Emulador de microprocesador Leon3

Autor:
Ing. Iriarte Fernandez, Nicolás Ezequiel

Director:
Esp. Lic. Horro, Nicolás Eduardo (INVAP.S.E.)

Esta planificación fue realizada en el curso de Gestión de proyectos entre el 29/08/2023 y el 10/10/2023.

Índice

1. Descripción técnica-conceptual del proyecto a realizar **5**

2. Identificación y análisis de los interesados **6**

3. Propósito del proyecto **7**

4. Alcance del proyecto **7**

5. Supuestos del proyecto **7**

6. Requerimientos **7**

7. Historias de usuarios (*Product backlog*) **8**

8. Entregables principales del proyecto **9**

9. Desglose del trabajo en tareas **9**

10. Diagrama de Activity On Node **11**

11. Diagrama de Gantt **13**

12. Presupuesto detallado del proyecto **15**

13. Gestión de riesgos **15**

14. Gestión de la calidad **17**

15. Procesos de cierre **18**

Registros de cambios

Revisión	Detalles de los cambios realizados	Fecha
0	Creación del documento	29/08/2023
1	Se completa hasta el punto 5 inclusive	07/09/2023
2	Se completa hasta el punto 9 inclusive Se aplican correcciones hasta el punto 5	10/09/2023
3	Se completa hasta el punto 12 inclusive Se aplican correcciones hasta el punto 9	17/09/2023

Acta de constitución del proyecto

Buenos Aires, 29/08/2023

Por medio de la presente se acuerda con el Ing. Iriarte Fernandez, Nicolás Ezequiel que su Trabajo Final de la Carrera de Especialización en Sistemas Embebidos se titulará "Emulador de microprocesador Leon3", consistirá esencialmente en la implementación de un prototipo de un emulador de Leon3 para desarrollo de software satelital y simuladores, y tendrá un presupuesto preliminar estimado de 621 horas de trabajo y US$19.280, con fecha de inicio 29/08/2023 y fecha de presentación pública 16/05/2024.

Se adjunta a esta acta la planificación inicial.

Dr. Ing. Ariel Lutenberg
Director posgrado FIUBA

Pinedo, Matías
INVAP.SE

Esp. Lic. Horro, Nicolás Eduardo
Director del Trabajo Final

1. Descripción técnica-conceptual del proyecto a realizar

Para productos de ámbitos espaciales, 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.

En este contexto, es común replicar los elementos de interés de manera programada, cumpliendo con cierto grado de representación. De manera tal que se comporten de la manera más similar posible a su contraparte física. Estos elementos, íntegramente desarrollados en software, se los llaman emulados o simulados. Uno de los componentes que se suele tener mayor interés en simular 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 el control de los subsistemas del satélite. Esto incluye recolectar información de diferentes subsistemas, analizarla y tomar las decisiones y acciones apropiadas cuando sea requerido.

El foco principal de este trabajo es el microprocesador contenido en la OBC. Cabe destacar, que hoy en día existen emuladores tanto de código abierto como privativos para distintos microprocesadores. Un ejemplo claro de emulador de código abierto es Qemu, que abarca un amplio abanico de microprocesadores, entre ellos, algunos utilizables en el ámbito espacial.

En el contexto de los emuladores, cada uno aborda el problema en cuestión, pero conlleva sus propias desventajas. Por ejemplo, en el caso de emuladores como Qemu, diseñados para ser genéricos, pueden requerir un esfuerzo adicional para adaptarlos a un procesador específico, como es el GR712RC. Esto se debe a que la flexibilidad genérica puede implicar una pérdida de rendimiento o la necesidad de familiarizarse con su API.

Por otro lado, los emuladores comerciales suelen estar optimizados para procesadores específicos, lo que puede brindar un excelente rendimiento en esas circunstancias particulares. Sin embargo, estos emuladores pueden ser limitados en cuanto a su capacidad de personalización, instrumentación y capacidad de integración, lo que puede dificultar su adaptación a necesidades específicas o la depuración de software en entornos no nominales.

Bajo estas premisas se plantea crear un emulador de microprocesador Leon3 para desarrollo de software satelital y simuladores. En la figura 1 se deja en evidencia un diagrama de bloques del hardware que se busca replicar. Al ser un desarrollo a medida, se tendrá la ventaja de la no-diversificación del procesador, es decir, estará únicamente orientado a un solo microprocesador. Esperando una ganancia en performance comparado con su contraparte de código abierto. Al mismo tiempo, se tendrá un conocimiento extenso del alcance y limitaciones de las capacidades del software en cuestión. Haciendo, de esta manera, más simple la integración y depuración en su uso.

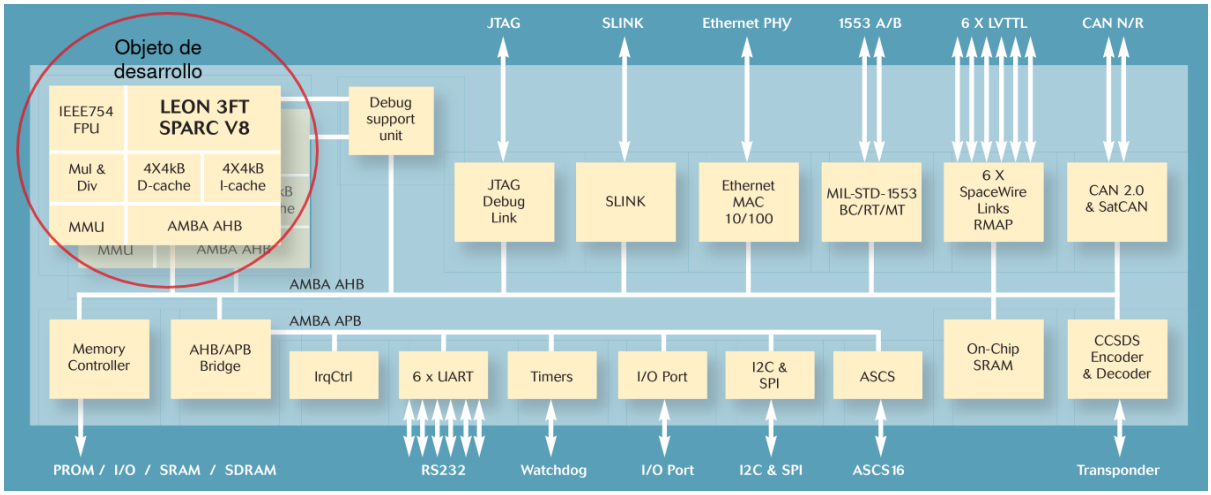

Figura 1. Diagrama en bloques del sistema.

Para lograr el objetivo descrito, se ha decidido utilizar el framework LLVM. Este se empleará para generar el conjunto de instrucciones admitido por el microprocesador Leon3. La elección de LLVM se fundamenta en varios factores clave que hacen de esta plataforma una opción sólida y versátil. Tales como que cuenta con una comunidad de desarrollo activa y comprometida, lo que garantiza un soporte continuo y actualizaciones regulares que son esenciales para mantener la robustez y eficiencia del sistema. También, al poseer soporte nativo con Leon3, es una fuente de análisis y guía en los casos donde los manuales no sean lo suficientemente claros.

Por último, es importante subrayar que el presente trabajo no se iniciará desde cero, sino que se aprovechará una base de código preexistente gentilmente proporcionada por el director de tesis. Esta base no solo servirá como una hoja de guía para el proceso de desarrollo, sino que también desempeñará un papel clave en la aceleración de todo el proyecto.

2. Identificación y análisis de los interesados

Rol	Nombre y Apellido	Organización	Puesto
Cliente	Lic. Pinedo, Matías	INVAP.SE	Especialista Tecnológico
Responsable	Ing. Iriarte Fernandez, Nicolás Ezequiel	FIUBA	Alumno
Orientador	Esp. Lic. Esp. Lic. Horro, Nicolás Eduardo	INVAP.S.E.	Director Trabajo final
Usuario final	Departamento de Embebidos y Sistemas Críticos	INVAP.S.E.	-

- Cliente: Lic. Matías Pinedo, del Departamento de Embebidos y Sistemas Críticos de la empresa INVAP S.E., quien aporta su visión y experiencia necesaria para desarrollar el proyecto.
- Orientador: es el director del trabajo final, quien aportará sus conocimientos técnicos y experiencia para la realización del proyecto.
- Usuario final: será el Departamento de Embebidos y Sistemas Críticos de la empresa INVAP S.E.

3. Propósito del proyecto

El propósito de este proyecto es desarrollar un emulador para ejecución de software de vuelo para microprocesadores de la familia Leon3 utilizando el framework LLVM a partir de un código preexistente provisto.

4. Alcance del proyecto

El presente trabajo incluye:

- El desarrollo del software de emulación.
- La implementación de una API para su uso en el lenguaje de programación C.
- El desarrollo de algún mecanismo de depuración para una ejecución dada.

El proyecto no incluye:

- El desarrollo de software para el procesador Leon3 representativo con el uso que se le piense dar.
- La implementación de bibliotecas utilitarías para generación de código fuente.
- La emulación completa sobre todos periféricos asociados.

5. Supuestos del proyecto

Para el desarrollo del presente proyecto se supone que:

- La empresa va a proveer de un modelo de referencia (físico o emulado) para la realización del proyecto.
- El proyecto puede ser realizado sin necesidad de hacer pruebas en campo.
- Se contará con soporte y apoyo por parte de expertos de la empresa.
- El proyecto puede ser realizado sin necesidad de ir a las oficinas de la empresa.

6. Requerimientos

1. Requerimientos funcionales
   1.1. El emulador deberá ejecutar los mismos binarios que se utilizan en el hardware real.
   1.2. El sistema debe ser compatible con el sistema operativo Linux.
   1.3. El emulador deberá poder ejecutar correctamente parte del set de instrucciones del procesador real.

1.4. Se deberán desarrollar tests unitarios que verifiquen el set de instrucciones desarrollado.

1.5. Se deberá desarrollar un ambiente de automatización de pruebas en Gitlab CI.

1.6. El software podrá utilizarse como biblioteca compartida.

2. Requerimientos de documentación

2.1. La API expuesta deberá estar documentada con Doxygen.

2.2. Se realizará un manual de usuario que describa los funcionamientos clave del software.

2.3. Se deberá redactar un informe de avance y la memoria técnica final del proyecto.

3. Requerimientos de desarrollo de software

3.1. El software deberá ser escrito en C++.

3.2. La API expuesta deberá ser en C.

3.3. El software debe mantenerse bajo control de versiones en Gitlab.

7. Historias de usuarios (*Product backlog*)

Roles:

- Desarrollador de modelos simulados: quien se encarga de desarrollar dispositivos de manera simulada. Tales como podrían ser memorias, FPGAs y/o periféricos.
- Desarrollador de software de vuelo: quien se encarga de desarrollar un driver o software que usa interfaces de bajo nivel. Durante el diseño e implementación, utiliza este sistema para interactuar con el hardware.
- Ingeniero de pruebas: quien se encarga de automatizar los ensayos de calificación de hardware y/o de componentes que lo emulen.

Story Points:

- Se analizarán las historias según dificultad, complejidad e incertidumbre, tomando valores de Fibonacci usando el siguiente criterio:
  1. Dificultad: cantidad de trabajo a realizar. Representado con la letra "**D**".
     - Bajo: 1
     - Medio: 3
     - Alto: 13
  2. Complejidad: complejidad de trabajo a realizar. Representado con la letra "**C**".
     - Bajo: 1
     - Medio: 5
     - Alto: 13
  3. Incertidumbre: riesgo del trabajo a realizar. Representado con la letra "**I**".
     - Bajo: 1

- Medio: 3
- Alto: 5

Historias de Usuarios:

- Como desarrollador de modelos simulados quiero poder comunicarme con el software de vuelo para verificar el correcto functionamiento de mi desarrollo. D: 3, C: 5, I: 5 Total = 13p.
- Como desarrollador de modelos simulados quiero poder ejecutar el software de vuelo como una biblioteca para evitar posibles desfasajes entre componentes. D: 3, C: 1, I: 1. Total = 5p.
- Como desarrollador de software de vuelo quiero poder estimular los registros, memorias y hardware para el desarrollo de controladores (*drivers*). D: 13, C: 13, I: 5. Total = 34p.
- Como desarrollador de software de vuelo quiero que mi software corra a tiempo real para que las emulaciones sean representativas. D: 5, C: 5, I: 5. Total = 21p.
- Como ingeniero de pruebas quiero que mis pruebas puedan ser ejecutadas automáticamente en un entorno de desarrollo continuo (por ejemplo GitLab CI) para ahorrar tiempo y evitar errores humanos. D: 3, C: 2, I: 2. Total = 8p.

8. Entregables principales del proyecto

Los entregables del proyecto son:

- Biblioteca de emulación con sus respectivos headers.
- Documentación Doxygen de la API.
- Manual de usuario.

9. Desglose del trabajo en tareas

1. Gestión del proyecto (24 h):
   1.1. Definición de requerimientos con el cliente (4 h).
   1.2. Planificación (20 h).
2. Investigación y capacitación en la arquitectura del microprocesador (33 h):
   2.1. Recopilación de documentación online (3 h).
   2.2. Lectura y comprensión de los documentos (10 h).
   2.3. Capacitación sobre el conjunto de instrucciones del procesador (20 h).
3. Desarrollo del ambiente de automatización (16 h):
   3.1. Creación del repositorio de desarrollo (2 h).

3.2. Creación del ambiente de CI/CD para tests (4 h).
3.3. Desarrollo del ambiente CI/CD para documentación (6 h).
3.4. Creación del ambiente de CI/CD para empaquetado (4 h).

4. Internalización de código pre-existente (46 h):

4.1. Revisión y análisis del código existente (12 h).
4.2. Selección del código a reutilizar e importación (30 h).
4.3. Refactorización del código (4 h).

5. Investigación y capacitación en el framework de LLVM (50 h):

5.1. Lectura de documentación (20 h).
5.2. Internalización con su API (20 h).
5.3. Pruebas de límites y alcances del framework (10 h).

6. Desarrollo del software (180 h):

6.1. Desarrollo de modelos de periféricos (40 h).
6.2. Desarrollo de modelos de memorias y registros (30 h).
6.3. Desarrollo de instrucciones. Se estima funcionalidad mínima de un subset de instrucciones, calculando 1 hora por instrucción. (110 h).
1) Desarrollo de instrucciones aritméticas y lógicas (40 h).
2) Desarrollo de instrucciones de carga y descarga de memoria (40 h).
3) Desarrollo de instrucctiones escritura/lectura de registros de control (30 h).

7. Desarrollo de tests unitarios y de API (90 h):

7.1. Pruebas unitarias de modelos (20 h).
7.2. Pruebas unitarias de instrucciones (40 h).
7.3. Pruebas de API (20 h).
7.4. Corrección de errores (10 h).

8. Documentación (50 h):

8.1. Creación del manual de usuario (40 h).
8.2. Documentación de API en Doxygen (10 h).

9. Presentación del trabajo (112 h):

9.1. Recopilación de datos de ensayos y organización de resultados para las memorias (40 h).
9.2. Redacción de las memorias (40 h).
9.3. Presentación del trabajo (32 h).

Cantidad total de horas: 621 horas.

10. Diagrama de Activity On Node

En la figura 2 se ilustra el diagrama de *Activity on Node*. Las tareas están agrupadas en la misma forma que en la sección anterior:

1. Gestión del proyecto.
2. Investigación y capacitación en la arquitectura del microprocesador.
3. Desarrollo del ambiente de automatización.
4. Internalización de código pre-existente.
5. Investigación y capacitación en el framework de LLVM.
6. Desarrollo del software.
7. Desarrollo de tests unitarios y de API.
8. Documentación.
9. Presentación del trabajo.

Cada tarea está etiquetada con una letra **T** seguida de su número de tarea. Por ejemplo, T1.1 corresponde a la tarea "1.1 - Definición de requerimientos con el cliente". El camino crítico, cuya duración es de 382 horas, se muestra resaltado en color rojo. El camino semicrítico se muestra resaltado en color azul. Mientras que el no crítico en se denota con una flecha puntuada.

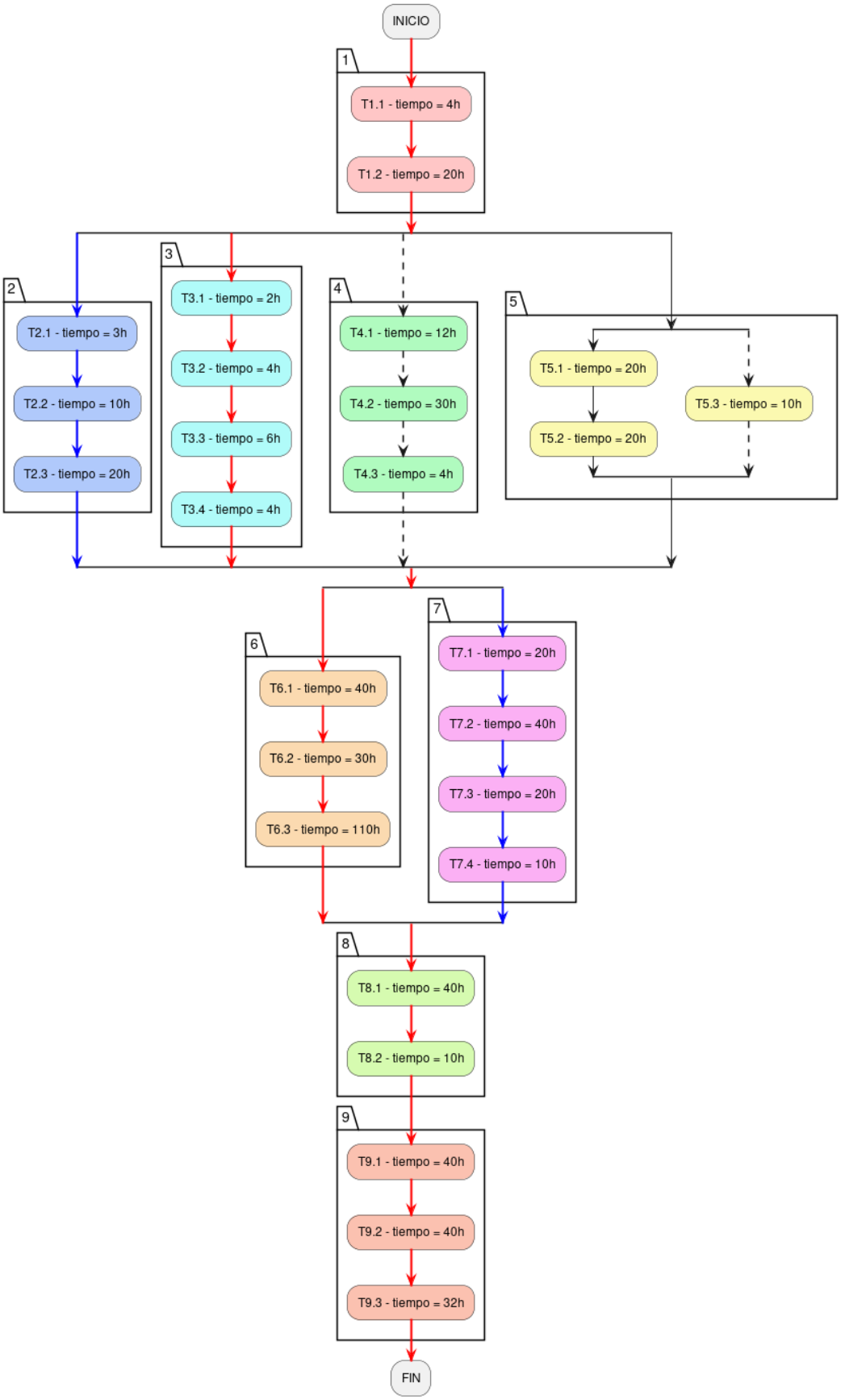

Figura 2. Diagrama de *Activity on Node*.

11. Diagrama de Gantt

Debido a que todas las tareas serán realizadas por el responsable del proyecto (es decir, una sola persona) las dependencias de las actividades del diagrama de *Activity on Node* fueron adaptadas para reflejar la imposibilidad de ejecutar tareas en paralelo.

El diagrama de Gantt se muestra separado en dos partes. Un cuadro 1 que contiene el desglose de tareas y la figura 3 que contiene el diagrama propiamente dicho.

WBS	Nombre	Inicio	Fin
1.1	Definición de requerimientos con el cliente	01/08/23	01/08/23
1.2	Planificación	03/08/23	12/08/23
2.1	Recopilación de documentación online	13/08/23	14/08/23
2.2	Lectura y comprensión de los documentos	15/08/23	19/08/23
2.3	Capacitación sobre el conjunto de instrucciones del procesador	20/08/23	29/08/23
3.1	Creación del repositorio de desarrollo	30/08/23	30/08/23
3.2	Creación del ambiente de CI/CD para tests	31/08/23	01/09/23
3.3	Desarrollo del ambiente CI/CD para documentación	02/09/23	04/09/23
3.4	Creación del ambiente de CI/CD para empaquetado	05/09/23	06/09/23
4.1	Revisión y análisis de código existente	07/09/23	12/09/23
4.2	Selección del código a reutilizar e importación	13/09/23	25/09/23
4.3	Refactorización del código	27/09/23	28/09/23
5.1	Lectura de documentación	29/09/23	08/10/23
5.2	Internalización con su API	09/10/23	18/10/23
5.3	Pruebas de límites y alcances del framework	19/10/23	23/10/23
6.1	Desarrollo de modelos de periféricos	24/10/23	11/11/23
6.2	Desarrollo de modelos de memorias y registros	12/11/23	25/11/23
6.3	Desarrollo de instrucciones	26/11/23	16/01/24
7.1	Pruebas unitarias de modelos	17/01/24	26/01/24
7.2	Pruebas unitarias de instrucciones	27/01/24	14/02/24
7.3	Pruebas de API	15/02/24	24/02/24
7.4	Corrección de errores	25/02/24	29/02/24
8.1	Creación del manual de usuario	01/03/24	19/03/24
8.2	Documentación de API en Doxygen	20/03/24	24/03/24
9.1	Preparación de las memorias	25/03/24	12/04/24
9.2	Redacción de las memorias	13/04/24	01/05/24
9.3	Presentación del trabajo	02/05/24	16/05/24

Cuadro 1. Desglose de tareas.

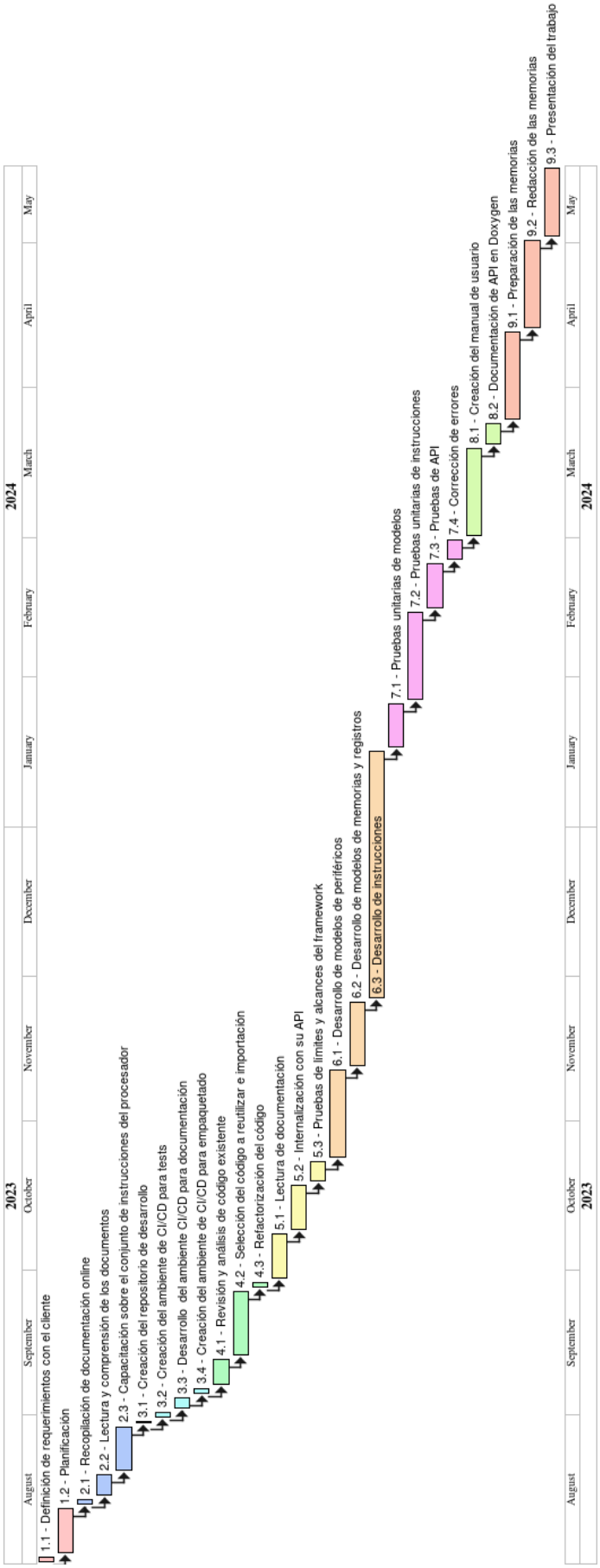

Figura 3. Diagrama de Gantt.

12. Presupuesto detallado del proyecto

A continuación se presenta el presupuesto detallado del proyecto expresado en dólares americanos:

COSTOS DIRECTOS			
Descripción	Cantidad	Valor unitario	Valor total
Horas del responsable	621	US$30	US$18.630
Computadora	1	US$600	US$600
Gastos operativos. (oficina, servicios, etc.)	10	US$5	US$50
SUBTOTAL			US$19.280
COSTOS INDIRECTOS			
Descripción	Cantidad	Valor unitario	Valor total
Modelo de referencia (físico o emulado)	1	X (*)	X (*)
SUBTOTAL			-
TOTAL			US$19.280

(*) Valor confidencial.

13. Gestión de riesgos

a) Identificación de los riesgos y estimación de sus consecuencias:

A continuación se detallan cinco posibles riesgos inherentes al proyecto. Los mismos son evaluados según su grado de severidad y su probabilidad de ocurrrencia tomando valores del 1 (bajo) al 10 (alto).

Riesgo 1: no finalizar el proyecto en el plazo de tiempo establecido.

- Severidad (S): 10. Severidad alta porque no se cumpliría con la fecha de entrega límite establecida.
- Ocurrecia (O): 6. Es probable ya que la totalidad del proyecto se realizará fuera del horario laboral.

Riesgo 2: dificultad en la capacitación en el *framework* LLVM.

- Severidad (S): 8. Severidad media/alta porque se planea tomar varias funcionalidad de dicho *framework*, que de otra manera implicarían un esfuerzo mucho mayor.
- Ocurrecia (O): 5. Es probable que suceda ya que es un proyecto de alta complejidad, pero al mismo tiempo tiene mucha documentación asociada.

Riesgo 3: bajo desempeño del emulador, con ejecuciones que no alcancen al tiempo real.

- Severidad (S): 3. Baja severidad, ya que se plantea un prototipo sin funcionanlidad completa, que luego podría ser optimizado para cumplir los requerimientos de performance.
- Ocurrecia (O): 6. Es probable que suceda ya que se trata de un *software* de alta complejidad computacional.

Riesgo 4: dificultad en la comprensión de documentos, tanto de *datasheets* como de manuales de arquitectura del procesador.

- Severidad (S): 10. Alta severidad, ya que sin entender dichos documentos no es posible desarrollar software que cumpla el correcto funcionamiento del procesador.
- Ocurrecia (O): 8. Se estima una alta probabilidad de ocurrencia debido a que son documentos sumamente extensos y de gran detalle técnico.

Riesgo 5: dificultad para localizar y depurar errores de desarrollo del *software*.

- Severidad (S): 4. Media severidad, ya que al estár internalizado con el código fuente, todas las fuentes de errores son facilmente identificables.
- Ocurrecia (O): 8. Media/alta ocurrencia, ya que es normal cometer errores en la programación de código.

b) Tabla de gestión de riesgos: (El RPN se calcula como RPN = S × O)

Riesgo	S	O	RPN	S*	O*	RPN*
1. No finalizar el proyecto en el plazo de tiempo establecido.	10	6	60	10	4	40
2. Dificultad en la capacitación en el *framework* LLVM.	8	5	40	-	-	-
3. Bajo desempeño del emulador, con ejecuciones que no alcancen al tiempo real.	3	6	18	-	-	-
4. Dificultad en la comprensión de documentos, tanto de *datasheets* como de manuales de arquitectura del procesador.	10	8	80	7	7	49
5. Dificultad para localizar y depurar errores de desarrollo del *software*.	4	8	32	-	-	-

Criterio adoptado: se tomarán medidas de mitigación en los riesgos cuyos números de RPN sean mayores a 50.

Nota: los valores marcados con (*) en la tabla corresponden luego de haber aplicado la mitigación.

c) Plan de mitigación de los riesgos que orifginalmente excedían el RPN máximo establecido.

Riesgo 1: se asignarán horas fijas semanales para garantizar que el proyecto se lleve a cabo en el tiempo estipulado.

- Severidad (S*): 10. Se mantiene la misma severidad que antes de la mitigación.
- Ocurrecia (O*): 4. Baja significativamente la probabilidad de ocurrencia de este riesgo ya que se tratará de ser riguroso con el seguimiento y control del plan de proyecto estipulado.

Riesgo 4: se utilizará el modelo de referencia provisto por la empresa para despejar dudas sobre la documentación respecto al funcionamiento esperado.

- Severidad (S*): 7. Se disminuye la severidad, ya que la documentación deja de ser el único punto de referencia para establecer el correcto funcionamiento del procesador.
- Ocurrecia (O*): 7. Reduce la ocurrencia levemente porque se chequearán ambas fuentes de verdad.

14. Gestión de la calidad

- Req #1: el emulador deberá ejecutar los mismos binarios que se utilizan en el hardware real.
  - Verificación: tomar un software conocido que haya sido ejecutado en el hardware real.
  - Validación: chequear que el emulador ejecute dicho software y finalize sin errores.
- Req #2: el sistema debe ser compatible con el sistema operativo Linux.
  - Verificación: no aplica.
  - Validación: sobre un entorno Linux chequear que el emulador se ejecute.
- Req #3: el emulador deberá poder ejecutar correctamente parte del set de instrucciones del procesador real.
  - Verificación: tomar algún binario que cuya ejecución deje la memoria en un estado conocido.
  - Validación: se la ejecución deje la memoria emulada en dicho estado.
- Req #4: se deberán desarrollar tests unitarios que verifiquen el set de instrucciones desarrollado.
  - Verificación: no aplica.
  - Validación: se podrán compilar y ejecutar dichos tests para verificar su funcionamiento.
- Req #5: se deberá desarrollar un ambiente de automatización de pruebas en Gitlab CI.
  - Verificación: no aplica.
  - Validación: se entregarán reporte de tests ejecutados por el CI.
- Req #6: el software podrá utilizarse como biblioteca compartida.
  - Verificación: se entregarán los *headers* y librería compartida para poder extender la funcionalidad y/o integrarla en otros ambientes.
  - Validación: inspeccionar el entregable.
- Req #7: la API expuesta deberá estar documentada con Doxygen.
  - Verificación: no aplica.
  - Validación: el cliente tendrá acceso a dicha documentación.
- Req #8: se realizará un manual de usuario que describa los funcionamientos clave del software.
  - Verificación: no aplica.
  - Validación: inspección del manual.
- Req #10: la API expuesta deberá ser en C.
  - Verificación: no aplica.
  - Validación: el cliente tendrá acceso al código fuente de la API.

15. Procesos de cierre

Al finalizar el proyecto se realizará una reunión final de evaluación del proyecto que contemplará las siguientes actividades:

- Pautas de trabajo que se seguirán para analizar si se respetó el plan de proyecto original:
  - Nicolás Iriarte comparará junto al director, los tiempos establecidos en el plan de proyecto con los tiempos que se obtuvieron en la realización del proyecto.
  - Nicolás Iriarte junto al director verificarán que se hayan cumplido la totalidad de los requerimientos solicitados por el cliente.
- Identificación de las técnicas y procedimientos útiles e inútiles que se emplearon, y los problemas que surgieron y cómo se solucionaron:
  - Nicolás Iriarte dejará registro de todos los procedimientos utilizados a lo largo del proyecto, e identificará cuáles fueron los más útiles y eficientes, así como también, los que no lo fueron.
  - Nicolás Iriarte dejará registro de todos los problemas que hayan surgido durante la realización del proyecto y cómo se solucionaron.
- Indicar quién organizará el acto de agradecimiento a todos los interesados, y en especial al equipo de trabajo y colaboradores:
  - Nicolás Iriarte presentará la defensa ante el jurado evaluador, en donde agradecerá al director, por su colaboración, tiempo, ayuda y compromiso con el proyecto; al cliente, por su confianza y por ofrecer la oportunidad de poder desarrollar el proyecto; y al jurado allí presente por su tiempo y disposición.