos. El trabajo no busca ser una herramienta finalizada, sino que busca ser una base sólida para futuros desarrollos. Dadas estas condiciones, el alcance del trabajo se limita a los siguientes puntos:

- Ejecución de instrucciones seleccionadas para ser implementadas en el emulador. Estas instrucciones fueron seleccionadas en base a su importancia y frecuencia de uso en el software de vuelo. Para más detalle sobre cada instrucción se recomienda leer el manual de arquitectura SPARC V8 [3].
- Desarrollo de las operaciones esperables en un emulador de microprocesador. Tales como la carga de binarios en RAM, la ejecución de instrucciones y la interacción con periféricos.
- Creación de una API para la interacción con el emulador. Este es el único punto de interacción que tendrá con el usuario con la aplicación, por lo que es de especial importancia su correcta documentación.

- Desarrollo de un modelo de memoria RAM, capaz de interactuar tanto con la API proporcionada al usuario para lecturas y escrituras arbitrarias, como para el modelo de procesador desarrollado.
- Desarrollo de un mecanismo de depuración de la herramienta. Tal como el volcado del estado del procesador en archivos de formato CSV, mostrando su estado paso a paso durante una ejecución.
- Manual de usuario de la herramienta. Este manual deberá ilustrar en detalle el uso esperado de la herramienta, también deberá exponer sus limitaciones.

# Capítulo 2

# Introducción específica

El presente capítulo describe los elementos componentes del sistema, el principio de funcionamiento y los requerimientos acordados con el cliente sobre el sistema.

## 2.1. Elementos componentes del sistema

Para entender el funcionamiento del sistema, es necesario conocer los elementos que lo componen. La presente sección describirá tanto los elementos físicos necesarios para conocer al sistema que se busca representar, como las herramientas de software utilizadas para el desarrollo del emulador.

### 2.1.1. Elementos físicos del sistema

La presente sección dará una descripción general de los elementos que serán de vital importancia para el proyecto.

#### Bootloader

El *bootloader* es una pieza de software relativamente simple, cuyo principal objetivo es cargar el software de vuelo en la memoria RAM del sistema. Una vez cargado el software de vuelo, le cede el control al software de vuelo, que se encarga de ejecutar la misión del sistema.

En misiones espaciales es común que el *bootloader* tenga la capacidad de verificar la integridad del software de vuelo antes de cargarlo en la memoria RAM, asimismo, de cargar diferentes versiones del software de vuelo en caso de que alguna de ellas se encuentre dañada.

#### Firmware

El *firmware* es un software que se encuentra almacenado en la memoria no volátil del sistema, encargado de inicializar sus periféricos. Usualmente, este componente es desarrollado por un equipo especializado tanto en la misión para la que se está desarrollando el sistema, como en la arquitectura del microprocesador que se está utilizando. En el caso de aplicaciones espaciales, el *firmware* es llamado software de vuelo o *Fly Software* (FSW).

Normalmente, en la memoria no volátil del sistema se almacenan múltiples versiones del software de vuelo. Varias de ellas están presentes para servir de redundancia, en caso de que alguna de ellas se encuentre dañada, como para almacenar

distintas versiones del software de vuelo con modos de operación más limitados pero seguros.

#### Unidad de procesamiento

Todo lo previamente mencionado se ejecuta en la unidad de procesamiento o CPU. La CPU es el componente encargado de ejecutar las instrucciones tanto del *bootloader* como del software de vuelo.

El CPU interactúa con los periféricos del sistema, especialmente con la memoria RAM del sistema, ya que es el medio de almacenamiento masivo de alta velocidad de acceso.

El CPU, una vez encendido, empezará a leer las instrucciones de una posición dada por el fabricante y ejecutarlas secuencialmente. Es deber del desarrollador posicionar al *bootloader* en esa posición de memoria.

### 2.1.2. Software utilizado

La presente sección describirá las herramientas de software que fueron utilizadas para el desarrollo del trabajo.

#### Lenguaje de programación C++

Dado que el emulador es un software que simula el comportamiento de un CPU, es necesario que el lenguaje de programación utilizado sea de bajo nivel, permitiendo mejores tiempos de ejecución y un mayor control sobre los recursos utilizados. Por lo tanto, se decidió utilizar C++ como lenguaje de programación para el desarrollo del emulador.

Otro motivo para su elección es que C++ es un lenguaje de programación ampliamente utilizado en la industria, por lo que facilita la comunicación con otros desarrolladores y la integración con otros sistemas [5].

#### Herramientas Clang

Se utilizó Clang Tidy [6] como herramienta de análisis estático de código. Permitiendo una mayor calidad del código y sintaxis uniforme. Asimismo, se forzaron reglas para evitar malas prácticas comunes en los lenguajes de programación de bajo nivel, tales como el uso de memoria no inicializada, o aritmética de punteros.

También se utilizó Clang Format [7] para mantener un estilo de código uniforme en todo el proyecto.

#### Editor de texto Emacs

Se utilizó el editor de texto Emacs [8] como herramienta de desarrollo tanto del código de programación como para toda su documentación.

Se eligió Emacs debido a la familiaridad con la herramienta, y a la facilidad de integración con otras herramientas de desarrollo [9].

#### Sistema de composición de textos LaTeX

Se utilizó LaTeX [10] como herramienta de escritura de la documentación para el manual de usuario del proyecto. Dicha elección se tomo debido a que LaTeX es una herramienta ampliamente utilizada en la industria para la escritura de documentos técnicos.

Para la edición de los documentos se utilizó el editor de texto Emacs con el modo de edición AucTeX [11].

El manual de usuario busca ser una guía completa y en detalle de la herramienta, no solo explicando la interfaz expuesta, sino decisiones de diseño y funcionamiento interno de la herramienta.

#### Doxygen

Se utilizó Doxygen [12] como herramienta de generación de documentación del código fuente. Con esta herramienta se generó un sitio web que describe la interfaz para interactuar con la herramienta (API). Dicha documentación busca ser una guía rápida para desarrolladores que estén utilizando el emulador.

#### GitLab

Se utilizó GitLab [13] como herramienta de control de versiones para el desarrollo del proyecto. Se optó por esta herramienta debido a que permite una integración sencilla con otras herramientas de desarrollo y familiaridad con la herramienta.

Cabe aclarar que se utilizó una instancia privada de GitLab, para mantener la privacidad del código fuente del proyecto.

## 2.2. Principio de funcionamiento

El funcionamiento de un CPU, y por lo tanto de un emulador, se puede mostrar de manera simplificada de forma gráfica en la figura 2.1.

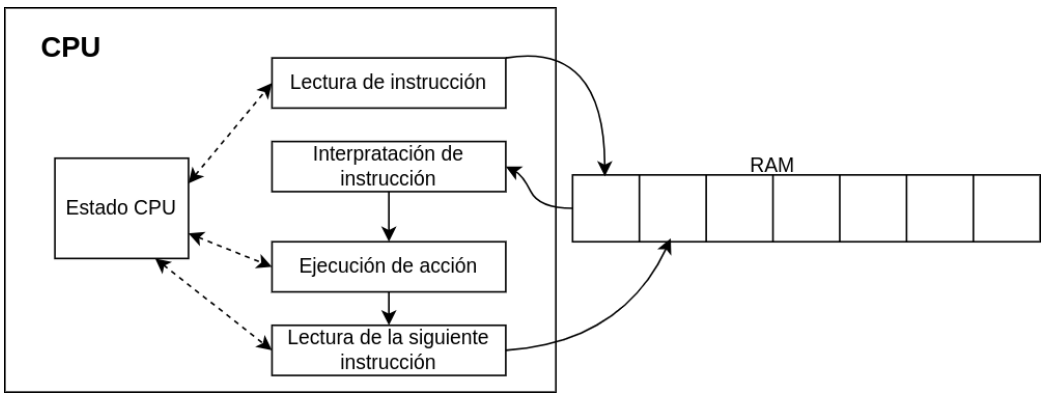



FIGURA 2.1. Funcionamiento de alto nivel de un CPU o emulador.

Tal como se observa en el gráfico, el procesador o CPU lee instrucciones de la memoria RAM y procede a interpretarla. En la arquitectura seleccionada, todas las instrucciones son de 32 bits, por lo que todas las lecturas de memoria son idénticas. La posición de memoria a leer es determinada por un registro especial

llamado *Program Counter* (PC por sus siglas en inglés) ubicado en el estado del CPU.

Una vez leída la instrucción, se la decodifica para identificar de qué tipo es y ejecutar la acción necesaria. Las instrucciones pueden ser de distintos tipos, como aritméticas, de salto condicional o incondicional, y afectan al estado del CPU de diferentes formas.

Este flujo se sucede de manera continua, y las instrucciones no siempre se ejecutan en el orden en el que se encuentran en la memoria. Esto se debe a que ciertas instrucciones pueden modificar el valor del PC, y por lo tanto, la próxima instrucción a ejecutar.

Algo a destacar, es que la arquitectura que se desarrolló es *big-endian*, es decir, los bytes más significativos se encuentran en las direcciones de memoria más altas. Esto es importante ya que la arquitectura x86, en la que se desarrolló el emulador, es *little-endian*, lo que implica que se deberán realizar conversiones de endianess en cada lectura o escritura de memoria.

Este detalle es de vital importancia, ya que un mismo valor de memoria puede ser interpretado de manera diferente dependiendo de la arquitectura utilizada. La figura 2.2 muestra un ejemplo de cómo se interpretaría el valor `0x12345678` en una arquitectura *big-endian* y en una *little-endian*.

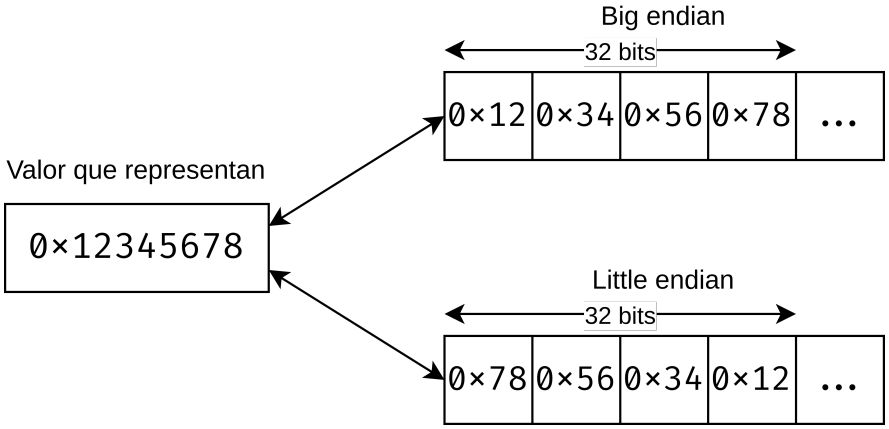



FIGURA 2.2. Relación entre valores y su representación en memoria.

Es importante recordar que, aunque los bytes estén en distintos ordenes en la memoria, el valor que representan es exactamente el mismo.

## 2.3. Requerimientos

A continuación, se presentan los requerimientos funcionales y no funcionales acordados con el cliente.

### 2.3.1. Requerimientos funcionales

1. El emulador deberá ejecutar los mismos binarios que se utilizan en el hardware real.
2. El sistema debe ser compatible con el sistema operativo Linux.

3. El emulador deberá poder ejecutar correctamente parte del set de instrucciones del procesador real.
4. Se deberán desarrollar tests unitarios que verifiquen el set de instrucciones desarrollado.
5. Se deberá desarrollar un ambiente de automatización de pruebas en GitLab CI.
6. El software podrá utilizarse como biblioteca compartida.

### 2.3.2. Requerimientos no funcionales

1. El software debe mantenerse bajo control de versiones en GitLab.
2. El software deberá ser escrito en C++.
3. La API expuesta deberá estar documentada con Doxygen.
4. La API expuesta deberá implementarse en C.
5. Se realizará un manual de usuario que describa los funcionamientos clave del software.

# Capítulo 3

# Diseño e implementación

El presente capítulo describe los elementos componentes del sistema en detalle, las consideraciones y decisiones técnicas tomadas durante el desarrollo del emulador, el diagrama de bloques del sistema, la arquitectura del software y los módulos que lo componen.

## 3.1. Consideraciones y desiciones técnicas

El trabajo realizado se puede dividir en cuatro grandes componentes, tal como se muestra en la figura 3.1:

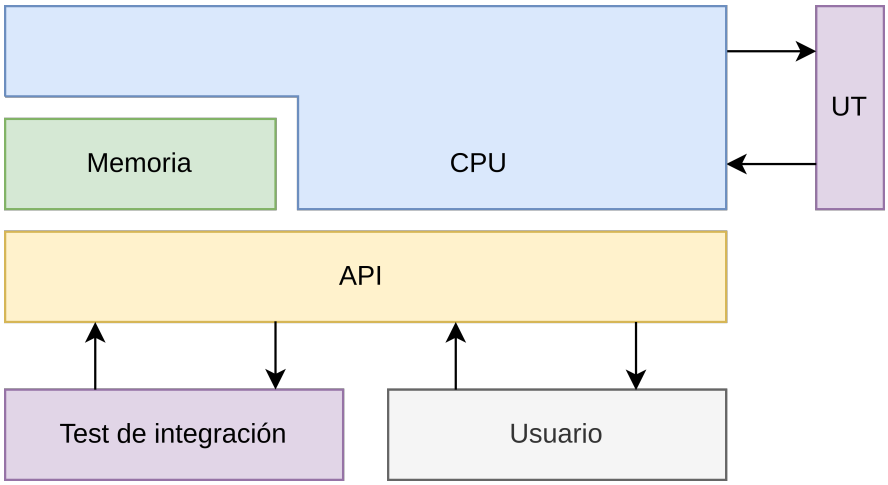



FIGURA 3.1. Diagrama en bloques del proyecto.

Dichos componentes serán desarrollados en el transcurso del capítulo.

1. Memoria.
2. CPU.
3. API.
4. Tests unitarios (UT) y de integración.

### 3.1.1. Memoria

En el trabajo se ha desarrollado un único modelo de memoria volátil (RAM) que consiste en un array de 32 gigabytes de tamaño. A dicho abstracción se le agregaron las operaciones de lectura y escritura de 32 bits. Por facilidad de testeo se

decidió inicializar toda la memoria con ceros para discernir entre escrituras erróneas y lecturas de direcciones no inicializadas.

Una decisión técnica que se tomó con la memoria del sistema fue la de utilizar *little-endian* en vez de *big-endian* como lo haría el hardware real. La razón de esta decisión fue que la arquitectura x86, en la que se desarrolló el emulador, utiliza *little-endian*, por lo que se evita tener que realizar conversiones de endianess en cada lectura o escritura de memoria, y dichas transformaciones fueron reubicadas en la capa de API para garantizar que el usuario final pueda observar la memoria memoria emulada con el mismo ordenamiento que en el hardware real.

Esta consideración implica que el firmware que se cargue se desee ejecutar deberá ser transformado a *little-endian* durante la carga para cumplir con el requerimiento de poder ejecutar los mismos binarios que se utilizan en el hardware real.

### 3.1.2. CPU

### 3.1.3. API

### 3.1.4. Tests unitarios y de integración

---

**NOTA:** Hasta aquí llegué, lo siguiente son borradores que se utilizarán cuando sean necesarios. Recordar de revisar el apéndice A.

---

En el presente trabajo, no se ha modelado una memoria persistente, por lo que se ha agregado una función que recibe un archivo binario (FSW) y lo carga en la memoria RAM del sistema. Dicha funcion posicionará al binario en la dirección correcta de memoria, donde el CPU lo pueda ejecutar.

La figura 3.2 muestra un diagrama de flujo del procedimiento de carga y ejecución de un software en el emulador desarrollado. El diagrama asume que ya se posee un archivo binario valido para la arquitectura SPARC V8.

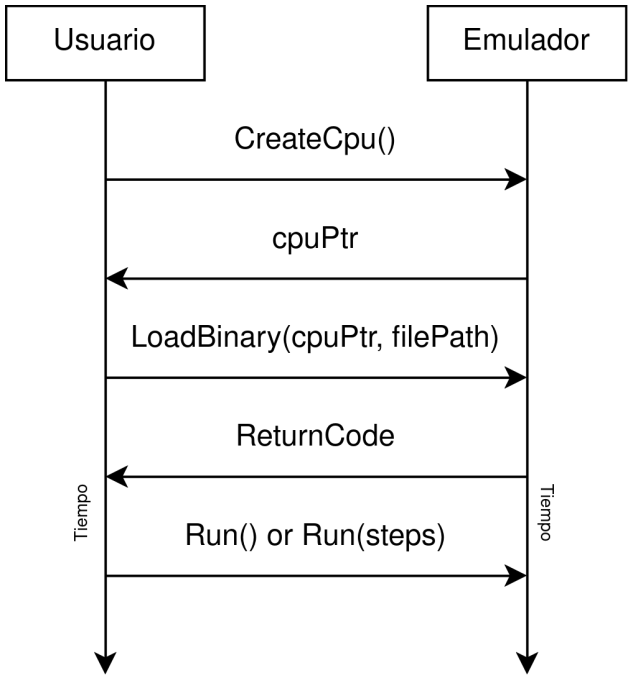



FIGURA 3.2. Proceso de carga de binarios en emulador.

Todo el procedimiento descrito anteriormente se lleva a cabo en la computadora a bordo (OBC) del sistema. La OBC es el componente que se encarga de orquestar el resto de los subsistemas del sistema. En un escenario real, la OBC tendría más componentes que los que se han modelado en este trabajo, tales como periféricos de comunicación UART, CAN y SpaceWire, entre otros. La figura 3.3, tomada de la página de Gaisler [1] y editada, muestra en detalle todos los componentes de la OBC GR712RC resaltando en rojo los componentes que se han modelado en el presente trabajo.

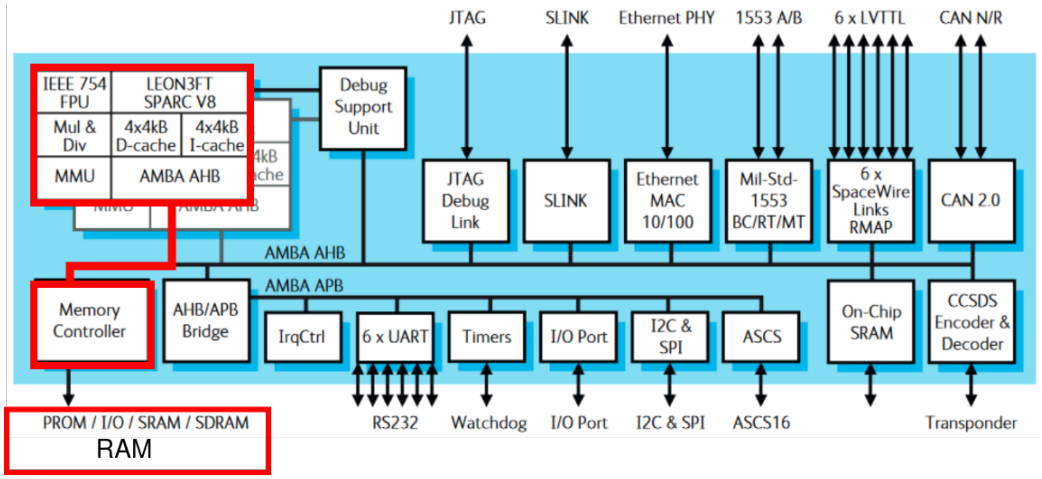



FIGURA 3.3. Componentes emulados.

Cabe destacar que se utilizó la placa de desarrollo GR712RC como referencia para el desarrollo del emulador, ya que es una placa de desarrollo de uso común en aplicaciones espaciales. Se pueden destacar las siguientes simplificaciones realizadas:

- Se ha modelado únicamente un CPU. Esta simplificación permite omitir todas las instrucciones de sincronización y manejo de interrupciones que se deberían implementar en un sistema multi-core.
- Se ha modelado el controlador de memoria (MCU por sus siglas en inglés *Memory Controller Unit*) correctamente.
- Las memorias PROM, I/O, SRAM y SDRAM se han modelado como un único bloque de memoria RAM. Dicha simplificación reduce significativamente el número de modelos a implementar y no afecta el funcionamiento del emulador.

Para la CPU, se desarrollaron las siguientes instrucciones de la arquitectura SPARC V8:

1. `UNIMP`: Genera un interrupción de procesador.
2. `Bicc`: Salto condicional (Números enteros).
3. `SETHi`: Escritura de los 22bits mas significativos de un registro.
4. `FBfcc`: Salto condicional (Números punto flotante).
5. `CBfcc`: Salto condicional (Códigos de condición).
6. `CALL`: Llamada a subrutina.
7. `ADD`: Adición.
8. `ADDcc`: Adición con código de condición.
9. `ADDX`: Adición con retorno de carro.
10. `SUB`: Sustracción.
11. `SUBcc`: Sustracción con código de condición.
12. `SUBX`: Sustracción con retorno de carro.
13. `AND`: Operación `AND` lógica.
14. `ANDcc`: Operación `AND` lógica con actualización de código de condición.
15. `ANDN`: Operación de `AND` negada lógica.
16. `OR`: Operación `OR` lógica.
17. `XOR`: Operación de *exclusive or*.
18. `ORcc`: Operación `OR` lógica con actualización de código de condición.
19. `SLL`: Desplazamiento lógico a la izquierda.
20. `SRL`: Desplazamiento lógico a la derecha.
21. `SRA`: Desplazamiento aritmético a la derecha.
22. `RDPSR`: Lectura del registro de estado del procesador.

23. `RDY_RDASR`: Lectura del registro `Y` del procesador.
24. `WRY`: Escritura del registro `Y` del procesador.
25. `WRPSR`: Escritura del registro `PSR` del procesador.
26. `WRWIM`: Escritura del registro `WIM` del procesador.
27. `WRTBR`: Escritura del registro `TBR` del procesador.
28. `TICC`: Interrupción en código de condición de enteros.
29. `JMPL`: Salto incondicional.
30. `FLUSH`: Limpieza de operaciones pendientes.
31. `SAVE`: Guardado de ventana de procesamiento.
32. `RESTORE`: Carga de ventana de procesamiento.

## 3.2. Consideraciones y decisiones tecnicas

Un desafío que se tuvo que resolver temprana en el desarrollo del emulador fue la endianess del sistema. El endianess es el orden en el que se almacenan los bytes en la memoria. En el caso de la arquitectura SPARC V8, se utiliza el ordenamiento *big-endian*, lo que significa que el byte más significativo se almacena en la dirección de memoria más alta. Por otro lado, la arquitectura x86 (Es decir, nuestros computadoras de escritorio) utiliza el ordenamiento *little-endian*, donde el byte menos significativo se almacena en la dirección de memoria más baja. Por lo tanto, al momento de interpretar los datos obtenidos de la memoria, se debe reordenar los bytes para que tengan sentido. Dicha problemática es propia únicamente del emulado

## 3.3. Diagrama de bloques

## 3.4. Arquitectura del software

## 3.5. Modulos componentes del software

## 3.6. Desarrollo del software

# Capítulo 4

# Ensayos y resultados

# Capítulo 5

# Conclusiones

## 5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

## 5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

# Apéndice A

# Glosario

La tabla A.1 muestra la lista de acrónimos y términos utilizados en el presente trabajo y busca ser una referencia rápida para el lector.

TABLA A.1. Glosario de acrónimos.

| Acrónimo | Significado |
|---|---|
| **API** | Interfaz de programación de aplicaciones (*Application Programming Interface*). |
| **CAN** | Red de área de control (*Controller Area Network*). |
| **CPU** | Unidad central de procesamiento (*Central Processing Unit*). |
| **CSV** | Valores separados por comas (*Comma Separated Values*). |
| **FSW** | Software de vuelo (*Flight Software*). |
| **MCU** | Unidad de control de memoria (*Memory Control Unit*). |
| **OBC** | Computadora a bordo (*On Board Computer*). |
| **OBSW** | Software a bordo (*On Board Software*) |
| **PROM** | Memoria programable de solo lectura (*Programmable Read-Only Memory*). |
| **PC** | Contador de programa (*Programm Counter*). |
| **RAM** | Memoria de acceso aleatorio (*Random Access Memory*). |
| **UART** | Receptor transmisor asíncrono universal (*Universal Asynchronous Receiver Transmitter*). |

# Bibliografía


[1] FrontGrade. *Microprocesador GR712RC*. https://www.gaisler.com/index.php/products/components/gr712rc. Ene. de 2023. (Visitado 13-09-2024).

[2] FrontGrade. *Microprocesador UT700*. https://www.frontgrade.com/product/ut700. Ene. de 2024. (Visitado 13-09-2024).

[3] SPARC. *Documentos técnicos*. https://sparc.org/technical-documents/. Sep. de 2024. (Visitado 13-09-2024).

[4] QEMU. *Sitio web oficial de QEMU*. https://www.qemu.org/. Ene. de 2024. (Visitado 13-09-2024).

[5] NASA. *Proyectos C++ públicos de la NASA*. https://github.com/orgs/nasa/repositories?q=lang%3Ac%2B%2B&type=all. Ene. de 2024. (Visitado 27-09-2024).

[6] LLVM. *Página oficial de Clang Tidy*. https://clang.llvm.org/extra/clang-tidy/. Ene. de 2024. (Visitado 27-09-2024).

[7] LLVM. *Página oficial de Clang Format*. https://clang.llvm.org/docs/ClangFormat.html. Ene. de 2024. (Visitado 27-09-2024).

[8] GNU. *Página oficial de Emacs*. https://www.gnu.org/software/emacs/. Ene. de 2024. (Visitado 27-09-2024).

[9] MELPA. *Repositorios de paquetes para Emacs MELPA*. https://melpa.org/. Ene. de 2024. (Visitado 27-09-2024).

[10] LaTeX. *Página oficial de Latex*. https://www.latex-project.org/. Ene. de 2024. (Visitado 27-09-2024).

[11] GNU. *Página oficial de AUCTeX*. https://www.gnu.org/software/auctex/. Ene. de 2024. (Visitado 27-09-2024).

[12] Doxygen. *Página oficial de Doxygen*. https://www.doxygen.nl/index.html. Ene. de 2024. (Visitado 27-09-2024).

[13] GitLab. *Página oficial de GitLab*. https://gitlab.com. Ene. de 2024. (Visitado 27-09-2024).