

Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Programa de Licenciatura en Ingeniería Electrónica



**Desarrollo de un conjunto de flujos de trabajo
para la implementación de software a bordo de
computadoras de guía, navegación y control espacial**

Informe de Trabajo Final de Graduación para optar por el título de
Ingeniero en Electrónica con el grado académico de Licenciatura

David Duarte Sánchez

Borrador de 31 de octubre de 2024

El documento **Requisitos para la entrega de Trabajos Finales de Graduación** a las bibliotecas del TEC indica que usted debe incluir la licencia de Creative Commons en la página siguiente de la portada.

Asegúrese entonces de **elegir la licencia correcta**, y ajustar el texto abajo a su selección.

Es necesario que **descargue el ícono** correcto en formato vectorial, y lo coloque en el directorio **fig/**.



Este trabajo titulado *Desarrollo de un conjunto de flujos de trabajo para la implementación de software a bordo de computadoras de guía, navegación y control espacial* por David Duarte Sánchez, se encuentra bajo la Licencia Creative Commons **Atribución-ShareAlike 4.0 International**.

Para ver una copia de esta Licencia, visite <http://creativecommons.org/licenses/by-sa/4.0/>.

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

David Duarte Sánchez

Cartago, 31 de octubre de 2024

Céd: 3-0507-0982

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Trabajo Final de Graduación
Acta de Aprobación

Defensa de Trabajo Final de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura

El Tribunal Evaluador aprueba la defensa del trabajo final de graduación denominado *Desarrollo de un conjunto de flujos de trabajo para la implementación de software a bordo de computadoras de guía, navegación y control espacial*, realizado por el señor David Duarte Sánchez y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

Dr. Alfonso Chaves Jiménez
Profesor Lector

Ing. William Marín Moreno
Profesor Lector

Dr. Johan Carvajal Godínez
Profesor Asesor

Cartago, 31 de octubre de 2024

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Trabajo Final de Graduación
Tribunal Evaluador
Acta de Evaluación

Defensa del Trabajo Final de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura

Estudiante: **David Duarte Sánchez** Carné: 2017239606

Nombre del proyecto: *Desarrollo de un conjunto de flujos de trabajo para la implementación de software a bordo de computadoras de guía, navegación y control espacial*

Los miembros de este Tribunal hacen constar que este trabajo final de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica y es merecedor de la siguiente calificación:

Nota del Trabajo Final de Graduación: _____

Miembros del Tribunal Evaluador

Dr. Alfonso Chaves Jiménez
Profesor Lector

Ing. William Marín Moreno
Profesor Lector

Dr. Johan Carvajal Godímez
Profesor Asesor

Cartago, 31 de octubre de 2024

Resumen

El resumen es la síntesis de lo que aparece en el resto del documento. Tiene que ser lo suficientemente conciso y claro para que alguien que lo lea sepa qué esperar del resto del trabajo, y se motive para leerla completamente. Usualmente resume lo más relevante de la introducción y contiene la conclusión más importante del trabajo.

Es usual agregar palabras clave, que son los temas principales tratados en el documento. El resumen queda fuera de la numeración del resto de secciones.

Evite utilizar referencias bibliográficas, tablas, o figuras en el resumen.

Palabras clave: GNC, Sistemas, procesadores embebidos, marcos de trabajo, model to model transofrmation, codigo embebido

Abstract

Same content as the Spanish version, just in English. Check [this site](#) for some help with the translation. For instance, the following is the automatic translation from a previous version of the “Resumen”.

The abstract is the synthesis of what appears in the rest of the document. It has to be concise and clear enough so that someone reading it knows what to expect from the rest of the text, and is motivated to read it in full. It usually summarizes the most relevant parts of the introduction and contains the most important conclusion of the work.

It is usual to add keywords, which are the main topics covered in the document. The abstract is left out of the numbering of the rest of the sections.

Avoid using bibliographical references, tables, or figures in the abstract.

Keywords: word 1, word 2,

a mis queridos padres

Agradecimientos

El resultado de este trabajo no hubiese sido posible sin el apoyo de Thevenin, Norton, Einstein y mi querido amigo Ohm.

Usualmente se agradece aquí a la empresa o investigador que dio la oportunidad de realizar el trabajo final de graduación.

No debe confundir el agradecimiento con la dedicatoria. La dedicatoria es usualmente una sola línea, con la persona a quien se dedica el trabajo.

El agradecimiento es un texto más elaborado, de carácter personal, en donde se expresa la gratitud por la oportunidad, el apoyo brindado, la inspiración ofrecida, el acompañamiento moral, etc.

David Duarte Sánchez

Cartago, 31 de octubre de 2024

Índice general

Índice de figuras	IV
Índice de tablas	VI
Revisar	VIII
1. Introducción	1
1.1. Proceso de diseño de los sistemas de Guía, Navegación y Control espacial .	1
1.1.1. Requerimientos de los sistemas	1
1.2. Sistemas embebidos para los sistemas GNC	2
1.2.1. Marco de Trabajo Yocto Project	2
1.3. Donde se ubica dentro del flujo de control	3
1.4. Hardware en el loop	3
1.5. Objetivos y estructura del documento	3
2. Marco teórico	4
2.1. Estimación	4
2.2. Control	5
2.3. Procesadores embebidos	5
2.3.1. Cortex-A9	5
2.3.2. Tarjeta de desarrollo ZedBoard	6
2.4. Marcos de trabajo	7
2.4.1. YOCTO	8
2.5. MATLAB	9
2.5.1. Simulink	9
2.6. Transformación de modelo a modelo	10
2.6.1. MATLAB Embedded Coder	10
2.6.2. MATLAB Simulink Coder	10
2.7. Código embebido	11
2.8. Compilación Cruzada	12
2.9. Contenedores	13
2.10. Protocolos de Comunicación	13
2.10.1. UART	13
2.10.2. SSH	14

2.11. Computadoras de guía, navegación y control	14
2.11.1. EXA ICEPS	15
2.12. Revisión literaria	15
2.12.1. Desarrollo de sistemas de navegación	16
2.12.2. Transformación de Lenguaje de Bloques a Código C	16
2.13. Avances recientes en GNCs	17
2.13.1. Programación de Sistemas GNC	17
3. Plataforma de desarrollo	19
3.1. Selección de la tarjeta de desarrollo	19
3.1.1. Requerimientos de la aplicación	19
3.1.2. Tarjetas candidatas	20
3.1.3. Criterios de comparación	22
3.2. Matriz de Pugh	24
3.3. Plataforma seleccionada	24
3.3.1. Especificaciones principales	24
3.4. Reflexión final	26
4. Síntesis de software para GNC embebido	27
4.1. Creación de los entornos de desarrollo	28
4.1.1. MATLAB	28
4.1.2. Contenedores	31
4.2. Caso de estudio 1 - Filtro Básico en MATLAB	37
4.2.1. Simulación del caso de estudio en MATLAB Simulink	41
4.3. Flujo de trabajo de la aplicación de transformación de modelo a modelo	42
4.3.1. Simulink Coder	44
4.3.2. Definición de parámetros	44
4.3.3. Compilación del Código C generado	47
4.4. Flujo de Trabajo EmbedSynthGNC	48
4.4.1. Creación de una capa de yocto	48
4.4.2. Caso de estudio 1 - Filtro Básico en ZedBoard	50
4.4.3. Integración del programa generado a la capa de Yocto	50
4.4.4. Generación de la imagen	51
4.4.5. Implementación de la imagen mínima en la tarjeta de desarrollo Zedboard	51
4.4.6. Conexión de la tarjeta de desarrollo con el computador host	52
4.4.7. Ejecución del caso de estudio y resultados	53
4.4.8. Comparación de resultados	54
4.5. Reflexión final	55
5. Caso de estudio IMU y PID	57
5.1. Caso de estudio 2 - IMU	57
5.1.1. Implementación en MATLAB Simulink	57
5.1.2. Bloques utilizados para la implementación	58

5.1.3.	Resultados de la simulación	61
5.1.4.	Implementación en la Tarjeta de desarrollo mediante EmbedSynthGNC	61
5.1.5.	Resultados de la implementación	64
5.2.	Caso de estudio 3 - PID	64
5.2.1.	Implementación en MATLAB Simulink	64
5.2.2.	Bloques utilizados para la implementación	64
5.2.3.	Implementación en la Tarjeta de desarrollo mediante EmbedSynthGNC	65
6.	Conclusiones	66
Bibliografía		67
A.	Métricas de comparación de señales	70

Índice de figuras

2.1. Tarjeta de desarrollo Zedboard	6
2.2. Flujo de trabajo Yocto Project	8
2.3. Flujo de trabajo Simulink Coder	11
2.4. Diagrama de compilación cruzada	12
4.1. Diagrama general del flujo de trabajo propuesto	27
4.2. Diagrama para generar el entorno de MATLAB	29
4.3. Menú de instalación de Python	29
4.4. Menú de instalación de Python	29
4.5. Menú de instalación de VS Code	30
4.6. Menú de instalación de VS Code	30
4.7. Retorno del comando 4.4	32
4.8. Retorno del comando 4.5	32
4.9. Retorno del comando 4.6	32
4.10. Diagrama para la elaboración del entorno para compilación cruzada	33
4.11. Diagrama para la elaboración del entorno para Yocto Project	34
4.12. Diagrama caso de estudio 1 - Filtro Básico	37
4.13. Librería de bloques	37
4.14. Librería de bloques - generador de onda senoidal	38
4.15. Configuración de los generadores de onda senoidal	38
4.16. Librería de bloques - sumador de señales	39
4.17. Configuración del bloque sumador de señales	39
4.18. Bloque función de transferencia	40
4.19. Librería de bloques - gráfico	41
4.20. Diagrama MATLAB Simulink para poder observar las salidas	41
4.21. Salida simulada del diagrama mostrado en la Figura 4.20	42
4.22. Flujo de trabajo MATLAB Simulink Coder	43
4.23. Pestaña Aplicaciones	44
4.24. Pestaña código C	44
4.25. Configuración de parámetros	44
4.26. Selección del procesador y la familia del procesador	45
4.27. Selección del tipo de archivo de construcción	46
4.28. Archivo comprimido en el directorio swap_area	46
4.29. Make File	47

4.30. Binario llamado simple_filter	47
4.31. Flujo de trabajo EmbedSynthGNC	49
4.32. Árbol de directorios de la capa	50
4.33. Estructura del archivo caso_de_estudio_1.bb	51
4.34. Puertos tarjeta de desarrollo Zedboard	53
4.35. Salida resultante de la imagen generada mediante el flujo de trabajo	54
4.36. Comparación de la salida resultante simulada (izquierda) y experimental (derecha)	54
5.1. Diagrama completo del caso de estudio 2 - IMU [30]	58
5.2. Bloque para la aceleración lineal	59
5.3. Bloque para la velocidad angular	59
5.4. Bloque para la simulación del comportamiento de la IMU	60
5.5. Bloque para la integración de la velocidad angular	61
5.6. Bloque para la lectura de archivos	61
5.7. Bloque para la suma de señales	62
5.8. Bloque para la simulación del comportamiento de la IMU	62
5.9. Bloque para convertir de Radianes a grados	63
5.10. Bloque para aplicar una función implementada mediante código	63
5.11. Diagrama completo del caso de estudio 3 - PID	65

Índice de tablas

2.1. Puertos de entrada y salida de la plataforma de desarrollo Zedbord	7
2.2. Especificaciones generales de la tarjeta de desarrollo ZeadBoard	7
3.1. Matriz de Pugh para seleccionar la tarjeta de desarrollo que mejor de adapte a los requerimientos del proyecto	24

Listings

4.1. Instalacion de docker, Linux	31
4.2. Instalacion de Ubuntu 20.04, Linux	31
4.3. Instalacion de Ubuntu 16.04, Linux	31
4.4. Lista de contenedores del sistema, Docker	32
4.5. Iniciar un contenedor, Docker	32
4.6. Ingresar a un contenedor, Docker	32
4.7. Instalacion del compilador cruzado, Contenedor	33
4.8. Instalacion de CMake, Contenedor	33
4.9. Instalacion de build essential, Contenedor	33
4.10. Generacion de usuario no root, Linux	34
4.11. Iniciar usuario no root, Linux	35
4.12. Requerimientos Yocto Zeus, Linux	35
4.13. Version de Yocto	35
4.14. BSP para Zedboard	35
4.15. Configuraciones adicionales, Yocto	36
4.16. Copiar archivos al contenedor, Linux	47
4.17. Copiar archivos del contenedor, Linux	47
4.18. Compilacion del programa, Linux	47
4.19. "Print Working Directory",Linux	48
4.20. Inicializar ambiente, Yocto	48
4.21. Generar nueva capa, Yocto	48
4.22. Agregar nueva capa, Yocto	48
4.23. Generar archivos de desarrollador, Yocto	51
4.24. Instalar la capa generada, Yocto	51
4.25. Copiar archivos root, Linux	52
4.26. Copiar sistema de archivos, Linux	52
4.27. Copiar archivo por protocolo SSH, Linux	53
A.1. Métricas de comparación de señales	70

Revisar

Capítulo 1

Introducción

1.1. Proceso de diseño de los sistemas de Guía, Navegación y Control espacial

La implementación de sistemas GNC en sistemas embebidos, conlleva una combinación de hardware y software especializado, por un lado, los microcontroladores son los encargados de gestionar los cálculos, mientras que los sensores proporcionan distintos tipos de datos por medio de las entradas. Por otro lado, los sistemas en tiempo real garantizan la respuesta en el momento requerido. Las aplicaciones de estos se pueden observar en drones, satélites y sondas espaciales [1].

Como se mencionó anteriormente los sistemas GNC son fundamentales en las misiones espaciales, están encargados de determinar la trayectoria óptima para cumplir los objetivos de la misión, además de calcular la secuencia de maniobras necesarias, determinar la posición, velocidad y orientación, también se encargan de aplicar las acciones correctivas necesarias para mantener la trayectoria [2].

1.1.1. Requerimientos de los sistemas

Los requerimientos de los sistemas GNC incluyen: precisión para determinar la posición y orientación del vehículo con gran exactitud, robustez para funcionar de manera confiable y tolerar fallos o perturbaciones generadas por el entorno, autonomía para poder operar sin depender de la intervención humana, flexibilidad para adaptarse a diferentes fases de la misión y un bajo consumo de potencia para minimizar el uso de los recursos limitados a bordo. Para cumplir con los requerimientos mencionados anteriormente se debe definir con precisión los requerimientos del sistema, como la precisión necesaria para determinar la posición del vehículo, la robustez del sistema para resistir fallos, las restricciones energéticas y de recursos computacionales a bordo. Una vez solventados estos requerimientos el sistema se plantea bajo una arquitectura modular la cual divide el sistema en bloques

independientes para las funciones de guía, navegación y control, facilitando el desarrollo, prueba y mantenimiento [3].

1.2. Sistemas embebidos para los sistemas GNC

El uso de sistemas embebidos ha transformado la navegación y el control aeroespacial. Estos sistemas integran hardware y software, permitiendo el procesamiento de datos en tiempo real, fundamental para la navegación precisa y el control de vuelo. Los sistemas embebidos gestionan sensores que recopilan información sobre altitud, velocidad y posición, permitiendo a pilotos y sistemas automáticos tomar decisiones rápidas y fundamentadas. Esta capacidad de respuesta es esencial en entornos cambiantes, como la aviación o el lanzamiento de cohetes. [4]

Además, los sistemas embebidos facilitan la integración de múltiples funciones en un solo dispositivo, reduciendo el peso y volumen de los equipos a bordo, un factor crucial en la industria aeroespacial. Por ejemplo, en los sistemas de control de vuelo, los microcontroladores y procesadores embebidos pueden gestionar desde la navegación hasta la comunicación y el monitoreo de sistemas críticos, todo desde una única unidad. Esta integración mejora la eficiencia del espacio y minimiza la posibilidad de fallos al reducir el número de componentes individuales que podrían fallar. [5]

Finalmente, la implementación de sistemas embebidos ha permitido avances significativos en la automatización y la inteligencia artificial aeroespacial. Los algoritmos embebidos procesan datos de manera eficiente, permitiendo la navegación autónoma y el control de vehículos sin intervención humana, especialmente relevante en misiones espaciales con comunicación limitada. Los sistemas embebidos mejoran la seguridad y eficiencia de las operaciones aeroespaciales, abriendo nuevas posibilidades para la exploración y el desarrollo de tecnologías futuras en este campo. [6]

1.2.1. Marco de Trabajo Yocto Project

Yocto Project es una iniciativa de código abierto que proporciona un conjunto de herramientas y recursos para crear sistemas operativos Linux personalizados, especialmente diseñados para dispositivos embebidos. Su objetivo principal es facilitar el desarrollo de software y la integración de componentes en una amplia variedad de hardware, permitiendo a los desarrolladores construir imágenes de sistema adaptadas a sus necesidades específicas. Utiliza BitBake, una herramienta que permite definir recetas para la construcción y empaquetado de software, lo que otorga gran flexibilidad y personalización. [7]

Además, soporta múltiples arquitecturas de hardware, como ARM, x86, MIPS y PowerPC, lo que lo hace adecuado para diferentes dispositivos, desde microcontroladores hasta sistemas más complejos. Al fomentar la reutilización de componentes y contar con

una comunidad activa que contribuye con mejoras y documentación, el Yocto Project se convierte en una solución ideal para el desarrollo de sistemas operativos en aplicaciones de IoT, electrónica de consumo y automatización industrial. [8]

1.3. Donde se ubica dentro del flujo de control

Diagrama profe Johan (ver en proxima reunion)

1.4. Hardware en el loop

El Hardware en el loop es una técnica fundamental en el desarrollo de sistemas GNC, ya que, permite simular el comportamiento del hardware en tiempo real, facilitando para los desarrolladores la prueba y validación del software sin requerir el hardware físico, es esta forma permite probar de forma exhaustiva el software asegurando el funcionamiento del hardware simulado y permite la validación de todo el sistema antes de su implementación final. Esta implementación genera una mayor precisión en las pruebas, la posibilidad de validar la autonomía del sistema, además de reducir significativamente el tiempo y los costos de implementación y prueba del hardware. En resumen, es una técnica esencial en el desarrollo de sistemas GNC espaciales, permitiendo una validación integral y eficiente de estos complejos sistemas antes de su despliegue en misiones reales [9] [10].

1.5. Objetivos y estructura del documento

El objetivo principal de este proyecto es desarrollar un conjunto de flujos de trabajo para la implementación de software a bordo de computadoras de guía, navegación y control espacial. Para lograr este objetivo se persiguen tres objetivos específicos. El primero consiste en Identificar una plataforma de hardware para el desarrollo de un modelo de ingeniería de una computadora de navegación espacial.

El segundo se encarga de Establecer flujos de trabajo para el prototipado de algoritmos de control de orientación y navegación para aplicaciones espaciales con hardware en el loop. Y por último el tercero consiste en Evaluar los casos de uso de una computadora de navegación y control espacial mediante la implementación de una aplicación de referencia demostrativa (caso de la IMU).

Este documento incluye lo siguiente: en el capítulo 2 se presenta el marco teórico, donde se esbozan los fundamentos de la propuesta realizada. En el capítulo 3 se detalla la solución propuesta y el modelo implementado para la solución del problema. En el capítulo 4 se presentan los resultados obtenidos. Por último, el capítulo 5 se presenta las conclusiones de la investigación y trabajo realizado, así como recomendaciones y trabajo a futuro por desarrollar.

Capítulo 2

Marco teórico

En este capítulo se presentan los conceptos teóricos que subyacen la propuesta de desarrollo de un conjunto de flujos de trabajo para la implementación de software a bordo de computadoras de guía, navegación y control espacial. La información expuesta se deriva tanto de conocimientos propios como información bibliográfica.

2.1. Estimación

La estimación implica el uso de modelos matemáticos y algoritmos para calcular las variables de estado del sistema. Estas variables son esenciales para comprender el comportamiento del sistema y para tomar decisiones informadas sobre su control. La estimación puede realizarse de dos maneras:

- Lazo abierto: En este enfoque, se utilizan modelos de estimación predefinidos sin retroalimentación, lo que significa que las estimaciones no se ajustan en función de las mediciones reales.
- Lazo cerrado: Este método ajusta las estimaciones en función de las mediciones reales y las salidas del sistema, lo que permite una mayor precisión y adaptabilidad.

Esta es crucial en aplicaciones donde las mediciones directas son difíciles o costosas de obtener, por ejemplo en los sistemas hidráulicos, la estimación de variables de estado permite optimizar el rendimiento y la eficiencia del sistema, asegurando que se mantengan las condiciones deseadas a pesar de las perturbaciones externas o errores en las mediciones [11]. La estimación es un componente clave en los sistemas de control, ya que facilita la comprensión y el manejo de sistemas complejos. Su implementación permite una operación más eficiente y efectiva, mejorando su capacidad de respuesta ante diversas condiciones operativas [12].

2.2. Control

Como se mencionó anteriormente la estimación es un componente clave en los sistemas de control, ya que este se enfoca en el desarrollo y diseño de sistemas capaces de regular y controlar variables de un proceso de manera autónoma. Estos sistemas utilizan sensores, actuadores y algoritmos de control para mantener las variables de interés dentro de los rangos permitidos, mejorando de esta forma la eficiencia, precisión y confiabilidad de los procesos. Su aplicación abarca desde sistemas espaciales hasta biorreactores y sistemas de iluminación.

2.3. Procesadores embebidos

Los procesadores embebidos son microprocesadores especializados en tareas dentro de un sistema más complejo. A diferencia de los procesadores de propósito general, estos están optimizados para ofrecer eficiencia energética, un tamaño compacto y costo reducido. Algunas de las características de los procesadores embebidos se presentan a continuación:

- Integración de periféricos: Incorporan periféricos específicos de la aplicación en un único chip, incluyendo temporizadores, puertos de entrada/salida y controladores de memoria.
- Arquitecturas de bajo Consumo: Diseñados para maximizar la duración de la batería en dispositivos portátiles, lo que es esencial para la operatividad de dispositivos móviles.
- Tamaño compacto: Su diseño permite reducir costos y facilitar la integración en espacios limitados, lo que los hace ideales para aplicaciones donde el espacio es crítico.
- Capacidad de respuesta en tiempo real: Pueden responder a eventos externos de manera predecible y determinista, lo que es crucial en aplicaciones que requieren una respuesta rápida y precisa.

2.3.1. Cortex-A9

Los procesadores embebidos basados en la arquitectura ARM Cortex-A9 se utilizan en aplicaciones de alto rendimiento y capacidades avanzadas de procesamiento. Aunque esta arquitectura no es un procesador embebido, sino más bien una familia de núcleos de procesador diseñado por ARM Holdings, los SoC que incorporan estos núcleos han demostrado ser una solución popular para aplicaciones embebidas [13]. Algunas de sus características son :

- Arquitectura de 32 bits basada en ARMv7-A.
- Alto rendimiento adecuado para aplicaciones exigentes como sistemas operativos embebidos, procesamiento multimedia y gráficos.
- Características avanzadas como unidades de coma flotante, unidades de procesamiento NEON para procesamiento multimedia y soporte para virtualización.

Algunos SoC que incorporan núcleos Cortex-A9 son:

- Nvidia Tegra 3: Combina cuatro núcleos Cortex-A9 y una GPU.
- Texas Instruments OMAP 4: Familia de SoC que combina núcleos Cortex-A9 y DSP.
- Xilinx Zynq-7000: Integra núcleos Cortex-A9 con lógica programable FPGA.

2.3.2. Tarjeta de desarrollo ZedBoard

La ZedBoard es una tarjeta de desarrollo basada en el Xilinx Zynq-7000 que como se mencionó anteriormente integra núcleos Cortex-A9 con la lógica programable para Field Programmable Gate Array, por sus siglas en inglés (FPGA). Esta plataforma es ideal para prototipar aplicaciones en el ámbito de sistemas embebidos. La tabla 2.2 resume las especificaciones que posee la tarjeta de desarrollo ZedBoard.

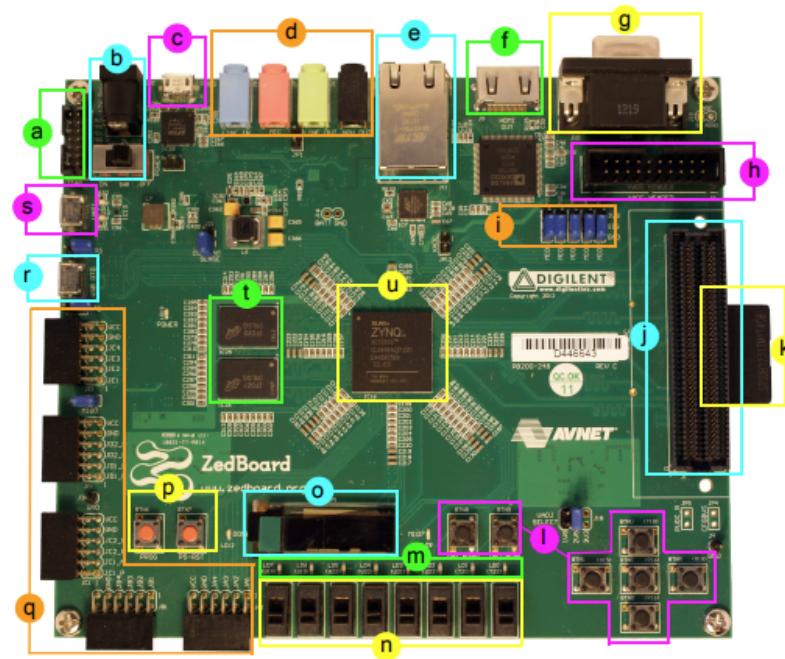


Figura 2.1: Tarjeta de desarrollo Zedboard

Como se pudo observar en 2.1, esta tarjeta de desarrollo cuenta con los puertos de conexión que se mencionan en la Tabla 2.1:

Tabla 2.1: Puertos de entrada y salida de la plataforma de desarrollo Zedbord

Identificador	Descripción	Identificador	Descripción
a	Conejero JTAG Xilinx	l	Pulsadores
b	Entrada de voltaje y interruptor de encendido	m	LEDs
c	USB-JTAG para programación	n	Interruptores
d	Puertos de audio	o	Pantalla OLED
e	Puerto de ethernet	p	Botones de programación y reinicio
f	Puerto HDMI (Salida de video)	q	Conejeros Pmod
g	Puerto VGA (Salida de video)	r	USB OTG para perifericos
h	Puerto XADC	s	USB UART
i	Jumpers de configuración	t	Memoria DDR3
j	Conejero FCM	u	Dispositivo Zynq
k	Entrada para tarjeta SD		

Tabla 2.2: Especificaciones generales de la tarjeta de desarrollo ZeadBoard

Especificación	Detalles
Procesador	Xilinx Zynq-7000 (XC7Z020)
Núcleos de Procesador	ARM Cortex-A9 de doble núcleo
Memoria DDR3	512 MB
Memoria Flash	256 MB QSPI
Almacenamiento	Tarjeta SD de 4 GB
Conectividad	Ethernet (10/100/1000 Mbps), USB OTG 2.0, USB-UART
Salidas de Video	HDMI (1080p), VGA de 8 bits, OLED 128x32
Audio	Códec de audio I2S
Puertos GPIO	54 pines GPIO
Interfaz de JTAG	Soporte para programación y depuración
Dimensiones	10.2 cm x 6.4 cm
Fuente de Alimentación	5V a través de conector de alimentación
Sistema Operativo	Soporte para Linux y otros sistemas embebidos
Expansión	Conejeros Pmod y FMC para módulos adicionales

Además de esto algunas especificaciones de la plataforma de desarrollo son mencionadas en 2.2. Por otro lado la ZedBoard es una plataforma de desarrollo altamente versátil que se destaca por su capacidad para ejecutar sistemas operativos como Linux, lo que la convierte en una opción ideal para proyectos de diseño y desarrollo de sistemas embebidos. Además de su compatibilidad con Linux, como se mencionó anteriormente la ZedBoard cuenta con especificaciones técnicas que la hacen destacar en el ámbito del desarrollo. Entre ellas se incluyen un procesador ARM Cortex-A9, una FPGA.

2.4. Marcos de trabajo

Los marcos de trabajo en sistemas embebidos son conjuntos de herramientas y bibliotecas que facilitan el desarrollo de aplicaciones en estos sistemas. Estos proporcionan

una estructura que permite abordar los desafíos específicos que presentan los sistemas embebidos.

Los sistemas embebidos interactúan con su entorno físico, lo que requiere un diseño que no solo considere los resultados de las operaciones, sino también el cumplimiento de plazos y restricciones específicas. En este contexto, las propiedades no funcionales, como el consumo energético, la latencia, la fiabilidad y el manejo de recursos, son críticas para el diseño y optimización del rendimiento general del sistema [14]. Los frameworks juegan un papel fundamental al proporcionar herramientas y bibliotecas predefinidas, permitiendo a los desarrolladores centrarse en la lógica de la aplicación en lugar de lidiar con los detalles de bajo nivel del hardware, lo que acelera el proceso de desarrollo y reduce la posibilidad de errores. Ejemplos de frameworks populares en sistemas embebidos incluyen Robot Operating System (ROS), utilizado en aplicaciones de robótica, y FreeRTOS, un sistema operativo de tiempo real diseñado para microcontroladores y sistemas embebidos [15].

2.4.1. YOCTO

Yocto es un marco de trabajo o bien del inglés (framework) popular utilizado en el desarrollo de sistemas embebidos, especialmente en la creación de distribuciones de Linux personalizadas para hardware específico. Yocto utiliza un proceso de construcción cruzada, lo que significa que el código se compila en una plataforma diferente a la que se ejecutará, permitiendo que el código se optimice para el hardware específico del sistema embebido [16].

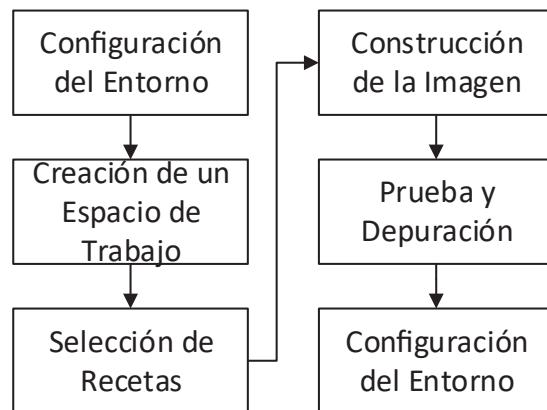


Figura 2.2: Flujo de trabajo Yocto Project

Una de las principales ventajas de Yocto es su flexibilidad en la configuración del sistema, permitiendo a los desarrolladores seleccionar paquetes específicos, configurar opciones de compilación y personalizar el sistema operativo según sus necesidades. Además, Yocto fomenta la reutilización de código a través de capas, que son colecciones de recetas, configuraciones y parches que se pueden agregar o eliminar fácilmente del flujo de trabajo de construcción [16].

2.5. MATLAB

MATLAB es un software de cálculo técnico desarrollado por MathWorks, ampliamente utilizado en diversas áreas de la ciencia y la ingeniería. Proporciona un entorno interactivo para el desarrollo de algoritmos, análisis de datos, visualización y cálculo numérico. Su facilidad para trabajar con vectores y matrices lo distingue de otros sistemas de cálculo [17]. Es comúnmente utilizado para simular sistemas eléctricos, como convertidores DC/DC, y realizar análisis numéricos en problemas complejos [18].

MATLAB juega un papel crucial en el desarrollo de sistemas de control aeroespaciales, facilitando la simulación, modelado y control de vehículos aéreos no tripulados (UAV) y otros sistemas relacionados. MATLAB, junto con Simulink, permite el modelado cinemático y dinámico de UAVs. Esto incluye la programación y control de sus movimientos, como cabeceo y guiñada, utilizando motores de corriente directa y encoders ópticos para medir su posición [19] [20].

2.5.1. Simulink

Como se mencionó anteriormente Simulink, es un entorno de simulación y diseño gráfico que forma parte del software MATLAB. Se utiliza principalmente para modelar, simular y analizar sistemas dinámicos, especialmente aquellos que involucran componentes eléctricos, mecánicos y de control, algunas de las características principales de Simulink son:

- Modelado Gráfico
- Simulación en Tiempo real
- Integración con MATLAB
- Diseño de Controladores
- Análisis de Sistemas Dinámicos

De esta forma podemos ver que simulink es una herramienta que permite a los usuarios crear modelos visuales de sistemas complejos utilizando bloques representativos, facilitando así su diseño y comprensión. Ofrece simulaciones en tiempo real, esenciales para evaluar el comportamiento de sistemas en ingeniería y control bajo diversas condiciones [17]. Su integración con MATLAB potencia las capacidades de análisis y programación, permitiendo un análisis más profundo y la personalización de simulaciones [21]. Simulink se aplica en diversas áreas como la ingeniería eléctrica, mecánica, robótica y diseño de sistemas de control, siendo especialmente útil para diseñar y probar controladores (como PID y Fuzzy), analizar sistemas dinámicos y desarrollar prototipos rápidos para sistemas embebidos [22].

2.6. Transformación de modelo a modelo

La transformación de modelo a modelo se refiere a un proceso en el que un modelo se convierte en otro, manteniendo la esencia de su estructura y funcionalidad, pero adaptándose a nuevas necesidades o contextos. Este concepto es fundamental en la Ingeniería de Software, especialmente dentro de la Arquitectura Dirigida por Modelos (MDA), donde se busca facilitar la interoperabilidad y la portabilidad de sistemas a través de la transformación de modelos independientes de la computación (CIM) a modelos independientes de la plataforma (PIM) y viceversa.

- Modelos de Datos a Modelos de Aplicación:
- Modelos de Negocio a Modelos de Implementación
- Modelos UML a Código Fuente

Para efectos de este trabajo el área de interés serán la transformación de UML a Código Fuente.

2.6.1. MATLAB Embedded Coder

El MATLAB Embedded Coder se adapta a esta definición de transformación de modelo a modelo, ya que permite a los usuarios generar código C y C++ a partir de modelos Simulink. Esto es especialmente útil en el desarrollo de sistemas embebidos, donde se requiere que los modelos de alto nivel se transformen en código que pueda ser ejecutado en hardware específico. Esta herramienta facilita la implementación de algoritmos y sistemas de control, asegurando que el modelo original se traduzca eficazmente en un formato que pueda ser utilizado en entornos de producción.

2.6.2. MATLAB Simulink Coder

Simulink Coder es una herramienta del entorno MATLAB/Simulink que permite generar automáticamente código C y C++ a partir de modelos gráficos, facilitando la implementación de algoritmos en hardware o software. Sus características incluyen la generación de código automática, integración con diversas plataformas de hardware, optimización del rendimiento y soporte para modelos complejos, lo que la hace ideal para aplicaciones en control de sistemas, simulación y pruebas, y desarrollo ágil. En resumen, Simulink Coder es esencial para ingenieros que desean transformar modelos teóricos en aplicaciones prácticas, mejorando tanto el proceso de desarrollo como el rendimiento del producto final.

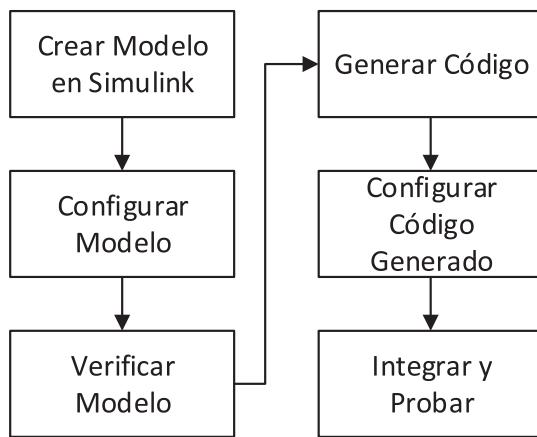


Figura 2.3: Flujo de trabajo Simulink Coder

Como se pudo observar en la Figura 2.3, primeramente se debe de diseñar el modelo utilizando bloques de Simulink para representar el sistema o algoritmo a implementar, una vez generado el diagrama se deben de ajustar las configuraciones del modelo, incluyendo parámetros como el tipo de solución, la frecuencia de muestreo y las opciones de simulación, además de esto se deben de realizar simulaciones para verificar que el modelo funcione correctamente y cumpla con los requisitos especificados.

Finalmente se debe de hacer uso de la herramienta Simulink Coder para generar automáticamente código C o C++ a partir del modelo validado. Adicionalmente a este código generado se pueden realizar configuraciones según se desea su ejecución, estas configuraciones son opciones adicionales para la generación del código, como optimización y estilo de codificación. Una vez aplicados todos los cambios necesarios se debe integrar el código generado en un entorno de desarrollo adecuado y realizar pruebas para asegurar que el código se comporta como se espera en el hardware objetivo.

2.7. Código embebido

El código embebido se refiere a un tipo de software diseñado para operar en dispositivos con recursos limitados, como microcontroladores y sistemas embebidos. Este código es fundamental en la programación de dispositivos electrónicos, permitiendo que estos realicen tareas específicas, como gestionar un sistema de automatización industrial o incluso operar en dispositivos móviles. Se caracteriza por su ejecución en dispositivos con recursos limitados, su capacidad para controlar dispositivos electrónicos, el uso de lenguajes de bajo nivel, la optimización de recursos y la necesidad de garantizar tiempos de respuesta determinísticos.

2.8. Compilación Cruzada

La compilación cruzada es un proceso clave en el desarrollo de software que permite compilar código fuente en un sistema operativo o arquitectura de hardware diferente al utilizado para el desarrollo. Este método es especialmente valioso en entornos como sistemas embebidos, donde la plataforma de destino no es adecuada para la compilación directa. Utiliza compiladores específicos, conocidos como compiladores cruzados, que generan código ejecutable para la plataforma de destino desde una plataforma de origen. Esto permite a los desarrolladores trabajar en sus máquinas locales, como Windows o Linux, mientras crean aplicaciones para dispositivos como microcontroladores o sistemas operativos variados.

Además de su uso en sistemas embebidos, la compilación cruzada es fundamental para el desarrollo multiplataforma, ya que facilita la creación de aplicaciones que funcionan en diferentes sistemas operativos sin necesidad de modificar el código base. Este enfoque no solo mejora la eficiencia del proceso de desarrollo y pruebas, sino que también evita la constante transferencia de código a la plataforma de destino. En resumen, la compilación cruzada es una técnica esencial en el desarrollo moderno de software, permitiendo a los desarrolladores abordar múltiples plataformas y arquitecturas con mayor facilidad.

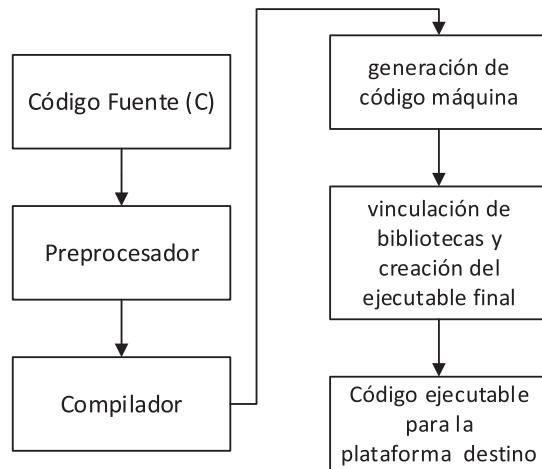


Figura 2.4: Diagrama de compilación cruzada

Como se pudo observar en la Figura 2.4, el proceso de compilación de un programa en lenguaje C comienza con el código fuente, que es el conjunto de instrucciones escritas por el programador. A continuación, se utiliza el preprocesador, que lleva a cabo tareas esenciales como la inclusión de archivos y la expansión de macros, generando un archivo intermedio que está listo para ser compilado. Posteriormente, el compilador analiza este archivo intermedio y produce código objeto, una representación en lenguaje máquina que aún no está completamente vinculada. Este código objeto es luego procesado por el ensamblador, que lo convierte en código máquina, específico para la arquitectura del sistema objetivo.

2.9. Contenedores

Docker ha revolucionado la forma en que se desarrollan y despliegan aplicaciones al ofrecer un entorno portátil y consistente. Gracias a su capacidad de empaquetar aplicaciones junto con todas sus dependencias, los desarrolladores pueden estar seguros de que su software funcionará de manera idéntica en cualquier entorno, ya sea local, en la nube o en producción. Esta portabilidad no solo ahorra tiempo en la configuración del entorno, sino que también reduce significativamente los problemas relacionados con "funciona en mi máquina". Además, el aislamiento que proporcionan los contenedores asegura que las aplicaciones operen sin interferencias, lo que es crucial para mantener la estabilidad y el rendimiento.

Por otro lado, la eficiencia de Docker es notable. A diferencia de las máquinas virtuales, los contenedores comparten el núcleo del sistema operativo, lo que permite un uso más optimizado de los recursos y un inicio casi instantáneo. Esto se traduce en una mayor agilidad y rapidez al escalar aplicaciones, ya que se pueden crear y gestionar múltiples instancias de contenedores con facilidad. La capacidad de versionar imágenes también es un gran beneficio, ya que permite a los equipos mantener un historial claro de cambios y revertir a versiones anteriores cuando sea necesario. En conjunto, estas características hacen de Docker una herramienta indispensable para la integración y entrega continua (CI/CD), mejorando significativamente los flujos de trabajo de desarrollo y despliegue.

2.10. Protocolos de Comunicación

Los protocolos de comunicación son un conjunto de reglas y convenciones que permiten la transmisión de datos entre dispositivos en una red. Estos protocolos son esenciales para garantizar que los dispositivos puedan intercambiar información de manera efectiva y segura [23].

2.10.1. UART

El Universal Asynchronous Receiver-Transmitter, por sus siglas en inglés (UART) es un protocolo de comunicación serial ampliamente utilizado para la transmisión de datos entre dispositivos. Su característica principal es que es asíncrono, lo que significa que no requiere una señal de reloj compartida entre el transmisor y el receptor.

Características:

- Transmisión Asíncrona: No necesita sincronización de reloj, lo que simplifica su implementación y reduce la complejidad del sistema.
- Configuración Simple: Opera comúnmente con configuraciones de 8 bits de datos, 1 bit de parada y 1 bit de paridad opcional, facilitando su uso en diversas aplicaciones.

- Distancia de Comunicación: Es efectivo para distancias cortas, generalmente menos de 15 metros, debido a la posible degradación de la señal a medida que aumenta la distancia.
- Velocidad de Transmisión: Las tasas de baudios (baud rate) pueden variar desde 300 hasta 115200 bps o más, dependiendo del hardware y las condiciones del entorno.

Aplicaciones:

- Comunicación entre microcontroladores.
- Interfaces para sensores y dispositivos periféricos.
- Envío de datos a través de puertos serie en computadoras y dispositivos embebidos.

2.10.2. SSH

El Secure Shell (SSH) es un protocolo de red que permite la administración segura de dispositivos y la transferencia de datos a través de redes inseguras. SSH proporciona autenticación y cifrado, garantizando que los datos transmitidos estén protegidos contra ataques maliciosos.

Características:

- Cifrado: Utiliza algoritmos de cifrado robustos, como AES, para proteger la información durante su transmisión.
- Autenticación: Permite autenticación mediante contraseña o claves públicas, aumentando significativamente la seguridad en el acceso a los sistemas.
- Túneles Seguros: Facilita la creación de túneles seguros para otros protocolos, lo que permite la transferencia protegida de datos sensibles.
- Interfaz de Línea de Comando: Proporciona acceso remoto a la línea de comandos, permitiendo a los administradores gestionar sistemas sin necesidad de estar físicamente presentes.

2.11. Computadoras de guía, navegación y control

Las computadoras de guía, navegación y control son esenciales en diversas aplicaciones, particularmente en aviación, navegación marítima y vehículos autónomos. Su función principal radica en procesar datos de sensores para ofrecer información precisa que facilite la toma de decisiones en tiempo real. Estas tecnologías son fundamentales para garantizar la seguridad y eficiencia en el transporte moderno.

Existen tres tipos principales de computadoras en este ámbito: las computadoras de navegación, que determinan la posición y rumbo de embarcaciones o aeronaves mediante sistemas GPS e iniciales; las computadoras de control, que regulan el movimiento y estabilidad de los vehículos, como los controladores de vuelo; y las computadoras de guía, que ofrecen rutas óptimas a través de sistemas de navegación por satélite. Los componentes clave incluyen sensores que recogen datos del entorno, procesadores que realizan cálculos complejos y interfaces de usuario que permiten la interacción con el sistema.

Las aplicaciones de estas computadoras son variadas y críticas. En la aviación, se utilizan para el control de vuelo en aeronaves comerciales y militares. En la navegación marítima, aseguran rutas seguras para barcos. Además, son integradas en vehículos autónomos como coches y drones, permitiendo una navegación eficiente sin intervención humana. En conjunto, estas tecnologías no solo mejoran la seguridad, sino que también optimizan la eficacia del transporte actual.

2.11.1. EXA ICEPS

La computadora de vuelo EXA ICEPS (Integrated Control and Engine Performance System) es una tecnología avanzada diseñada para gestionar y optimizar el rendimiento del motor y otros sistemas críticos en aeronaves. Su integración de sistemas permite combinar el control del motor con diferentes funciones de la aeronave, lo que resulta en una gestión más eficiente y segura durante el vuelo.

Entre sus principales características se destacan el monitoreo en tiempo real, que proporciona datos sobre el rendimiento del motor, permitiendo a los pilotos tomar decisiones informadas. Además, la EXA ICEPS optimiza el rendimiento al maximizar la eficiencia del combustible y reducir las emisiones, contribuyendo así a operaciones más sostenibles. Sus funciones incluyen el control automático de los parámetros del motor y la facilitación del diagnóstico y mantenimiento predictivo, lo que ayuda a reducir costos operativos.

Este sistema ejemplifica cómo la tecnología moderna está revolucionando la aviación, mejorando tanto la seguridad como la eficiencia operativa. Al integrar múltiples funciones y proporcionar información crítica en tiempo real, la EXA ICEPS no solo optimiza el rendimiento de las aeronaves, sino que también promueve prácticas más sostenibles en la industria.

2.12. Revisión literaria

En los últimos años, las computadoras de guía, navegación y control han mostrado grandes avances en el desarrollo de sistemas autónomos.

2.12.1. Desarrollo de sistemas de navegación

En 2022, se presentó un sistema de planificación y control de navegación para vehículos autónomos en entornos urbanos. Este sistema permite la planificación de rutas basadas en la posición actual del vehículo y su destino, utilizando un controlador clásico que asegura el seguimiento de la trayectoria mediante odometría y correcciones visuales. Los resultados se simularon utilizando herramientas como ROS y Gazebo, lo que demuestra la viabilidad de estos sistemas en entornos complejos [24].

2.12.2. Transformación de Lenguaje de Bloques a Código C

La traducción de código de control de lenguaje de bloques a C implica un proceso de conversión donde cada bloque visual se asocia con una estructura de código en C. Esto se puede hacer utilizando herramientas de software que generan automáticamente el código C a partir de la lógica definida en el entorno de bloques. Este proceso no solo facilita la programación, sino que también permite la optimización del código generado para mejorar el rendimiento en sistemas de navegación autónoma.

XOD

XOD es un entorno de programación visual basado en bloques que permite a los usuarios crear programas para microcontroladores como Arduino. Este software genera automáticamente código en C++ a partir de la lógica definida en bloques. Los usuarios pueden conectar componentes gráficamente y, al finalizar, acceder al código generado, que es abierto y personalizable. XOD es gratuito y permite la creación de nuevos nodos para componentes específicos, lo que facilita la adaptación a diferentes proyectos [25].

Visual Microcontroller

Este software proporciona un lenguaje de programación gráfico para microcontroladores, desarrollado en C#. Utiliza una interfaz gráfica que permite a los usuarios diseñar diagramas que representan la lógica de control. El sistema compila el código a partir de diagramas gráficos, generando código intermedio en C antes de llegar al código hexadecimal necesario para la programación del microcontrolador [26].

LabVIEW

LabVIEW es un entorno de desarrollo que utiliza un enfoque gráfico para la programación. Aunque es más conocido en el ámbito de la ingeniería, también permite la generación de código en C. LabVIEW facilita la creación de aplicaciones de control y adquisición de

datos, y su capacidad para traducir diagramas de bloques a código C lo convierte en una opción útil para proyectos que requieren un control preciso de hardware.

Simulink

Como se mencionó anteriormente, Simulink, parte de MATLAB, proporciona un entorno gráfico para modelar, simular y analizar sistemas dinámicos. Permite a los usuarios crear modelos utilizando bloques y, posteriormente, generar código C automáticamente a partir de estos modelos. Esta herramienta es especialmente valiosa en aplicaciones de ingeniería donde se requiere un alto grado de precisión y control sobre el comportamiento del sistema.

2.13. Avances recientes en GNCs

En el marco del proyecto EROSS+ (European Robotic Orbital Support Services), se ha trabajado en el diseño de un sistema GNC altamente autónomo para misiones de servicio robótico en órbita. Este proyecto, que abarca desde 2021 hasta 2023, busca integrar técnicas avanzadas de navegación visual y control de cumplimiento para la captura y manipulación de satélites, mostrando un enfoque en la autonomía y la eficiencia operativa [27].

Otro desarrollo notable es el programa de NASA sobre GNC autónomo, que incluye sistemas para el transbordador espacial. Este programa se centra en la optimización de trayectorias de vuelo y la adaptación de sistemas GNC para diferentes condiciones de vuelo, lo que demuestra la importancia de la flexibilidad en el diseño de estos sistemas [28].

Además, la actividad VV4RTOS, apoyada por la Agencia Espacial Europea, se ha centrado en la verificación y validación de sistemas de control basados en optimización. Esto incluye el desarrollo de software GNC en tiempo real, lo que permite una validación más efectiva y segura de los sistemas diseñados [29].

2.13.1. Programación de Sistemas GNC

Los lenguajes de bloques, como Simulink, son comúnmente utilizados para diseñar y simular sistemas de control. Estos lenguajes permiten a los ingenieros visualizar el flujo de datos y las interacciones entre componentes de manera intuitiva. Sin embargo, la necesidad de traducir estos modelos a código C es crucial para su implementación en hardware real.

A pesar de los avances, existen desafíos significativos en la implementación de sistemas GNC. La variabilidad en los entornos operativos y la necesidad de adaptarse a condiciones cambiantes requieren algoritmos robustos y adaptativos. La optimización de estos sistemas es fundamental para asegurar su efectividad en misiones críticas.

Un estudio reciente sobre el sistema CubeNav destaca la importancia de desarrollar herramientas de análisis de navegación que faciliten las operaciones de GNC en misiones de CubeSats. Este enfoque busca reducir la curva de aprendizaje y minimizar errores humanos, lo que es esencial para misiones de bajo presupuesto y alta complejidad [29].

Capítulo 3

Plataforma de desarrollo

En este capítulo se pretende identificar una plataforma de hardware para el desarrollo de un modelo de ingeniería de una computadora de guía, navegación y control espacial por sus siglas en inglés (GNC), para llevar a cabo este objetivo se plantean los requerimientos que se deben de tomar en cuenta para elegir una tarjeta de desarrollo que logre satisfacer las necesidades de este proyecto, seguido de esto se seleccionaran un grupo de tarjetas las cuales cumplan con los requerimientos previamente establecidos, estas serán comparadas para poder determinar cuál de las tarjetas de desarrollo seleccionadas puede cumplir de mejor forma la tarea seleccionada.

3.1. Selección de la tarjeta de desarrollo

Para la selección de la tarjeta de desarrollo se partirá de la definición de los requerimientos de operación del sistema, esto tomando en cuenta las operaciones más comunes que realizan los sistemas GNC, una vez definidos los requerimientos, se seleccionaran al menos 3 tarjetas candidatas, esto con el fin de establecer los criterios de comparación para el desarrollo de una matriz de Pugh.

3.1.1. Requerimientos de la aplicación

Al elegir una tarjeta de desarrollo para un sistema de guía, navegación y control (GNC) en aplicaciones espaciales, se deben de tener en cuenta varios factores clave. Dentro de ellos se encuentran el procesamiento, más precisamente la capacidad de cálculo, ya que estos sistemas requieren de un procesamiento intensivo para los cálculos de trayectoria, estimación de estado y control. Seguido de esto se debe de considerar que sea un sistema de baja latencia, además que el mismo tenga soporte para sensores y actuadores para poder medir y controlar el sistema, además de esto los puertos de entrada y salida deben ofrecer la precisión necesaria para leer los datos de los sensores que se conecten al mismo.

Por otro lado el sistema debe de contener capacidades de tiempo real estricto, ya que los sistemas GNC deben de tomar decisiones críticas en el momento requerido. Además de tener la capacidad de ejecutar un sistema operativo de tiempo real (RTOS) o bien Linux en tiempo real. También un aspecto importante a contener por la tarjeta de desarrollo es el consumo de energía, esto sin dejar de lado las capacidades de simulación y pruebas, ya que, en la interfaz de simulación la tarjeta se debe de poder conectar a un entorno de pruebas de hardware-in-the-loop por sus siglas en inglés (HIL), además de las capacidades de depuración y monitoreo.

Finalmente se deben de tomar en cuenta aspectos como lo son el Tamaño, peso y forma de la tarjeta buscando que las mismas contengan un tamaño compacto, ya que los sistemas espaciales siempre se deben de integrar en espacios reducidos y la resistencia del mismo.

3.1.2. Tarjetas candidatas

Bajo los requerimientos planteados anteriormente se eligieron las siguientes tarjetas de desarrollo, las mismas se presentarán con sus características.

Xilinx ZCU102 Evaluation Kit

La tarjeta de desarrollo ZCU102 contiene procesamiento basado en Zynq UltraScale+ MPSoC, el cual combina un procesador ARM Cortex-A53 de 64 bits con una FPGA de alto rendimiento, la cual es excelente para el procesamiento en tiempo real y algoritmos personalizados.

Por otro lado, esta ofrece una amplia gama de interfaces de comunicación como: PCIe, Ethernet, I2C, SPI, UART, GPIO. En cuanto a la eficiencia energética esta opción contiene mecanismos para el control de energía, además de ser compatible con entornos de simulación y tiene interfaces JTAG para una buena depuración. Finalmente es una tarjeta ampliamente utilizada en la industria en sistemas de prototipos avanzados y despliegue de HIL.

En síntesis esta opción ofrece un procesamiento potente y versátil además de ser excelente para desarrollar y escalar sistemas GNC complejos, por otro lado, es una tarjeta de desarrollo costosa.

NVIDIA Jetson AGX Xavier

Para el caso de la tarjeta AGX Xavier de NVIDIA incorpora una CPU ARM v8.2 de 64 bits y una GPU NVIDIA Volta, prestaciones las cuales se encargan de proporcionar un alto nivel de procesamiento de datos en paralelo especialmente utilizado para aplicaciones de visión por computador o inteligencia artificial para sistemas GNC.

Las interfaces de comunicación presentes en esta tarjeta son puertos I2C, SPI, UART y

GPIO además de contener adicionalmente soporte nativo para cámaras y sensores de alta gama. Además presenta capacidades en tiempo real, ya que se puede implementar con el NVIDIA Jetpack SDK.

En cuanto a la eficiencia energética, la misma posee un diseño optimizado para bajo consumo. Además de ser compatible con interfaces de pruebas y simulación mediante el uso de entornos como Tensor RT y otras plataformas propietarias del desarrollador de la tarjeta de desarrollo.

Esta tarjeta es ideal para sistemas GNC con un procesamiento intensivo de datos ya sean de visión por computador o bien inteligencia artificial, por otro lado posee un desarrollo potente para el procesamiento de tareas en paralelo o bien de aprendizaje reforzado, finalmente contiene un buen soporte para aplicaciones en tiempo real y simulaciones. Por otro lado contiene un bajo procesamiento lógico comparado con las FPGA para aplicaciones en tiempo real extremo, y esta tarjeta de desarrollo se encuentra más enfocada en la implementación de soluciones que requieran inteligencia artificial.

TMS320C6678 Development Kit

La tarjeta TMS320C6678 es basada en un procesador para procesamiento digital de señales (DSP) de 8 núcleos, está enfocado a aplicaciones de procesamiento intensivo en tiempo real, soporta interfaces como lo son Ethernet, SPI, UARTm I2C y GPIO, además de tener opciones para expandir la conectividad de la misma, por otro lado como mencionamos anteriormente es uno de los sistemas más optimizados para el procesamiento en tiempo real por medio de la plataforma propietaria TI RTOS.

Sobre la eficiencia energética, esta tarjeta de desarrollo ofrece herramientas específicas para la optimización del consumo de energía, haciéndola adecuada para entornos de consumo energético restringido, finalmente es compatible con Code Composer Studio, el cual es un entorno de desarrollo integrado que facilita las labores de integrar, simular y depurar el código implementado. Por tanto podemos decir que es una tarjeta de desarrollo muy adecuada para los sistemas de procesamiento de señales y control, tiene una gran capacidad para soportar aplicaciones industriales y aeroespaciales. Por otro lado, podemos ver que es una plataforma menos flexible que una FPGA.

ZedBoard de Avnet

En cuanto a procesamiento, para la tarjeta ZedBoard, tenemos que utiliza un procesador Xilinx Zynq-7000 APSoC el cual combina un procesador ARM Cortex-A9 dual core con una FPGA programable, de esta forma tomando el procesador ARM el cual es ideal para ejecutar algoritmos de control y lógica de navegación en un entorno de RTOS o bien Linux, por otro lado la FPGA Zynq-7000 permite la ejecución de tareas de procesamiento paralelo en hardware como el filtrado de señales o algoritmos de estimación de estados, ofreciendo baja latencia y flexibilidad en tiempo real.

En cuanto a las interfases de entrada y salida, incluye varias opciones como lo son: GPIO, I2C, SPI, UART. Además de esto contiene puertos Ethernet, micro usb y HDMI los cuales resultan útiles para la comunicación externa y visualización de los sistemas de desarrollo.

La combinación de un procesador ARM con una FPGA permite un equilibrio en el consumo de energía, ya que la mayoría de tareas intensivas se pueden llevar a cabo en la FPGA y el SoC Zynq ofrece opciones de ahorro de energía lo cual siempre representa un beneficio para las aplicaciones embebidas. En cuanto a las pruebas y simulaciones, cuenta con entornos como Vivado y SDK de Xilinx esto con el fin de realizar simulaciones de HIL. Por otro lado ofrece aplicaciones para depuración como lo es JTAG para el monitoreo en tiempo real de las aplicaciones ejecutándose en la FPGA y en el procesador ARM.

3.1.3. Criterios de comparación

Una vez presentadas las tarjetas de desarrollo candidatas se procede con la definición de los criterios de comparación: para este caso los criterios a tomar en cuenta son los siguientes:

1. Capacidad de procesamiento: La capacidad de procesamiento en dispositivos de desarrollo para sistemas GNC es crucial porque garantiza la ejecución en tiempo real de algoritmos complejos, como los de control y fusión de sensores, que son esenciales para la estabilidad y precisión del sistema. Permite procesar grandes volúmenes de datos de múltiples sensores de manera simultánea y rápida, ejecutar tareas en paralelo, y realizar cálculos intensivos como la planificación de trayectorias y control adaptativo. Además, un procesamiento robusto facilita la simulación HIL, asegurando pruebas y simulaciones realistas.
2. Soporte para sensores y actuadores: El soporte para sensores y actuadores es esencial en dispositivos de desarrollo para sistemas GNC porque estos sistemas dependen de la entrada de múltiples sensores como acelerómetros, giroscopios, GPS, entre otros, para monitorear y estimar la posición, orientación y velocidad del vehículo en tiempo real. La capacidad de interactuar directamente con estos sensores, y con actuadores que ejecutan las acciones de control, es fundamental para garantizar la retroalimentación continua y precisa necesaria para el correcto funcionamiento del sistema GNC. Interfaces como I2C, SPI, UART, y GPIO permiten esta integración, asegurando un control eficiente y adaptable.
3. Capacidad de trabajo en tiempo real: La capacidad de trabajo en tiempo real es vital en dispositivos de desarrollo para sistemas de guía, navegación y control (GNC) porque estos sistemas requieren respuestas inmediatas y precisas ante cambios en el entorno o en las condiciones del vehículo. Los algoritmos de control, como los de estabilidad y trayectoria, deben ejecutarse sin demoras para garantizar la seguridad y el rendimiento óptimo del sistema. Sin procesamiento en tiempo real, las decisiones

de control podrían retrasarse, afectando la estabilidad y el control del vehículo, lo cual es crítico en aplicaciones como navegación autónoma o vuelo espacial.

4. Consumo de energía: El consumo de energía es crucial en dispositivos de desarrollo para sistemas de guía, navegación y control (GNC), especialmente en aplicaciones espaciales o autónomas, donde los recursos energéticos son limitados. Un consumo eficiente permite que el sistema funcione de manera prolongada sin comprometer su rendimiento, maximizando la duración de la misión y asegurando que los componentes críticos, como sensores y actuadores, siempre reciban suficiente energía. Además, la gestión adecuada del consumo evita el sobrecalentamiento y prolonga la vida útil de los dispositivos, lo que es fundamental en entornos de operación prolongada o difíciles de acceder.
5. Características Físicas: Las características físicas, como el tamaño, peso y forma, son importantes en dispositivos de desarrollo para sistemas de guía, navegación y control (GNC) porque estos sistemas suelen implementarse en entornos con restricciones de espacio y peso, como en vehículos aéreos, drones o satélites. Un dispositivo compacto y ligero facilita la integración en estos sistemas sin afectar su desempeño ni su capacidad de carga. Además, un diseño físico optimizado es clave para minimizar los efectos de vibraciones, choques o cambios de temperatura, asegurando un funcionamiento fiable en condiciones extremas.
6. Costo: El costo es un factor importante en dispositivos de desarrollo para sistemas de guía, navegación y control (GNC) porque influye directamente en la viabilidad económica del proyecto, especialmente en fases de prototipado o prueba. Un dispositivo con un costo adecuado permite realizar iteraciones y pruebas sin superar el presupuesto, facilitando el acceso a tecnologías avanzadas sin comprometer la calidad. Además, un costo equilibrado permite escalar el proyecto o implementar múltiples sistemas de prueba, optimizando el desarrollo sin sacrificar funcionalidad o capacidad técnica.
7. Escalabilidad del sistema: La escalabilidad del sistema es crucial en dispositivos de desarrollo para sistemas de guía, navegación y control (GNC) porque permite adaptar el hardware y software a medida que el proyecto crece en complejidad o requisitos técnicos. Un dispositivo escalable facilita la integración de nuevos sensores, algoritmos más avanzados o mayores capacidades de procesamiento sin necesidad de cambiar completamente la plataforma. Esto ahorra tiempo y costos, además de asegurar que el sistema pueda evolucionar para cumplir con las demandas de futuras fases del desarrollo o nuevas aplicaciones, manteniendo la flexibilidad y la eficiencia.
8. Soporte del flujo de trabajo Yocto Project: Distribuciones ligeras y personalizables como Yocto Project se usan en hardware embebido como procesadores ARM. Estas permiten construir un sistema operativo ajustado a las necesidades del sistema GNC.

3.2. Matriz de Pugh

Tabla 3.1: Matriz de Pugh para seleccionar la tarjeta de desarrollo que mejor se adapte a los requerimientos del proyecto

Criterios	Peso	ZCU102	AGX Xavier	TMS320C6678	Zedboard
Capacidad de procesamiento	15	15	15	8	10
Soporte para sensores	15	15	15	15	15
Soporte para actuadores	10	10	10	10	10
Soporte de sistemas de tiempo real	20	20	20	15	20
Características Físicas	10	4	7	10	10
Costo de la tarjeta	10	7	7	7	10
Escalabilidad del sistema	10	10	6	6	10
Soporte de Yocto Project	10	10	0	0	10
Suma general		80	80	75	95
Posición		3	2	4	1

Como se pudo observar en la Tabla 3.1, un claro ganador según los requerimientos establecidos para este proyecto ha sido la tarjeta de desarrollo ZedBoard, ya que es la mejor opción en cuanto a características como lo son la capacidad de procesamiento y el soporte de sistemas en tiempo real, además de esto cabe destacar que es una de las tarjetas con una mejor relación costo/rendimiento, ya que, procesador que contiene la misma unidad central de procesamiento que la tarjeta ICEPS desarrollada por EXA, la cual como se mencionó en 2.11.1, es un sistema todo-en-uno para CubeSats que combina potencia, computación a bordo y comunicación en una sola plataforma compacta. Está diseñado para ser eficiente y personalizable, con capacidad para transferencias de datos de alta velocidad y almacenamiento expansible.

3.3. Plataforma seleccionada

Como se pudo observar en la Tabla 3.1, la tarjeta de desarrollo seleccionada fue la Tarjeta Zedboard de Avnet. La ZedBoard es una tarjeta de desarrollo basada en el SoC (System-on-Chip) Xilinx Zynq-7000. Diseñada para aplicaciones de desarrollo en sistemas embebidos y de procesamiento de señales, la ZedBoard combina la potencia de un procesador ARM con la flexibilidad de una FPGA (Field Programmable Gate Array), proporcionando una plataforma versátil para la investigación, el desarrollo y la prueba de diversas aplicaciones, incluidas las de guía, navegación y control (GNC).

3.3.1. Especificaciones principales

Como se mencionó en el capítulo 2 en la Tabla 2.2 y anteriormente en este capítulo la tarjeta en cuestión presenta las siguientes especificaciones principales.

Procesador y FPGA

SoC Xilinx Zynq-7000: La ZedBoard integra un procesador ARM Cortex-A9 dual-core junto con una FPGA programable de la serie 7-Series. ARM Cortex-A9: Ofrece un rendimiento de procesamiento general que puede ejecutar sistemas operativos como Linux o FreeRTOS, lo que es útil para tareas de control y procesamiento de datos. FPGA: La FPGA proporciona capacidad para implementar lógica personalizada, lo que permite el desarrollo de algoritmos específicos en hardware para procesamiento en tiempo real y alta velocidad.

Interfaces de E/S

GPIO: General Purpose Input/Output, permite la interacción con una amplia gama de periféricos y sensores. I2C, SPI, UART: Protocolos de comunicación estándar que facilitan la integración con diversos dispositivos de sensor y actuadores. Ethernet: Conectividad de red para comunicación y transmisión de datos. USB: Puertos USB para conexión de dispositivos externos y almacenamiento. HDMI: Salida de video para visualización de datos y control gráfico. JTAG: Para depuración y programación de la FPGA y el procesador ARM.

Memoria

RAM: Incluye memoria DDR3 SDRAM para el procesador y la FPGA, proporcionando espacio suficiente para la ejecución de sistemas operativos y algoritmos complejos. Flash: Memoria flash para almacenamiento de configuraciones y datos persistentes.

Alimentación y Consumo de Energía

La ZedBoard se alimenta típicamente a través de una entrada de 5V, con un diseño que optimiza el consumo energético para aplicaciones de desarrollo. Sin embargo, el consumo real depende del uso de la FPGA y el procesador.

Tamaño y Factor de Forma

Dimensiones: Aproximadamente 15.24 cm x 22.86 cm (6 x 9 pulgadas), lo que la hace adecuada para prototipos sin ser excesivamente grande. Diseño: Compacta pero con suficiente espacio para interfaces y módulos adicionales.

Capacidades de Desarrollo

Entornos de Desarrollo: Compatible con Xilinx Vivado Design Suite y SDK, proporcionando herramientas avanzadas para diseño, simulación, y depuración. Ejemplos de Aplicaciones: Adecuada para aplicaciones que requieren procesamiento en paralelo, desarrollo de sistemas embebidos, y prueba de algoritmos de control.

3.4. Reflexión final

Como se pudo observar a lo largo de este capítulo se analizaron los requerimientos de hardware que se deben de tomar en cuenta para el desarrollo de este proyecto, seguido de esto se tomaron cuatro tarjetas de desarrollo con el fin de elegir entre las que presentaran las prestaciones adecuadas para la tarea a realizar. La Avnet Zedboard y la computadora de vuelo EXA ICEPS comparten el procesador ARM Cortex-A9 de 32 bits, lo que permite una comparación directa en rendimiento y capacidades. Esta similitud en la arquitectura proporciona una base sólida para validar casos de estudio antes de su implementación en sistemas críticos como la EXA ICEPS. La Zedboard se presenta como una herramienta valiosa para el desarrollo y prueba de algoritmos en un entorno controlado.

Capítulo 4

Síntesis de software para GNC embebido

Como se pudo observar en el capítulo 3, se realizó la selección de la tarjeta de desarrollo Zedboard, además de esto uno de los requerimientos que se tomó en cuenta fue la compatibilidad de esta con el flujo de trabajo de Yocto Project.

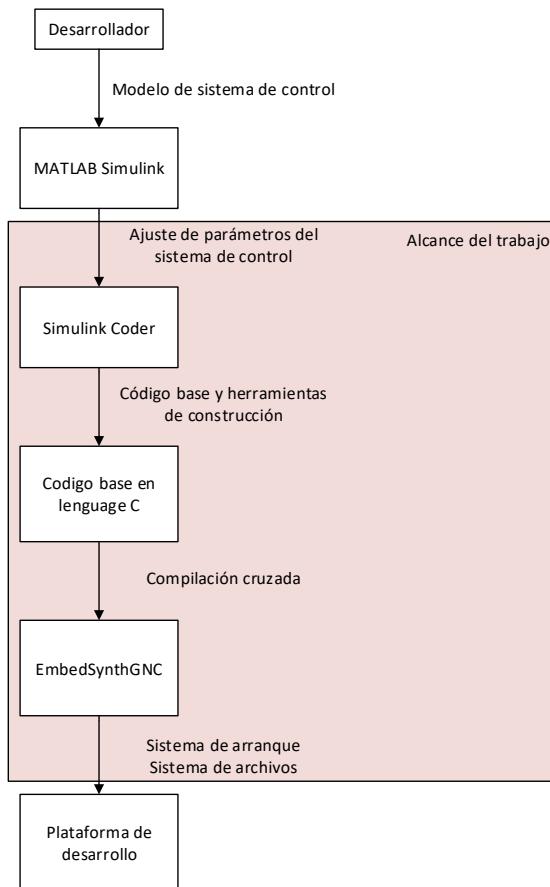


Figura 4.1: Diagrama general del flujo de trabajo propuesto

En este capítulo se pretende establecer los flujos de trabajo para el prototipado de algoritmos de control, orientación y navegación para aplicaciones espaciales. Esto mediante la implementación de un caso de estudio básico. Como se puede observar en la Figura 4.1, primeramente el desarrollador toma un modelo de sistema de control, el cual debe implementar en MATLAB Simulink por medio de los bloques del sistema.

Seguido de esto se realiza un ajuste de parámetros con el objetivo que el mismo opere dentro de las condiciones esperadas, una vez ajustado el sistema, se debe de ingresar el modelo al flujo de trabajo de Simulink Coder, el cual nos dará como salida el código base y las herramientas de construcción requeridas para poder compilar el archivo ejecutable, cabe destacar que la compilación a realizar es cruzada, esta nos permite compilar los archivos en un sistema distinto al objetivo.

Finalmente se debe de ingresar el archivo binario generado al flujo de trabajo de EmbedSynthGNC, el cual, se encargara de dar como salida un sistema de archivos de arranque y el sistema de directorios listos para implementar en la plataforma de desarrollo.

4.1. Creación de los entornos de desarrollo

En esta sección se describen los pasos necesarios para la creación de los entornos de desarrollo utilizados en este proyecto. En el mismo se incluye la información del hardware y software utilizado, además de las versiones de los sistemas para cada una de las implementaciones. Cabe destacar que el computador anfitrión en el cual se implementan estos entornos de desarrollo tiene las siguientes características:

- Procesador Intel i9 - 13900k
- Almacenamiento 1 TB
- Memoria 16 GB
- Video Nvidia RTX4060Ti

4.1.1. MATLAB

Para el entorno de MATLAB se utiliza como sistema operativo Windows 11, en su versión de 64 bits, se destina a este sistema operativo un espacio en disco de 500 GB, en el cual se instalen los programas requeridos para la generación de contenido y el análisis de datos. La versión utilizada es MATLAB 2024b por medio de una licencia de prueba de la versión full, la cual nos permitirá generar y probar el contenido necesario para los casos de uso a tratar. Por otro lado, para el análisis y procesamiento de datos se utiliza Python 3.12.3.

Como se muestra en la Figura 4.2, para la generación de este entorno de trabajo primariamente se deberá de realizar la instalación de Python 3.12.3 y seguido de esto la instalación

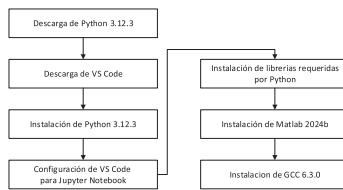


Figura 4.2: Diagrama para generar el entorno de MATLAB

de MATLAB con las herramientas requeridas. Finalmente se debe de instalar GCC en su versión 6.3.0.

Primeramente se debe de realizar la descarga de Visual Studio desde [este enlace](#), esto con el fin de poder hacer uso de Jupyter Notebook para el análisis de datos, además de esto se debe de descargar Python en su versión 3.12.3 desde esta [dirección](#).



Figura 4.3: Menú de instalación de Python

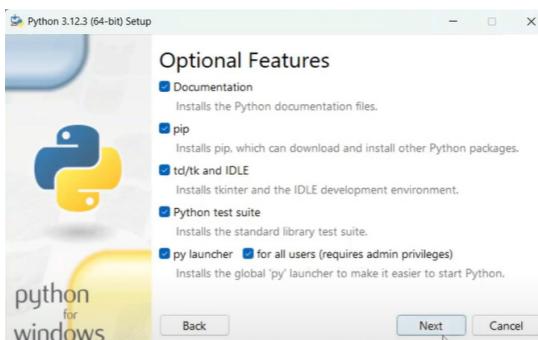


Figura 4.4: Menú de instalación de Python

Una vez descargados estos programas se debe de realizar la instalación de los mismos, primeramente se debe de instalar Python, el cual como se observó en la Figura 4.3 se debe de seleccionar la opción para agregar Python al directorio PATH, esto con el fin de poder instalar los entornos y librerías requeridas. Seguido de esto se debe de seleccionar la opción de instalación personalizada (Customize installation) con el fin de poder agregar las características que se mostraron en la Figura 4.4.

Una vez concluida la instalación de Python se debe proseguir con VS Code, el cual como se pudo observar en la Figura 4.5, se deben de aceptar los términos y condiciones del

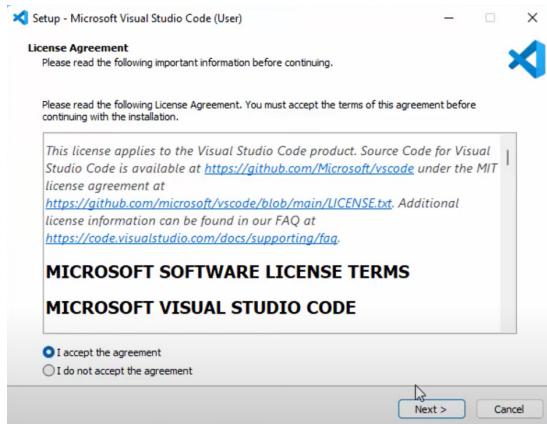


Figura 4.5: Menú de instalación de VS Code

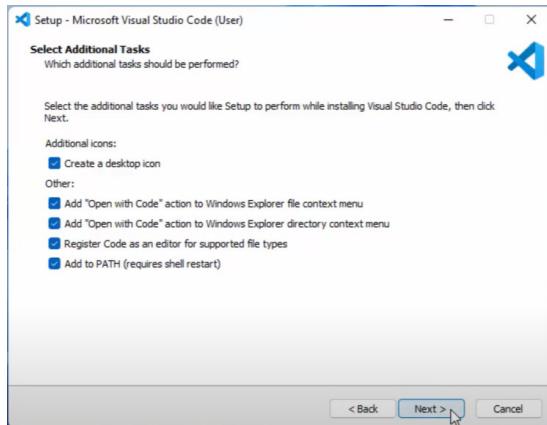


Figura 4.6: Menú de instalación de VS Code

programa y en el siguiente menú se deben seleccionar las opciones marcadas en la Figura 4.6.

Finalmente se debe de realizar la descarga de MATLAB la cual se puede conseguir por medio de [este enlace](#), se debe de generar una cuenta y descargar la versión de prueba que tiene una durabilidad de 30 días, o bien si se cuenta con la licencia del producto no es necesario recurrir a esta. Dentro de la instalación es importante mencionar que se podrán solicitar todas las herramientas dentro del periodo de prueba, pero las que son fundamentales para el desarrollo del mismo son las siguientes:

- MATLAB Simulink
- MATLAB Coder

Además, se instala la herramienta GCC en su versión 6.3.0, el cual es el compilador de código C, este se utilizará para compilar el código y probar que el mismo funcione de la forma esperada; cabe destacar, que este no es el compilador cruzado, este compilador nos genera un archivo binario el cual podremos probar en el sistema Windows, esto con

el fin de poder realizar correcciones más eficientes al sistema de control en caso de ser requerido.

4.1.2. Contenedores

En la elaboración del proyecto se decanta por el uso de contenedores, ya que como se mencionó en 2.9, estos permiten encapsular todas las dependencias y configuraciones necesarias para ejecutar una aplicación. Esto asegura que la misma aplicación funcione de manera consistente en diferentes entornos, ya sea durante el desarrollo, en pruebas o en producción.

Cada contenedor opera de forma independiente, lo que permite ejecutar múltiples aplicaciones en el mismo sistema sin interferencias. Este aislamiento reduce significativamente el riesgo de conflictos entre diferentes versiones de software y dependencias, lo que se traduce en un entorno más estable y predecible. Los contenedores se elaboran en el computador anfitrión en un sistema operativo Linux en su versión 22.04 LTS.

Comandos utilizados para la creación y gestión de contenedores

```
1 sudo apt install docker.io
```

Listing 4.1: Instalacion de docker, Linux

Como se pudo observar, mediante el uso de 4.1, se realiza la instalación de docker.io, el cual como se mencionó en 2.9, es una plataforma que permite la creación, despliegue y gestión de aplicaciones en contenedores.

```
1 sudo docker run -it ubuntu:20.04 /bin/bash
```

Listing 4.2: Instalacion de Ubuntu 20.04, Linux

Por medio del comando 4.2 se realiza la construcción del contenedor. Este último comando se encarga de descargar la imagen de Ubuntu 20.04, crea y ejecuta un contenedor en modo interactivo además de proporcionar acceso al terminal del contenedor, donde se ejecutan comandos como si fuera una máquina virtual con Ubuntu 20.04. Este contenedor será utilizado para realizar la compilación cruzada, se utiliza Ubuntu 20.04, ya que esta versión contiene la versión de GCC que satisface los requerimientos impuestos por otros flujos de trabajo.

```
1 sudo docker run -it ubuntu:16.04 /bin/bash
```

Listing 4.3: Instalacion de Ubuntu 16.04, Linux

Por otro lado por medio del comando mostrado en 4.3, se genera el contenedor que se utilizará para la implementación del flujo de trabajo de Yocto Project, cabe destacar que se utiliza Ubuntu 16.04, ya que, es la versión que cumple con los requerimientos del flujo de trabajo de Yocto, esto se debe a que al trabajar con Yocto Zeus y este ser desarrollado en el año 2019, esta era la versión de Ubuntu más estable para la fecha.

```
1 sudo docker ps -a
```

Listing 4.4: Lista de contenedores del sistema, Docker

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1420fedd3067	ubuntu:20.04	"bash"	13 days ago	Exited (255) 4 days ago		crazy_brattain
20a26f16e5ac	ubuntu:16.04	"/bin/bash"	4 weeks ago	Exited (0) 4 days ago		dreamy_jemison

Figura 4.7: Retorno del comando 4.4

Una vez generados los dos contenedores, los mismos podrán ser consultados mediante el comando que se muestra en 4.4, este se encarga de devolver una lista como la que se muestra en 4.7, donde se puede observar de izquierda a derecha las columnas denominadas: Container ID, el cual es el identificador del contenedor, este valor se debe de utilizar para referirse al contenedor para las operaciones que se realizan fuera del entorno, también se muestra la Imagen que el contenedor tiene instalada, el comando de creación del mismo, la creación del contenedor, el status, los puertos definidos y el Nombre.

Para efectos de este proyecto se tomará importancia solamente a las columnas de (Container ID) e (Image).

```
1 sudo docker start <ID del contenedor>
```

Listing 4.5: Iniciar un contenedor, Docker

```
rex@rex-millize:~$ sudo docker start 1420fedd3067
1420fedd3067
```

Figura 4.8: Retorno del comando 4.5

Por medio del comando que se muestra en 4.5 se inicia el contenedor deseado, esto siempre dependerá del ID que se coloque, tras la ejecución de este comando la salida del mismo se puede observar en la Figura 4.8.

```
1 sudo docker exec -it <ID del contenedor> \bin\bash
```

Listing 4.6: Ingresar a un contenedor, Docker

```
rex@rex-millize:~$ sudo docker exec -it 1420fedd3067 /bin/bash
root@1420fedd3067:/#
```

Figura 4.9: Retorno del comando 4.6

Finalmente, para la ejecución del contenedor en terminal se debe de hacer uso del comando 4.6, este se encarga de acceder a la terminal del contenedor deseado. Como se puede observar en la Figura 4.9 ya la terminal se encuentra ejecutando el contenedor construido bajo el ID del contenedor "1420fedd3067".

Una vez generados los contenedores, se procede a generar el entorno de desarrollo requerido dentro de los mismos.

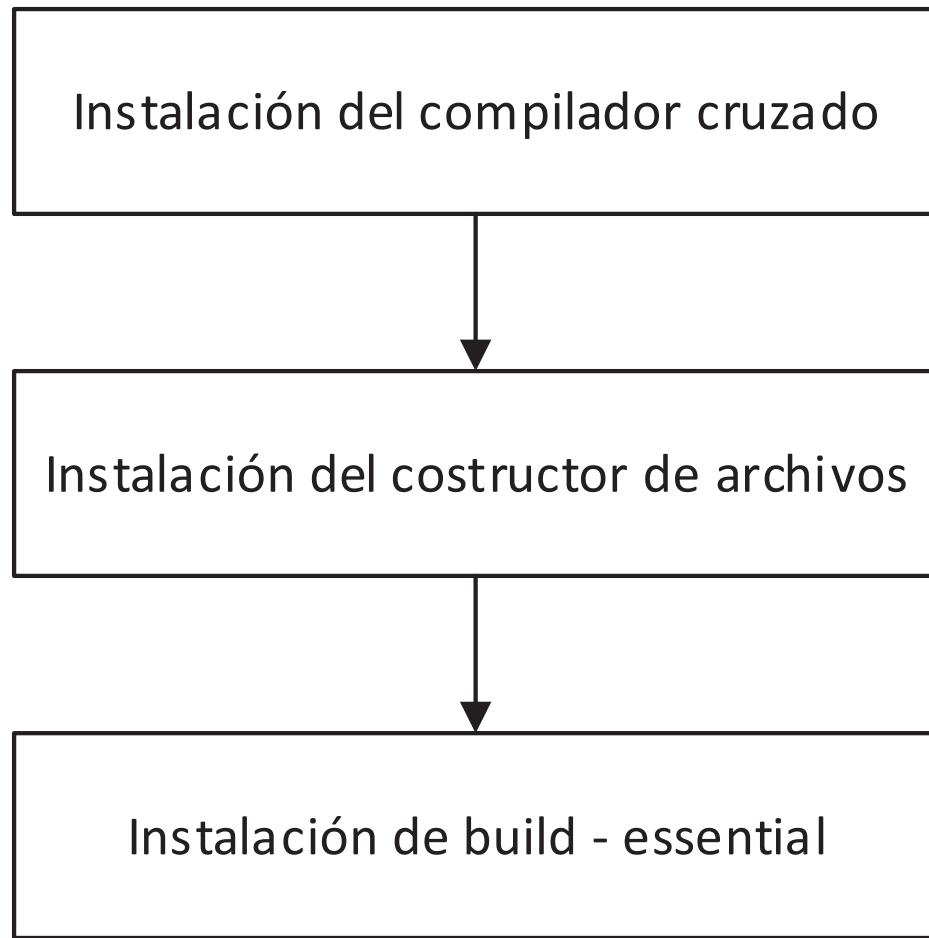


Figura 4.10: Diagrama para la elaboración del entorno para compilación cruzada

Contenedor para compilación Cruzada

Para la compilación de los archivos se elabora un entorno para compilación cruzada, esto se debe a que el archivo se deberá de compilar para la ejecución en la plataforma de desarrollo seleccionada, es por esto que para la preparación del entorno se siguen los pasos que se muestran en la Figura 4.10

```
1 sudo apt install gcc-arm-linux-gnueabihf
```

Listing 4.7: Instalacion del compilador cruzado, Contenedor

```
1 sudo apt install cmake
```

Listing 4.8: Instalacion de CMake, Contenedor

```
1 sudo apt install build-essential
```

Listing 4.9: Instalacion de build essential, Contenedor

Primeramente se debe de instalar el compilador cruzado, para nuestros efectos será arm-linux-gnueabihf-gcc, esto lo podemos hacer mediante el comando que se muestra en 4.7.

Seguido de esto debemos de instalar CMake el cual es una herramienta de construcción multiplataforma y de código abierto que se utiliza para gestionar la construcción de software, el mismo se instala mediante el comando que se muestra en 4.8. Finalmente debemos de instalar build-essential las cuales son herramientas que nos ayudaran a compilar el programa generado, esta instalación se logra mediante el uso del comando mostrado en 4.9. Con estas tres instalaciones queda preparado el entorno para la compilación cruzada.

Contenedor para Yocto Project

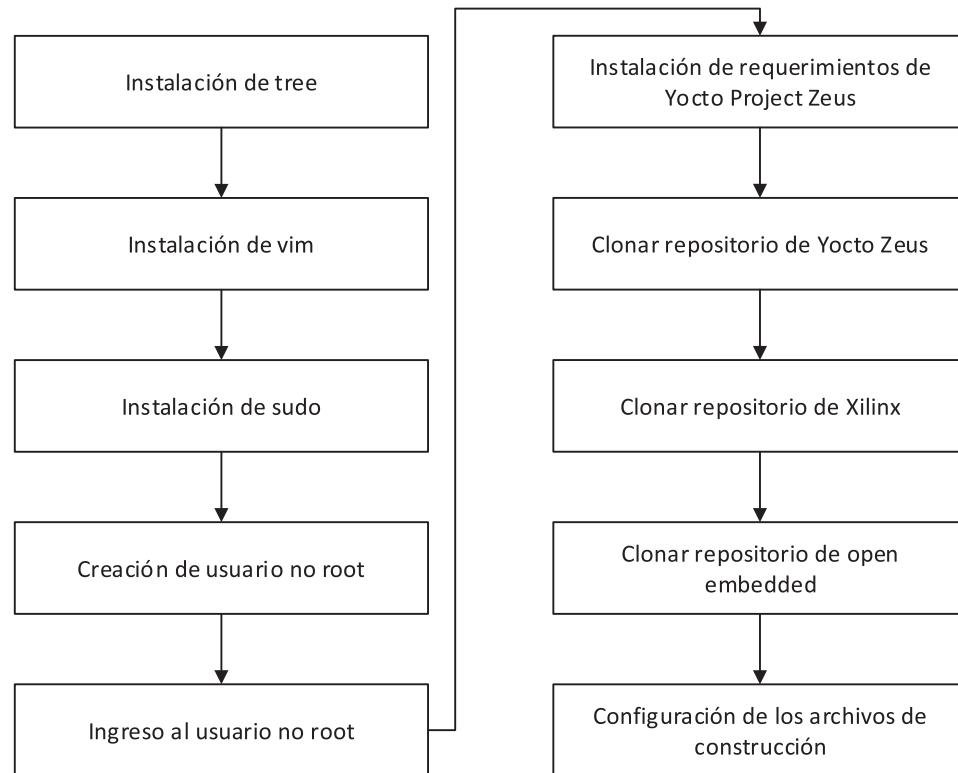


Figura 4.11: Diagrama para la elaboración del entorno para Yocto Project

El contenedor para Yocto Project, será en el cual se lleve a cabo la construcción de la imagen a utilizar en la plataforma de desarrollo, para esto se requiere la instalación de algunas dependencias las cuales se pueden observar en la Figura 4.11.

Creación de un usuario no root

Para el uso del marco de trabajo de Yocto se debe de generar un usuario no root, esto principalmente por razones de seguridad y manejo adecuado de permisos. El usuario se puede generar por medio de los siguientes comandos:

```

1 apt - get install -y sudo
2 useradd - ms/bin/bash myuser
  
```

```

3 echo "myuser:password" | chpasswd
4 usermod - aG sudo myuser

```

Listing 4.10: Generacion de usuario no root, Linux

De forma que según el comando observado en 4.10, en la primera línea instalamos sudo, esto con el fin de poder ejecutar comandos con privilegios de administrador sin tener que iniciar sesión como usuario root, seguido de esto en la segunda línea se agrega el usuario denominado (myuser), en la tercera línea se genera una contraseña para este usuario la cual se define como (chpasswd), finalmente se agrega en el archivo usermod el nuevo usuario el cual, mediante el comando sudo tiene acceso a los privilegios de administrador.

```

1 su - myuser

```

Listing 4.11: Iniciar usuario no root, Linux

Cada vez que iniciemos el contenedor siempre lo haremos como usuario root, para poder iniciar con el usuario no root generado en 4.10, se debe de hacer uso del comando que se muestra en 4.11, el cual se encarga de iniciar el entorno bajo el usuario llamado (myuser). Cada vez que se realice alguna instalación o procedimiento que requiera permisos de administrador se deberá de ingresar la contraseña.

Yocto Project

Como se observó en 2.4.1, yocto presenta flexibilidades a la hora de configurar un sistema permitiendo al desarrollador seleccionar paquetes específicos y personalizar el sistema operativo.

```

1 sudo apt - get install gawk wget git - core diffstat unzip texinfo
2 gcc - multilib build - essential chrpath socat cpio python python3
3 python3 - pip python3 - pexpect xz - utils debianutils
4 iputils - ping python3 - git python3 - jinja2
5 libegl1 - mesa libsdl1 .2 - dev pylint3 xterm

```

Listing 4.12: Requerimientos Yocto Zeus, Linux

Para que el mismo funcione de forma correcta debemos de instalar los requerimientos del marco de trabajo los cuales se pueden observar en 4.12.

```

1 git clone -b zeus https://git.yoctoproject.org/git/poky
2 cd poky

```

Listing 4.13: Version de Yocto

```

1 git clone -b zeus https://github.com/Xilinx/meta-xilinx
2 git clone -b zeus https://github.com/openembedded/meta-openembedded.git

```

Listing 4.14: BSP para Zedboard

La versión de yocto a utilizar es Yocto Zeus, ya que esta versión es la que contiene soporte para la tarjeta de desarrollo seleccionada, los archivos de esta versión se deben de clonar de 4.13. Una vez clonado el repositorio se debe de ingresar al directorio (poky) y clonar dentro del mismo el repositorio 4.14 el cual contiene la versión de paquete de soporte para la tarjeta requerida, esto con el fin de generar una imagen para la tarjeta de desarrollo seleccionada.

```

1 source oe - init - build - env
2
3 echo "MACHINE ??=\\"zedboard-zynq7\\"" >> conf/local.conf
4 echo "IMAGE_FEATURES +=\\\"package-management\\\" >> conf/local.conf
5 echo "DISTRO_HOSTNAME =\\\"zynq\\\" >> conf/local.conf
6
7 bitbake-layers add-layer ../meta-xilinx/meta-xilinx-bsp/
8 bitbake-layers add-layer ../meta-openembedded/meta-oe/

```

Listing 4.15: Configuraciones adicionales, Yocto

Una vez clonados los repositorios podemos continuar con algunas configuraciones adicionales que se deben de realizar, con el fin de configurar correctamente el entorno de trabajo, estas se muestran en 4.15. La primera línea se encarga de configurar el entorno de trabajo dentro del directorio llamado (build). Dentro del entorno de trabajo generado en (build) en el directorio llamado (conf) encontramos archivos como lo son el local.conf el cual es un archivo de configuración utilizado por el sistema de compilación BitBake. Este archivo esta escrito en lenguaje de configuración de BitBake, define aspectos importantes como la arquitectura del objetivo, la distribución del sistema las imágenes a crear y las herramientas utilizadas, además de incluir rutas adicionales a capas de Yocto o bien a configuraciones específicas de las recetas.

Además de esto también nos encontramos con el archivo llamado bblayers.conf, este al igual que local.conf, es un archivo de configuración de BitBake que define las capas que se utilizan durante el proceso de compilación. En este como se mencionó anteriormente se encuentran las rutas de las diferentes capas que el sistema de compilación de Yocto debe usar. Estas capas contienen las recetas y los archivos de configuración que controlan la compilación de paquetes, imágenes y el sistema operativo en general. Cabe destacar que el orden en el que se listan las capas puede ser importante, ya que capas más específicas pueden sobreescribir las configuraciones o recetas de capas más generales.

Una vez comprendidos estos conceptos la línea 3 de 4.15 se encarga de agregar la máquina, que en nuestro caso sería la tarjeta de desarrollo ZedBoard de Avnet, la línea 4 se encarga de habilitar la administración de paquetes, esto quiere decir que se asegura que el sistema que se está construyendo incluirá las herramientas necesarias para gestionar paquetes (como instalar, actualizar o eliminar) después de que el sistema haya sido desplegado. La línea 4 es la encargada de establecer el nombre del anfitrión, el sistema operativo generado tendrá el nombre de anfitrión configurado como zynq. Por otro lado, la línea 7 y 8 se encargan de agregar las capas al archivo de bblayers.conf.

Finalizada la configuración básica de los entornos de trabajo se sigue con las siguientes

secciones en donde se desarrollaran los casos de estudio con el objetivo de demostrar el funcionamiento del flujo de trabajo propuesto en la Figura 4.1.

4.2. Caso de estudio 1 - Filtro Básico en MATLAB

Como caso de estudio se seleccionó una aplicación la cual permite realizar una comparación de resultados antes y después del procesamiento de datos, es por esto que se decidió implementar un filtro básico de tipo paso bajo haciendo uso de los bloques de MATLAB Simulink.

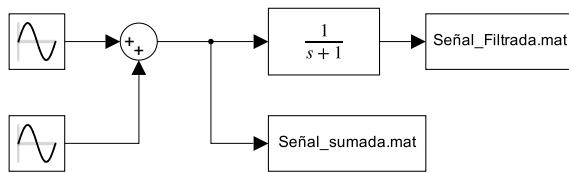


Figura 4.12: Diagrama caso de estudio 1 - Filtro Básico

Para la construcción del caso de estudio sé propone el diagrama de la Figura 4.12, donde para su construcción se llevaran a cabo los siguientes pasos.

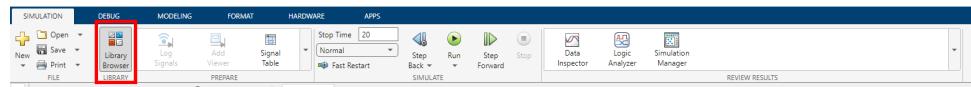


Figura 4.13: Librería de bloques

Primeramente se deberá de tener acceso a la librería de bloques, la misma se puede encontrar en la ubicación que se muestra en la Figura 4.13, a esta se debe de acceder haciendo click en el ícono que se encuentra resaltado en la Figura anteriormente mencionada.

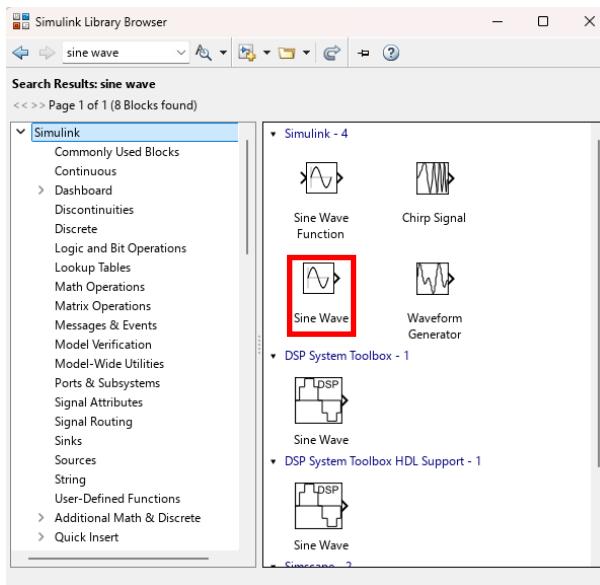


Figura 4.14: Librería de bloques - generador de onda senoidal

Una vez en la librería de bloques, como se puede observar en la Figura 4.14 se debe ingresar en el buscador marcado en color verde la palabra clave del bloque que se desea buscar, para este caso se busca el generador de onda seno del cual se ocupan dos bloques.

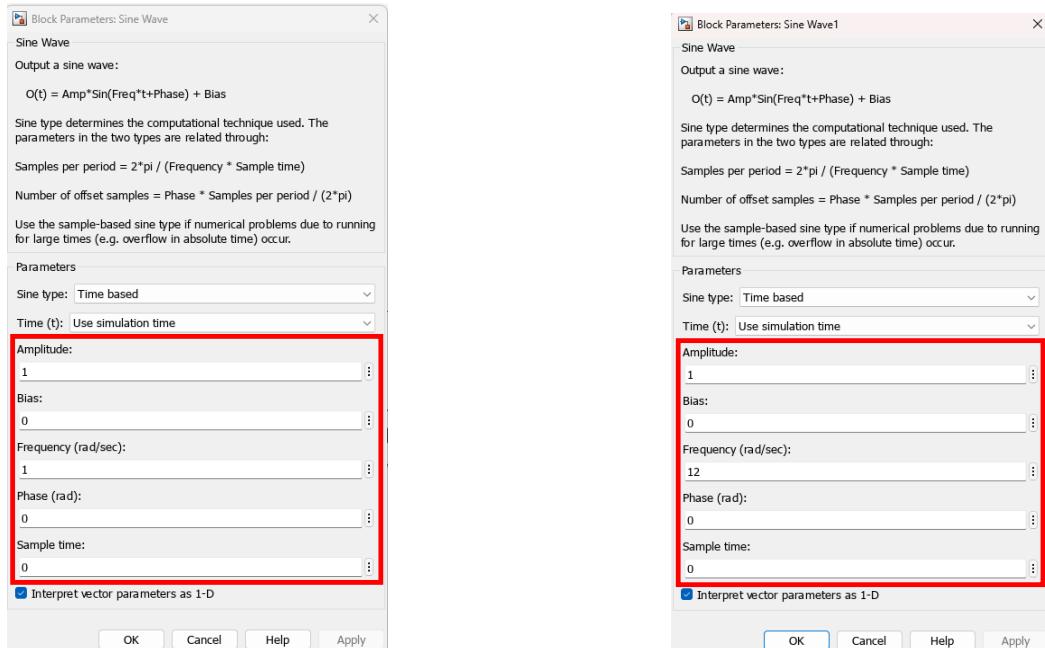


Figura 4.15: Configuración de los generadores de onda senoidal

Para la configuración de los generadores de onda senoidales se deben de seguir las configuraciones que se muestran en la Figura 4.15, de forma que al primer bloque generador de onda seno se le debe colocar la configuración que se muestra en 4.15a y al segundo la que se muestra en 4.15b. De esta forma quedan configurados los bloques encargados de generar las ondas según los parámetros indicados.

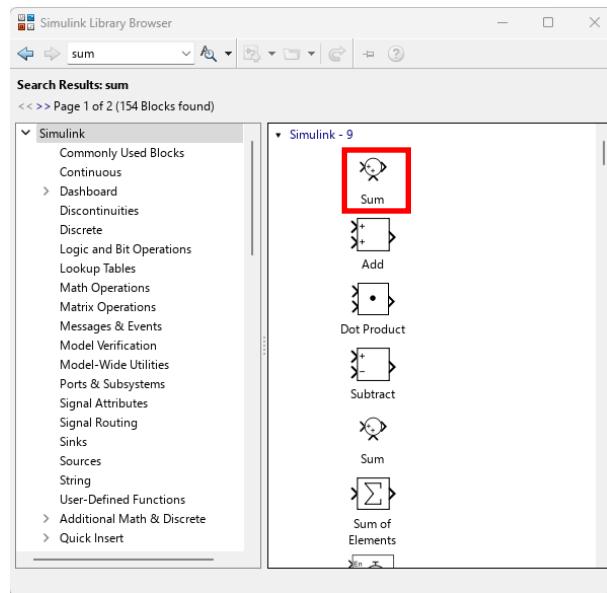


Figura 4.16: Librería de bloques - sumador de señales

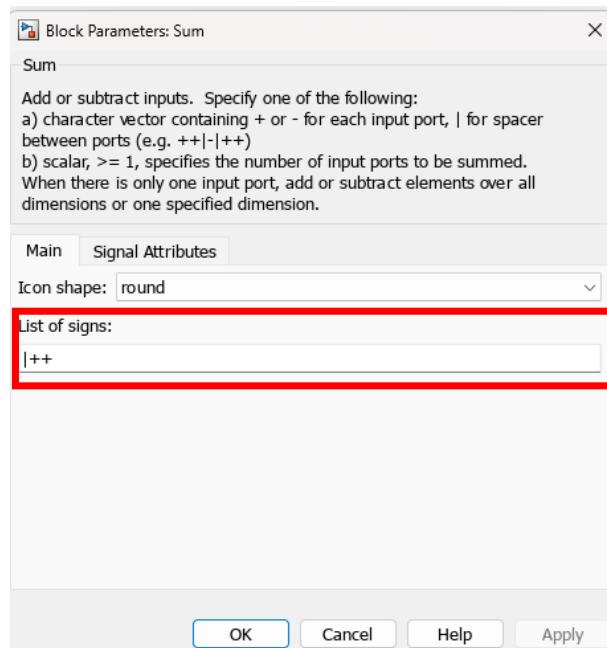


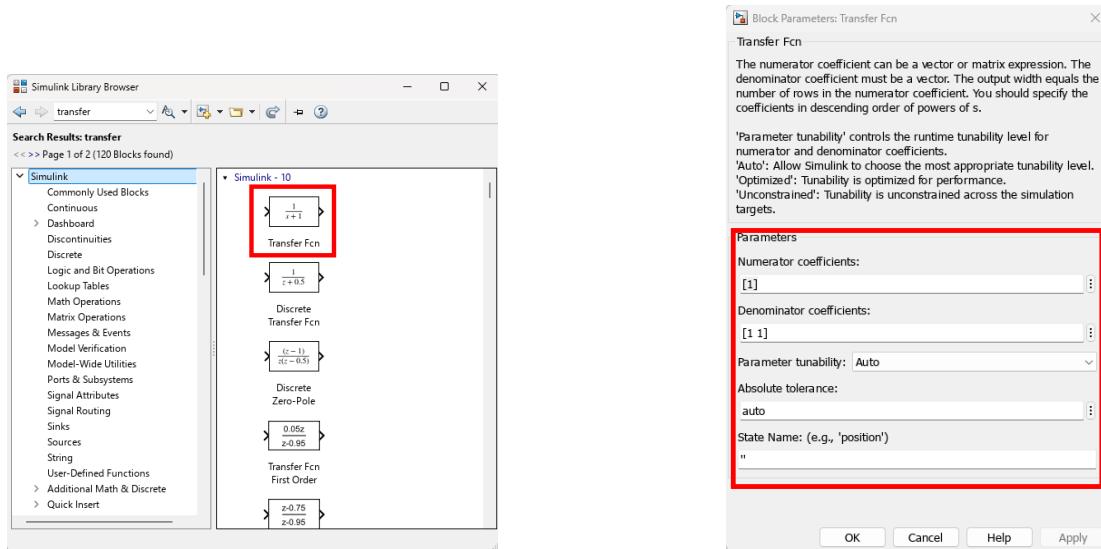
Figura 4.17: Configuración del bloque sumador de señales

Seguido de esto se debe de implementar el bloque sumador el cual se muestra en la Figura 4.16, este mismo se debe de configurar como se muestra en la Figura 4.17, este bloque nos ayudara a sumar las dos señales senoidales, ya que al sumar ondas de diferentes frecuencias, se puede observar un fenómeno llamado modulación, donde la onda resultante presenta un patrón que varía en el tiempo.

$$y(t) = \sin(t) + \sin(12t) \quad (4.1)$$

Por otro lado la función de transferencia a utilizar en el filtro será:

$$H(S) = \frac{1}{S + 1} \quad (4.2)$$



(a) Librería de bloques - función de transferencia

(b) Configuración del bloque función de transferencia

Figura 4.18: Bloque función de transferencia

Como se puede observar en la Figura 4.18, podemos observar a la izquierda en 4.18a el bloque de función de transferencia en la librería de bloques, por otro lado a la derecha en 4.18b, podemos observar la configuración que debemos de colocar al mismo, esto con el fin que ejecute la función que se muestra en 4.2. De esta forma al aplicar el filtro a la señal compuesta, la onda $\sin(t)$ pasará a través del filtro con poca atenuación, mientras que la onda $\sin(12t)$ será significativamente atenuada debido a su alta frecuencia.

Estos bloques mencionados anteriormente se colocan como se muestra en la Figura 4.12 de modo que se obtienen como salida del sistema dos archivos, uno llamado señal sumada el cual contiene los datos crudos de la modulación de las dos señales senoidales y otro denominado señal filtrada el cual contiene los datos de la señal filtrada por la función de transferencia.

4.2.1. Simulación del caso de estudio en MATLAB Simulink

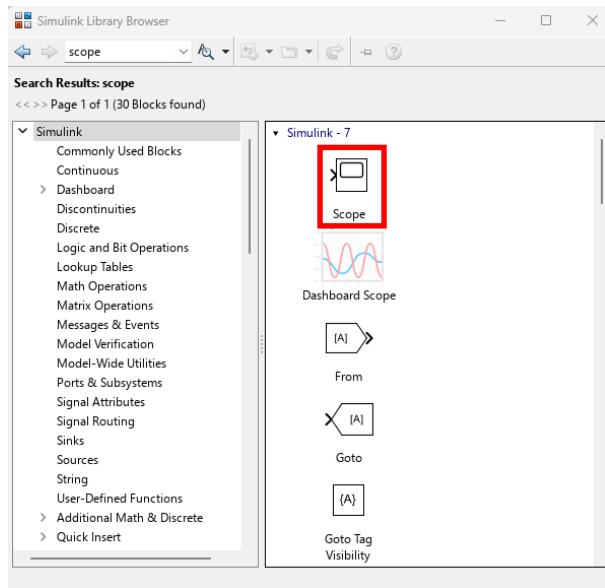


Figura 4.19: Librería de bloques - gráfico

Para la simulación del caso de estudio que se muestra en la Figura 4.12 se requiere un bloque adicional, el cual se encarga de mostrar de forma grafica los resultados de la ejecución, el bloque se muestra en la Figura 4.19.

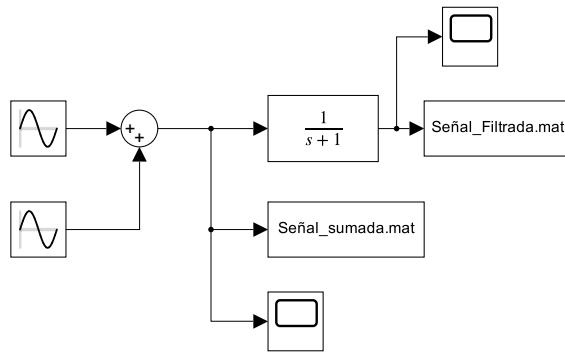


Figura 4.20: Diagrama MATLAB Simulink para poder observar las salidas

De esta forma el diagrama de la Figura 4.12, se colocan dos bloques de gráfico en el diagrama como se muestra en la Figura 4.20, esto con el objetivo de poder observar las señales de salida en cada uno de los puntos de interés.

Resultados obtenidos con la ejecución de la simulación

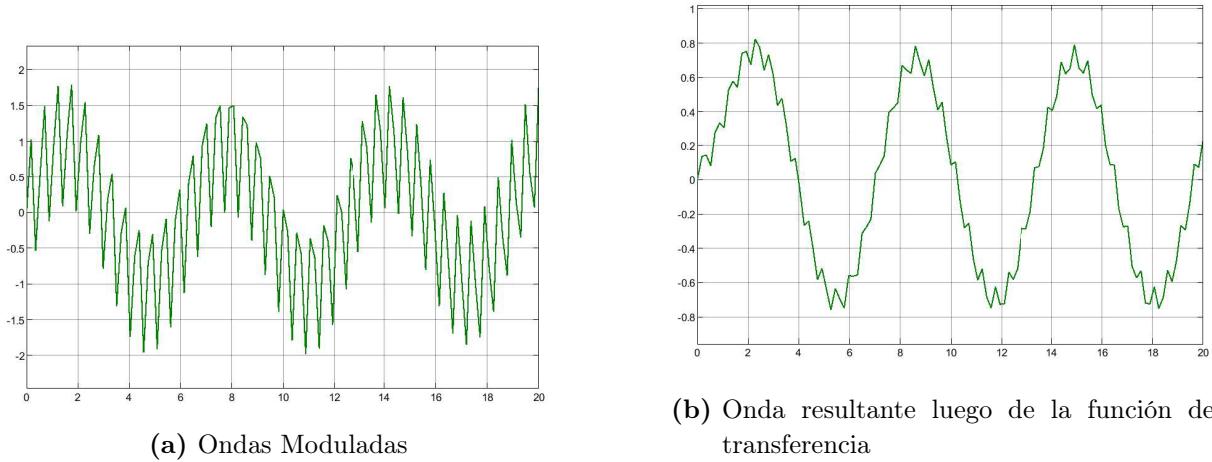


Figura 4.21: Salida simulada del diagrama mostrado en la Figura 4.20

En la Figura 4.21a se puede observar la modulación de las dos señales senoidales, por otro lado en la Figura 4.21b se puede observar la salida de la función de transferencia. Siendo la salida esperada de la función de transferencia, ya que al ser un filtro paso bajo atenúa las señales que estén por debajo de la frecuencia de corte, que para este filtro es de 1 rad/s . Como la señal compuesta contiene una onda seno con frecuencia de 1 rad/s y otra con frecuencia de 12 rad/s es posible observar aun componentes de la frecuencia atenuada.

4.3. Flujo de trabajo de la aplicación de transformación de modelo a modelo

Para poder cumplir con el objetivo de embeber el sistema, se debe de hacer uso de la herramienta MATLAB Simulink Coder. Como se mencionó en 2.6, a este tipo de aplicaciones se les conoce como transformadores de modelos, en este caso se emplea para lograr transformar el diagrama de control implementado en MATLAB Simulink en un código de lenguaje C, manteniendo de esta forma la estructura del modelo, pero realizando una adaptación al formato requerido, este convertidor de modelos garantiza la consistencia del modelo de control además de realizar la adaptación del mismo a las características de la tarjeta de desarrollo ZedBoard. Algunos de los parámetros que se pueden configurar en este transformador de modelos son: parámetros de la solución, implementación en hardware y generación de código.

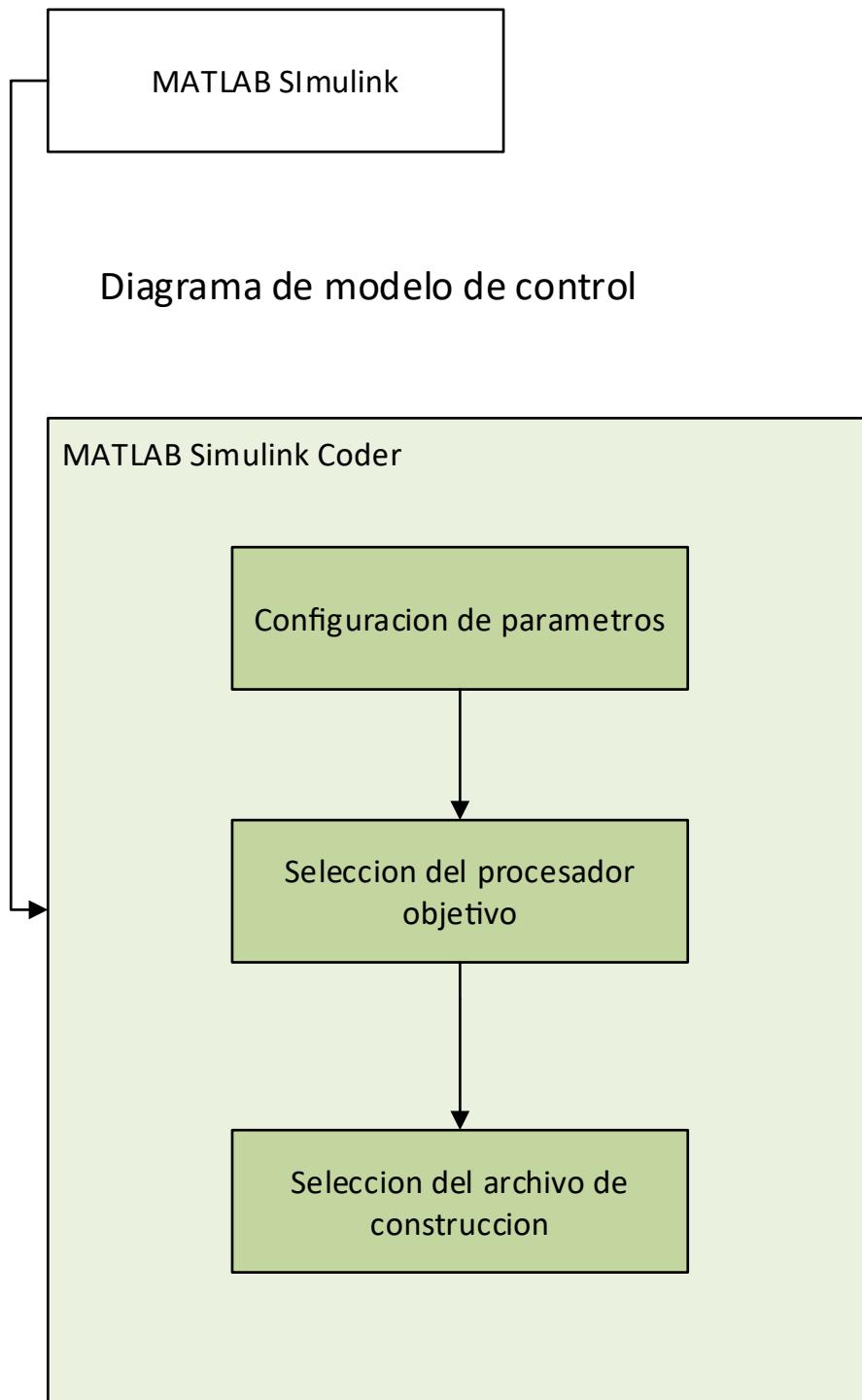


Figura 4.22: Flujo de trabajo MATLAB Simulink Coder

En la Figura 4.22 se puede encontrar un diagrama que contempla los pasos a seguir en esta sección, los cuales se explicaran a lo largo de este capítulo, además, se definirán los parámetros que se deben de utilizar y el funcionamiento de estos dentro de la generación del código C.

4.3.1. Simulink Coder

Una vez comprobado el comportamiento esperado por el caso de estudio mediante la simulación, se puede proceder con la ejecución del flujo de trabajo de MATLAB Simulink Coder, esto con el fin de transformar el modelo generado en MATLAB Simulink a un modelo de lenguaje de programación C. Cabe destacar que para esta implementación se utilizó el diagrama que se muestra en la Figura 4.12, ya que, este solamente contiene como salida los archivos con los datos numéricos del sistema y no contiene las salidas gráficas agregadas en 4.2.1.

4.3.2. Definición de parámetros

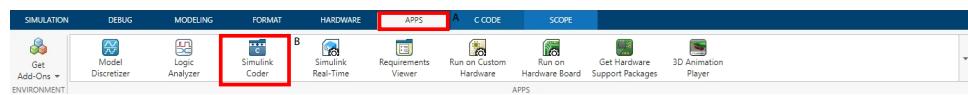


Figura 4.23: Pestaña Aplicaciones



Figura 4.24: Pestaña código C

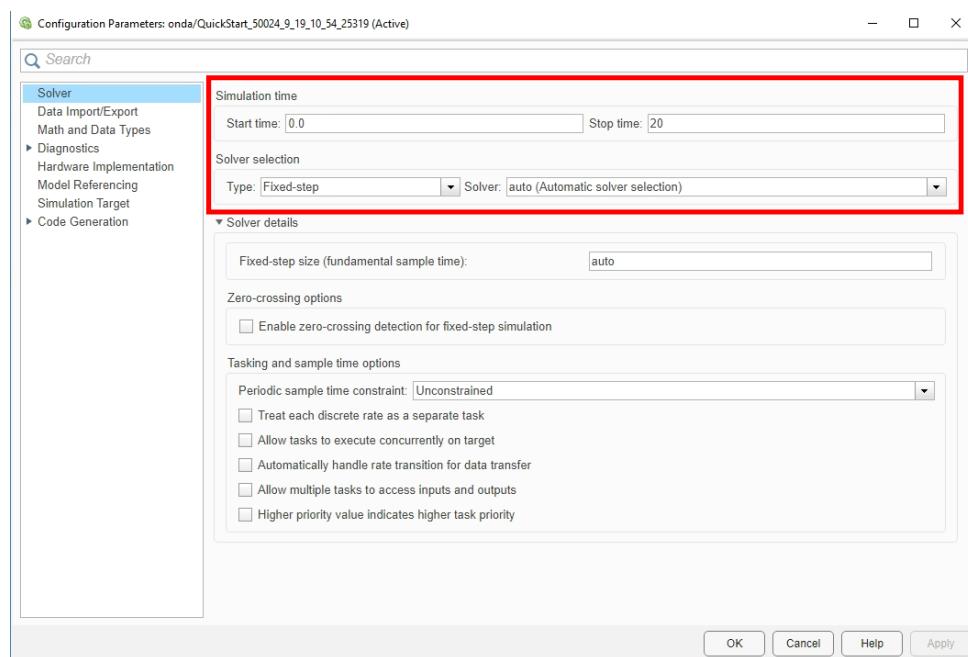


Figura 4.25: Configuración de parámetros

Para la definición de parámetros, se debe de estar en el entorno de MATLAB Simulink, una vez en el entorno mencionado anteriormente se debe ir a la pestaña denominada

Aplicaciones, o bien APPS como se muestra en la Figura 4.23, se deberá de seleccionar la aplicación denominada Simulink Coder, cuando seleccionamos esta opción se abrirá una pestaña llamada código C, o bien C CODE como se pudo observar en la Figura 4.24.

Una vez estemos en la pestaña de código C, debemos de ir a la opción de configuración de parámetros, en la Figura 4.24 se observa esta opción bajo el nombre de settings, una vez presionada la opción se abre una ventana emergente como la que se muestra en la Figura 4.25, en la pestaña denominada Solver se deberán de proporcionar los datos sobre el tiempo de ejecución de la prueba.

Selección del procesador objetivo

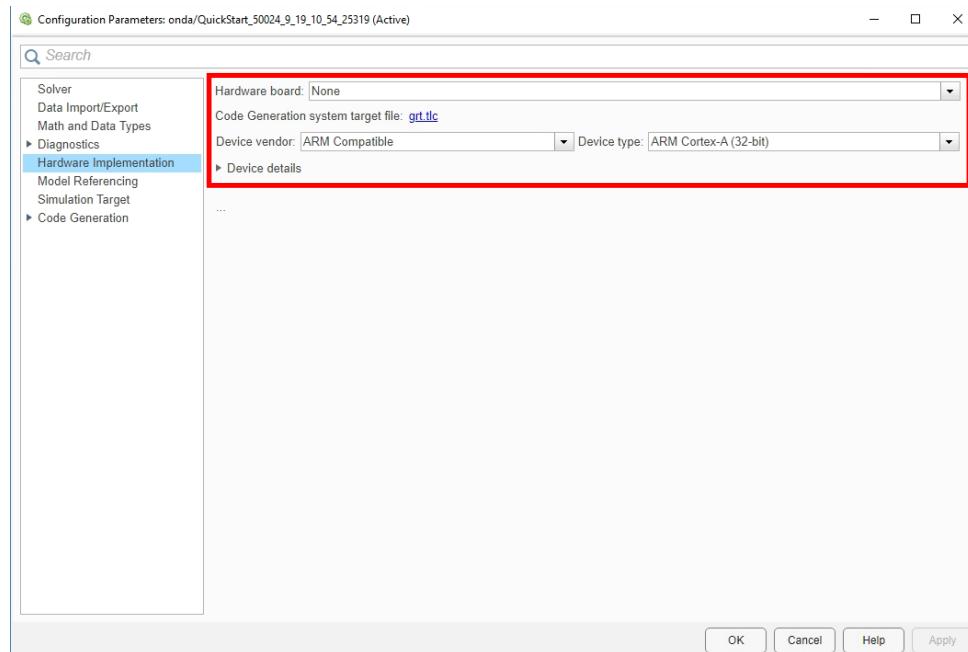


Figura 4.26: Selección del procesador y la familia del procesador

Continuando en la sección de configuración de parámetros ahora debemos de ir a la pestaña llamada implementación de hardware o bien Hardware Implementation, en donde deberemos de colocar los datos de Device Vendor el cual hace referencia al tipo de procesador que contiene la tarjeta de desarrollo, para nuestro caso sería ARM Compatible y el Device Type que seria a la familia que pertenece el procesador, para nuestro caso sería un ARM Cortex-A de 32-bits tal y como se muestra en la Figura 4.26.

Selección del tipo de archivo de construcción

Anteriormente configuramos los parámetros de tiempo de operación y procesador de la tarjeta de desarrollo, ahora debemos de configurar el tipo de archivo que se utilizará para la generación de los archivos binarios, como se deberá de realizar una compilación cruzada se debe de elegir un tipo de archivo el cual nos permita compilar los binarios para

la ejecución del sistema sin importar el sistema operativo de la máquina host. Es por esto que se debe de seleccionar en la pestaña de Code Generation el Toolchain denominado CMake tal y como se muestra en la Figura 4.27, además de esto se debe de marcar tanto la opción denominada como Generate code only, como Package code and artifacts, esta última nos genera como salida un archivo comprimido con todos los requerimientos de la aplicación para poder ser construida.

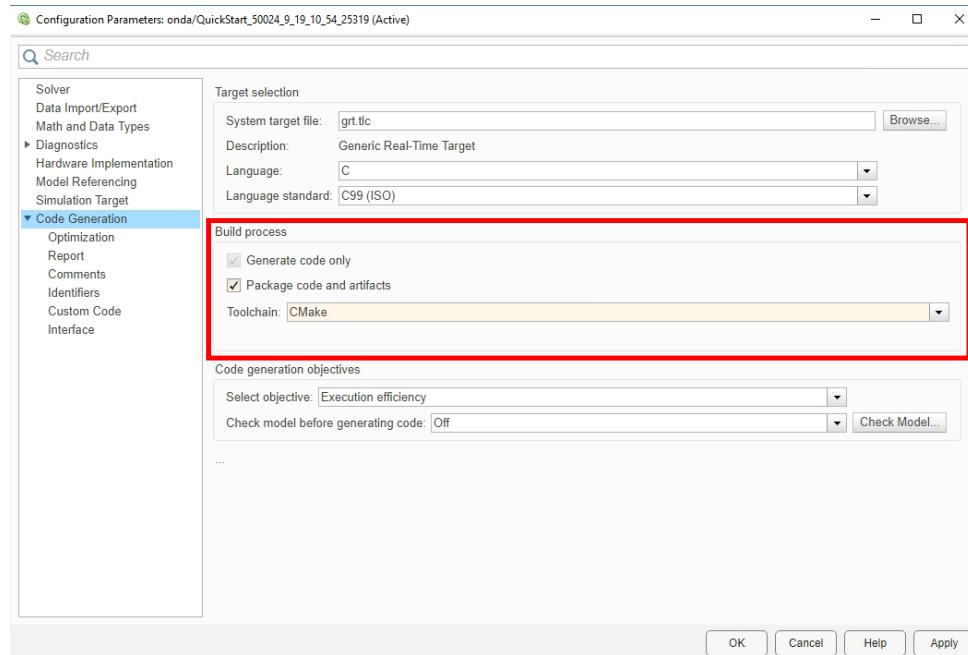


Figura 4.27: Selección del tipo de archivo de construcción

Generación de archivos de compilación

Una vez configurados todos los parámetros mencionados anteriormente debemos de proceder con la construcción de los archivos, para esto se debe de ir a la barra de tareas a la opción denominada como generar código, la misma se puede observar en la Figura 4.24 bajo el nombre de Build.

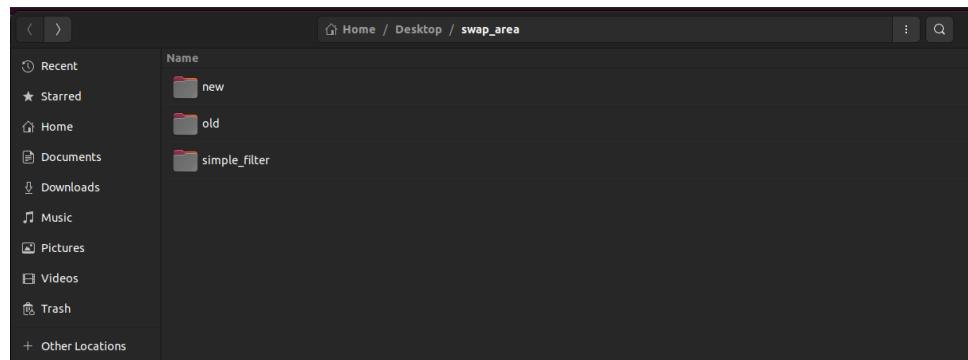


Figura 4.28: Archivo comprimido en el directorio swap_area

```

1  sudo docker cp /direccion/del/archivo
2  <id_de_contenedor>:/direccion/del/contenedor

```

Listing 4.16: Copiar archivos al contenedor, Linux

```

1  sudo docker cp <id_de_contenedor>:/direccion/del/contenedor
2  /direccion/del/archivo

```

Listing 4.17: Copiar archivos del contenedor, Linux

Seguido de esto se debe de copiar el archivo comprimido generado en [4.3.1](#), al contenedor con Ubuntu 16.04 generado en [4.1.2](#). Primeramente colocaremos el archivo comprimido en un directorio llamado swap_area, tal y como se muestra en [4.28](#). Seguido de esto se debe de descomprimir el archivo. Una vez descomprimido el archivo, como se mencionó anteriormente, lo enviaremos al contenedor haciendo uso del comando [4.16](#).

4.3.3. Compilación del Código C generado

```

1  cmake -DCMAKE_C_COMPILER=arm-linux-gnueabihf-gcc
2  CMakeLists.txt -DMATLAB_ROOT=/home/test/simple_filter/R2024b/

```

Listing 4.18: Compilacion del programa, Linux

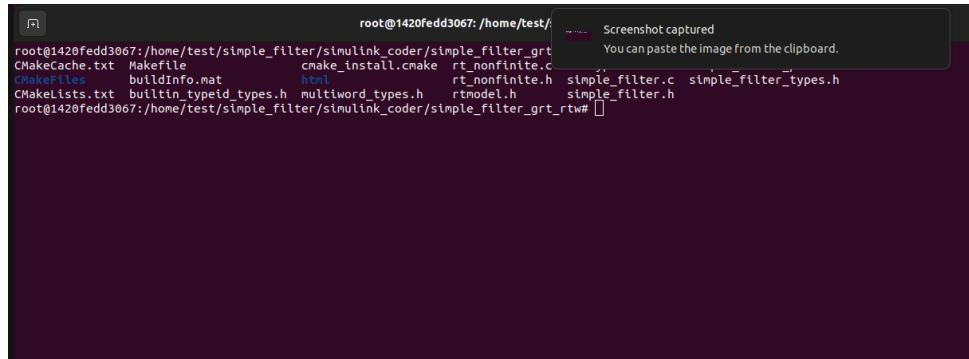


Figura 4.29: Make File

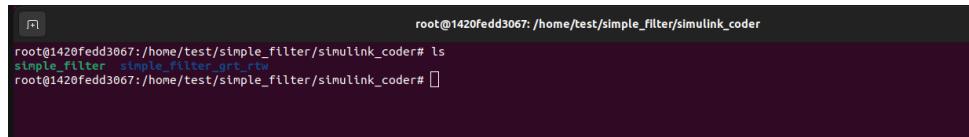


Figura 4.30: Binario llamado simple_filter

Para la compilación del archivo binario se deberá hacer uso del comando [4.18](#) por el cual se construirá el Makefile, tal y como se muestra en la Figura [4.29](#), una vez generado el Makefile se ejecutó el comando (Make) el cual da como salida los binarios requeridos para la ejecución del programa. Los mismos se observan como se muestra en la Figura [4.30](#).

Una vez compilado el archivo binario, se puede continuar con el flujo que se presenta en el diagrama que se muestra en la Figura [4.1](#), lo cual sería la implementación de los binarios en una imagen de yocto.

4.4. Flujo de Trabajo EmbedSynthGNC

Como se pudo observar anteriormente se realizó la compilación cruzada de un caso de estudio, el mismo ahora se debe de implementar en un sistema operativo a la medida mediante el flujo de trabajo de Yocto Project, como se mencionó en 2.4.1, Yocto Project es un marco de trabajo utilizado para el desarrollo de sistemas embebidos especializado en la construcción de distribuciones de Linux a la medida, cabe destacar que el flujo de trabajo a seguir se explica brevemente en la Figura 4.31.

En el desarrollo de esta sección se muestran los pasos que se siguieron para la generación de una imagen, la integración de una capa personalizada con el binario generado en 4.3.3 y la implementación de la misma en la tarjeta de desarrollo seleccionada.

4.4.1. Creación de una capa de yocto

```
1 pwd
```

Listing 4.19: "Print Working Directory",Linux

```
1 source oe-init-build-env build/
```

Listing 4.20: Inicializar ambiente, Yocto

Para la generación de una capa de yocto primero debemos de estar seguros que nos encontramos en el directorio denominado POKY, esto lo podemos verificar por medio del uso del comando 4.19, seguido de esto se debe de inicializar el entorno de desarrollo esto mediante el comando que se muestra en 4.20, este se encarga de inicializar el entorno así también como todos los requerimientos necesarios para poder hacer uso de las variables de entorno con las cuales opera el marco de trabajo.

```
1 bitbake-layers create-layer <nombre-de-la-capa>
```

Listing 4.21: Generar nueva capa, Yocto

```
1 bitbake-layers add-layer ../<nombre-de-la-capa>
```

Listing 4.22: Agregar nueva capa, Yocto

Seguido de esto se debe de utilizar el comando que se muestra en 4.21 el cual se encargara de generar el árbol de directorios que se puede observar en la Figura 4.32. Una vez implementado este comando se debe de hacer uso del comando que se muestra en 4.22 para poder agregar la capa al archivo denominado bblayers.conf el cual contiene todas las rutas de acceso a las capas requeridas para generar la imagen. Para este caso de estudio se debe de generar una capa con el nombre de (meta-embedsynthGNC).

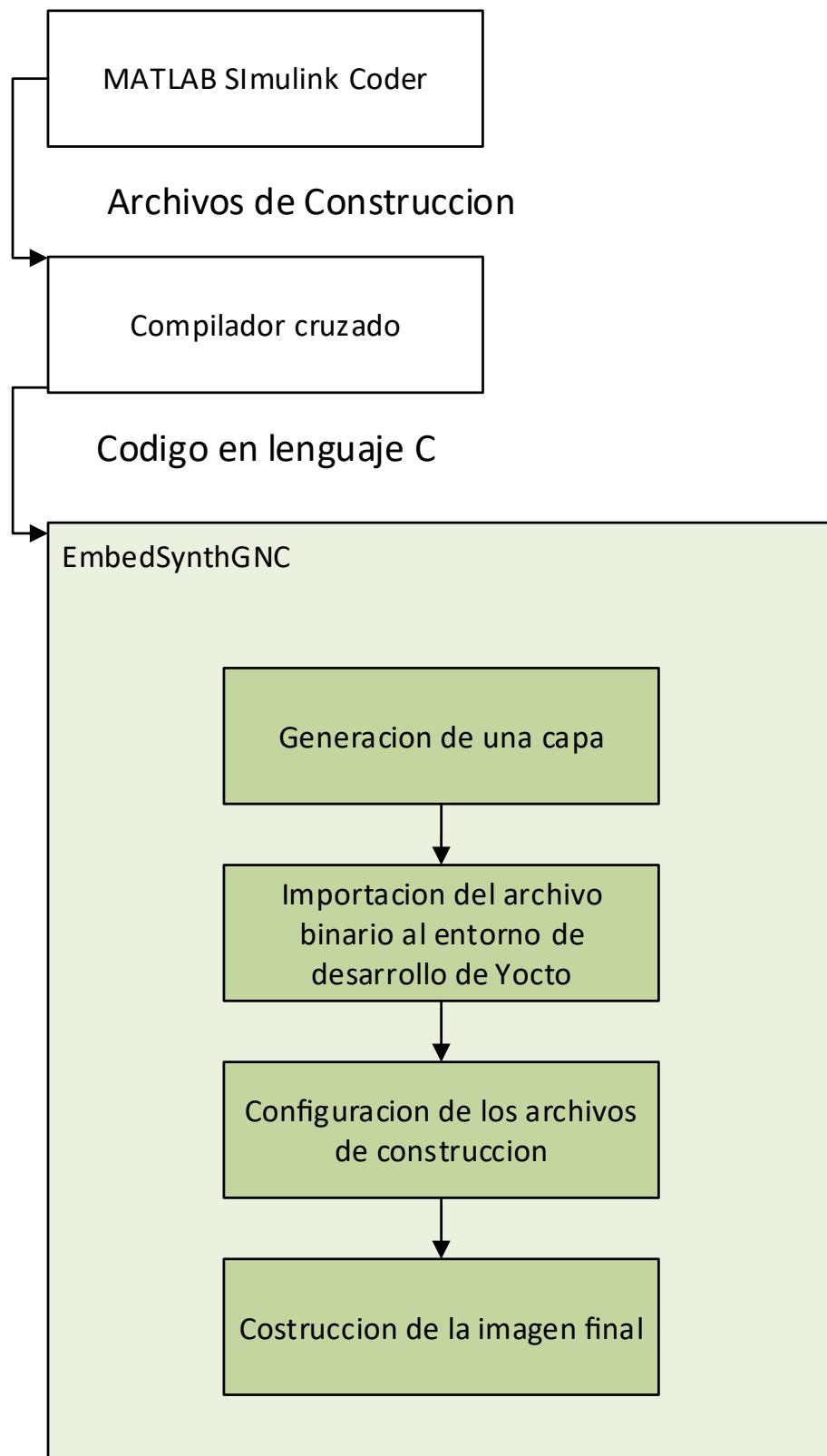


Figura 4.31: Flujo de trabajo EmbedSynthGNC

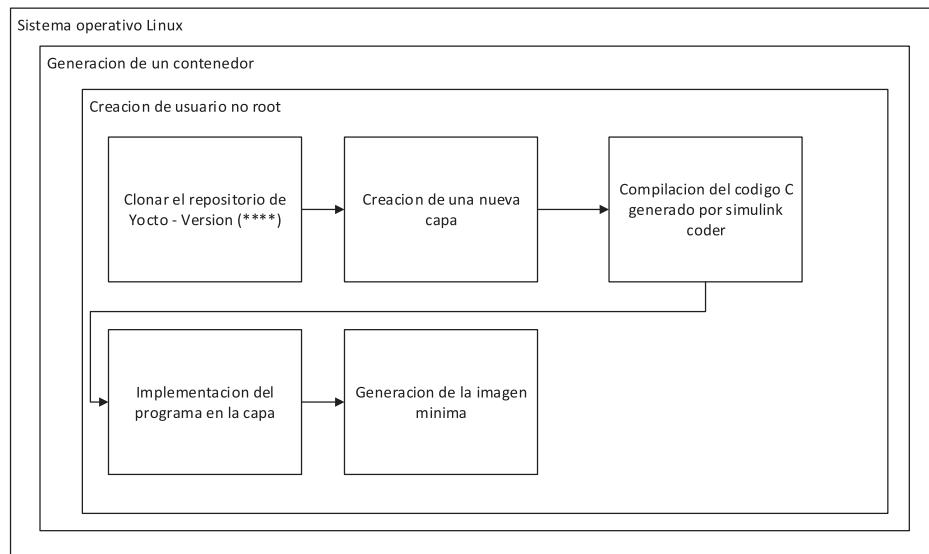


Figura 4.32: Árbol de directorios de la capa

4.4.2. Caso de estudio 1 - Filtro Básico en ZedBoard

En esta sección se integrará el caso de estudio generado en 4.3.3, a un sistema operativo a la medida por medio del marco de trabajo de Yocto Project.

4.4.3. Integración del programa generado a la capa de Yocto

Para la implementación del binario generado en 4.3.3, se deben de generar algunos directorios, esto con el objetivo de mantener un entorno de trabajo limpio y ordenado. Para contener todos los directorios que se deben de crear, se genera un directorio llamado (recipes-core) el cual dentro del mismo deberá de contener un directorio llamado (caso_de_estudio_1) el cual se encargara de contener el archivo de configuración de la capa llamado (caso_de_estudio_1.bb); este archivo se genera mediante el comando que se observa en 4.21, además de generar este archivo se debe de crear un directorio llamado "files" que será el encargado de contener el archivo binario compilado en 4.3.3.

caso_de_estudio_1.bb

Como se mencionó anteriormente el archivo llamado sistema_control.bb es el encargado de la configuración de la capa, el mismo contiene los comandos de instalación y la dirección en donde se encontrara el archivo binario en el sistema de archivos de la imagen del sistema embebido.

La estructura que debe de contener ese archivo para instalar binarios en el sistema son las que se pueden observar en la Figura 4.33.

```

DESCRIPTION = "Simulink App precompiled binary"
LICENSE = "CLOSED"
SRC_URI = "file://simple_filter"

S = "${WORKDIR}"

do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${WORKDIR}/simple_filter ${D}${bindir}/simple_filter
}

RDEPENDS_${PN} += "glibc"

```

Figura 4.33: Estructura del archivo caso_de_estudio_1.bb

4.4.4. Generación de la imagen

```
1 bitbake-layers add-layer ../<nombre-de-la-capa>
```

Listing 4.23: Generar archivos de desarrollador, Yocto

Antes de generar la imagen mínima se debe de tener en consideración ejecutar la línea de comando que se muestra en 4.23, esto con el fin de generar archivos de desarrollo en lugar de archivos de imagen en formato iso. Una vez generados estos cambios se debe de iniciar de nuevo el entorno por medio del comando 4.20, seguido de esto se deberá de ejecutar el comando de "bitbake core-image-minimal.el" el cual se encarga de comenzar a generar la imagen mínima.

```
1 IMAGE_INSTALL_APPEND+= caso\_de\_estudio\_1
```

Listing 4.24: Instalar la capa generada, Yocto

Finalmente para la instalación de la capa dentro de la imagen se debe de editar el archivo de configuracion local llamado local.conf el cual se encuentra en la direccion build/conf/local.conf en el cual se deberá de agregar la linea que se muestra en 4.24 , esto con el fin de agregar la capa generada anteriormente al sistema de archivos de la imagen a generar.

4.4.5. Implementación de la imagen mínima en la tarjeta de desarrollo Zedboard

Para la implementación de la imagen mínima desarrollada en 4.4.4 en la tarjeta de desarrollo, se deben de desarrollar los siguientes pasos en la máquina Host, Primeramente se debe de formatear la tarjeta SD de al menos 4 GB, las particiones de la misma se tienen que observar de la siguiente forma:

- raíz = 100 MB FAT 32
- sistema de archivos = 3.5 GB ext6(linux filesystem format)

Seguido de esto se debe de ir a la ruta (ruta donde se encuentran los archivos de imagen de la zedboard), mientras que en la máquina host se debe de ir a la ruta seleccionada para almacenar los archivos temporalmente y se deben de copiar los archivos del contenedor a la máquina host mediante el comando que se muestra en 4.17, esto con el objetivo de poder enviar los archivos a la tarjeta SD más adelante.

```
1 sudo cp boot.bin boot.scr
2 core-image-minimal-zedboard-zynq7.cpio.gz.u-boot
3 u-boot.img uEnv.txt uImage zynq-zed.dtb /media/root
```

Listing 4.25: Copiar archivos root, Linux

```
1 sudo cp core-image-minimal-zedboard-zynq7.tar.gz /mnt/partition2
```

Listing 4.26: Copiar sistema de archivos, Linux

Para el sistema Root se deberán de copiar los archivos mediante el comando que se muestra en 4.25, por otro lado en la partición denominada FileSystem se debe de copiar el archivo "core-image-minimal-zedboard-zynq7.tar.gz" mediante el uso del comando que se muestra en 4.26.

4.4.6. Conexión de la tarjeta de desarrollo con el computador host

Como protocolo de comunicación se establece primeramente el protocolo UART, el cual como se mencionó en 2.10 consiste en un protocolo de comunicación serie que permite la transmisión y recepción de datos de manera asíncrona entre dos dispositivos y en el caso de la tarjeta de desarrollo se conecta según se muestra en el diagrama de la Figura 4.34, mediante el uso del puerto marcado como "S.en el diagrama. Para poder leer la consola se hace uso de Minicom el cual se encarga de la emulación de terminal en Linux que permite la comunicación serie con dispositivos a través de puertos seriales.

Una vez que se estableció la conexión por medio de SSH el cual como se menciona en 2.10, consiste en protocolo de comunicación en red que permite el acceso remoto seguro a sistemas, proporcionando autenticación y encriptación de datos, ya que es mejor que UART para comunicaciones remotas porque proporciona autenticación y encriptación, garantizando la seguridad de los datos transmitidos, mientras que UART es un protocolo simple y sin mecanismos de seguridad, adecuado solo para comunicaciones locales y de corto alcance.

El diagrama de este protocolo de comunicación se puede observar en 4.34, el cual se logra mediante la conexión al puerto denominado en el diagrama como e.

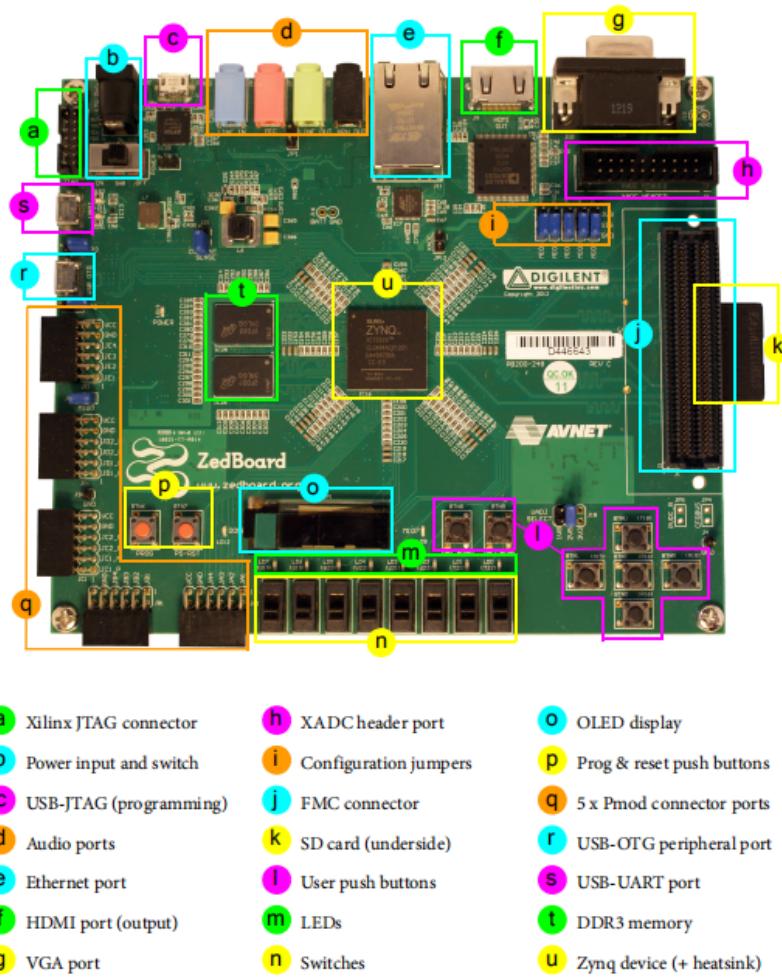


Figure 6.1: ZedBoard layout and interfaces (front)

Figura 4.34: Puertos tarjeta de desarrollo Zedboard

4.4.7. Ejecución del caso de estudio y resultados

Una vez implementada la imagen de Yocto en la tarjeta de desarrollo se puede ejecutar el caso de estudio. Para esto será necesario dirigirnos al directorio en el cual se instaló el archivo binario, el mismo se encuentra en la ruta usr-bin-simple_filter. Una vez encontrado el archivo basta con ejecutarlo mediante el uso del nombre del mismo simple_filter. Cuando el archivo se ejecuta genera dos archivos de salida llamados Raw_signal.mat y Filtered_signal.mat, mediante el uso del programa en Python desarrollado se pueden leer los archivos generados y crear gráficos a partir de los mismos. Los resultados se deben de transmitir al computador por medio del comando 4.27.

```
1 scp user@ip:/ruta/del/archivo/zedboard .
```

Listing 4.27: Copiar archivo por protocolo SSH, Linux

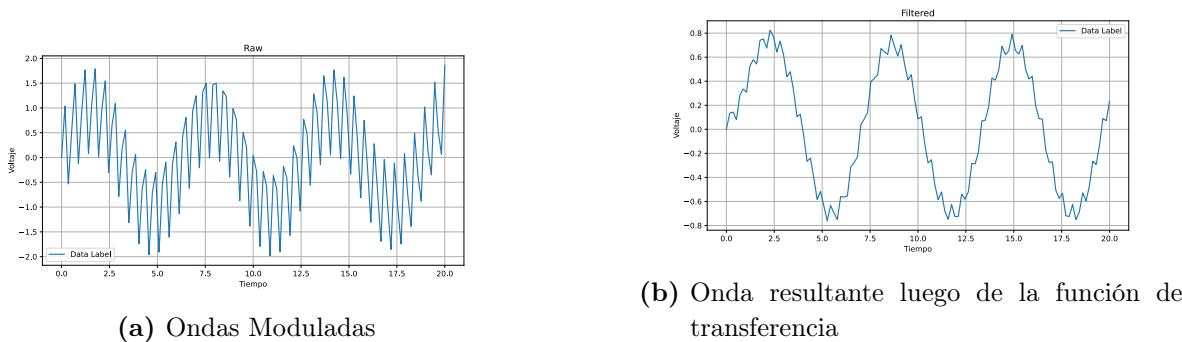


Figura 4.35: Salida resultante de la imagen generada mediante el flujo de trabajo

4.4.8. Comparación de resultados

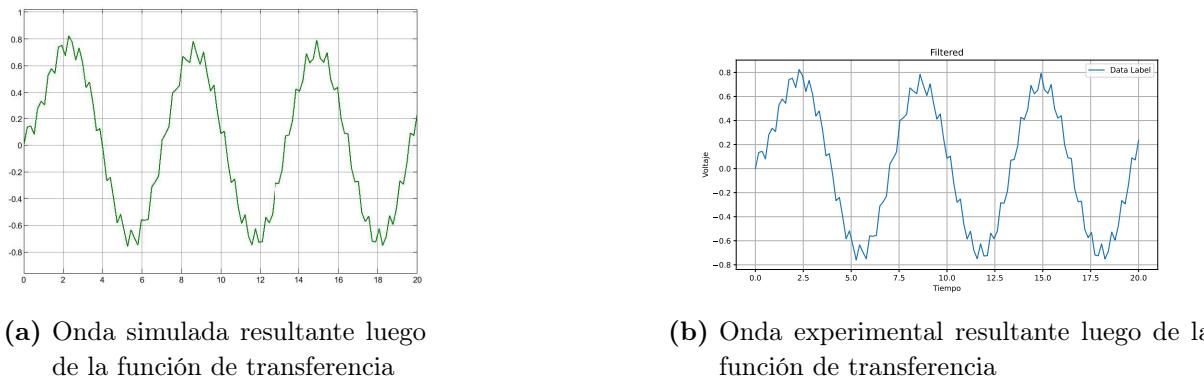


Figura 4.36: Comparación de la salida resultante simulada (izquierda) y experimental (derecha)

Como se pudo observar en la Figura 4.36, a la izquierda se puede observar la gráfica proveniente de la simulación del sistema, por otro lado en la derecha se puede observar la gráfica proveniente de la ejecución del sistema en la tarjeta de desarrollo, estas mediciones corresponden a las señales resultantes luego de la etapa de filtrado del caso de estudio desarrollado en esta sección.

Para un mejor análisis de resultados se utilizó el código de Python que se muestra en A, el cual se encarga de tomar tanto el archivo de salida de la simulación como el archivo de salida del caso experimental y compararlas, esto con el objetivo de poder medir el error promedio absoluto, el error cuadrático medio y la raíz cuadrada del mismo, esto con el objetivo de poder cuantificar las diferencias entre la señal de salida simulada y la experimental.

Para este caso de estudio se obtuvieron los siguientes resultados:

- Error Promedio Absoluto = $1,930 \times 10^{-18}$ [V]
- Error Cuadrático Medio = $2,14 \times 10^{-34}$ [V^2]
- Raíz del Error Cuadrático Medio = $1,46 \times 10^{-17}$ [V]

Los valores de error obtenidos son bajos, lo cual nos indica un alto nivel de correspondencia entre la señal simulada y la señal experimental. Por un lado tenemos el error promedio absoluto, el cual represente la diferencia promedio entre la simulación y el experimento, al ser esta métrica representada por un valor de $1,930 \times 10^{-18}$ V sugiere que las diferencias entre ambas señales son despreciables. Por otro lado, tenemos el error cuadrático medio, este es asociado con un valor de $2,14 \times 10^{-34}$ V², lo que indica que las desviaciones en las mediciones son mínimas y puntuales. Finalmente la raíz del error cuadrático medio, la cual es representada por el valor de $1,46 \times 10^{-17}$ V representa la magnitud promedio de las diferencias cuadráticas. La cercanía a cero que otorga este valor refuerza la alta precisión del modelo en relación con la señal experimental.

Al tener valores tan bajos en las métricas encargadas de medir el error de los modelos se puede respaldar que el proceso experimental es acorde a los resultados de simulación, cumpliendo con los requisitos de exactitud de forma notable.

4.5. Reflexión final

Los resultados de error obtenidos son altamente satisfactorios, lo cual respalda la implementación del caso de estudio realizada a lo largo de este capítulo. Estos valores indican una correspondencia prácticamente perfecta entre la simulación y el proceso experimental. La alta precisión alcanzada no solo valida el modelo desarrollado, sino que también resalta la consistencia y fiabilidad del proceso experimental. Es por esto que se aprueba el marco de trabajo realizado y se continúa con la investigación del mismo.

@ -1,291 +1,300 @@

Capítulo 5

Caso de estudio IMU y PID

5.1. Caso de estudio 2 - IMU

Como caso de estudio de implementación se desarrolló también la simulación de una Unidad de medición inercial, IMU por sus siglas en Ingles, se siguió el uso del modelo que se presenta en [30]. Este ejemplo muestra cómo generar y fusionar datos de sensores IMU usando MATLAB Simulink. Permitiendo modelar con precisión el comportamiento de un acelerómetro, un giroscopio y un magnetómetro, además de poder fusionar sus salidas para calcular la orientación.

Una IMU es un grupo de sensores que incluye un acelerómetro para medir aceleración y un giroscopio para medir velocidad angular. Frecuentemente, también se incluye un magnetómetro para medir el campo magnético de la Tierra. Cada uno de estos tres sensores produce una medición de tres ejes, constituyendo una medición de 9 ejes en total. Ademas de esto un Sistema de Referencia de Actitud y Rumbo (AHRS, por sus siglas en inglés) toma las lecturas de sensores de 9 ejes y calcula la orientación del dispositivo. Esta orientación se da en relación con el marco NED, donde N es la dirección del Norte Magnético. El bloque AHRS en Simulink logra esto usando una estructura de filtro de Kalman indirecto [30].

5.1.1. Implementación en MATLAB Simulink

Como se puede observar en la Figura 5.1, este es el caso de estudio que se propone en [30], a este caso de estudio se le deben de realizar unas modificaciones de acuerdo al funcionamiento deseado que se tiene para este caso de estudio, siempre generando datos en el ámbito de simulación en MATLAB para luego contrastar los mismos con los datos obtenidos en la ejecución del modelo en la tarjeta de desarrollo seleccionada.

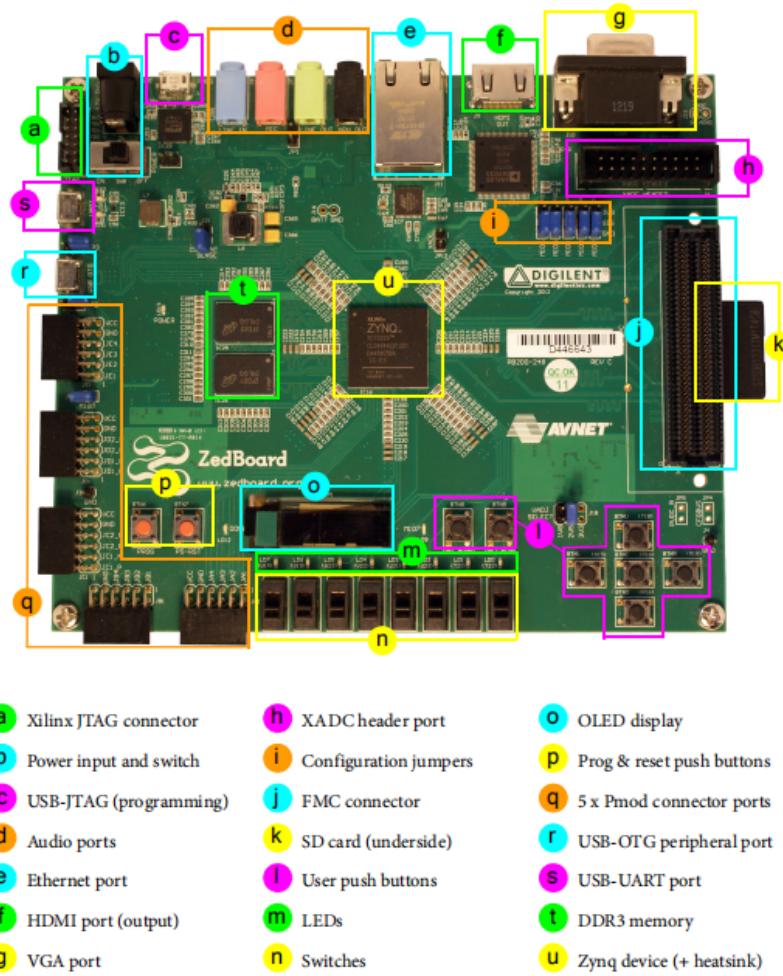


Figure 6.1: ZedBoard layout and interfaces (front)

Figura 5.1: Diagrama completo del caso de estudio 2 - IMU [30]

5.1.2. Bloques utilizados para la implementación

Los bloques utilizados se obtienen en la librería de bloques de MATLAB Simulink. A continuación se muestran los bloques requeridos, así como la configuración de los mismos para la correcta operación del modelo. La implementación del sistema se divide en dos partes, el primer parte se encarga de generar los archivos necesarios para la operación del sistema mientras que la segunda parte del sistema se encarga de leer los archivos con los datos y generar los dos archivos de salida del programa.

Sistema para la generación de archivos

Este sistema es el encargado de generar los archivos de entrada, estos mismos contienen los datos de tiempo y valores para la correcta implementación del sistema

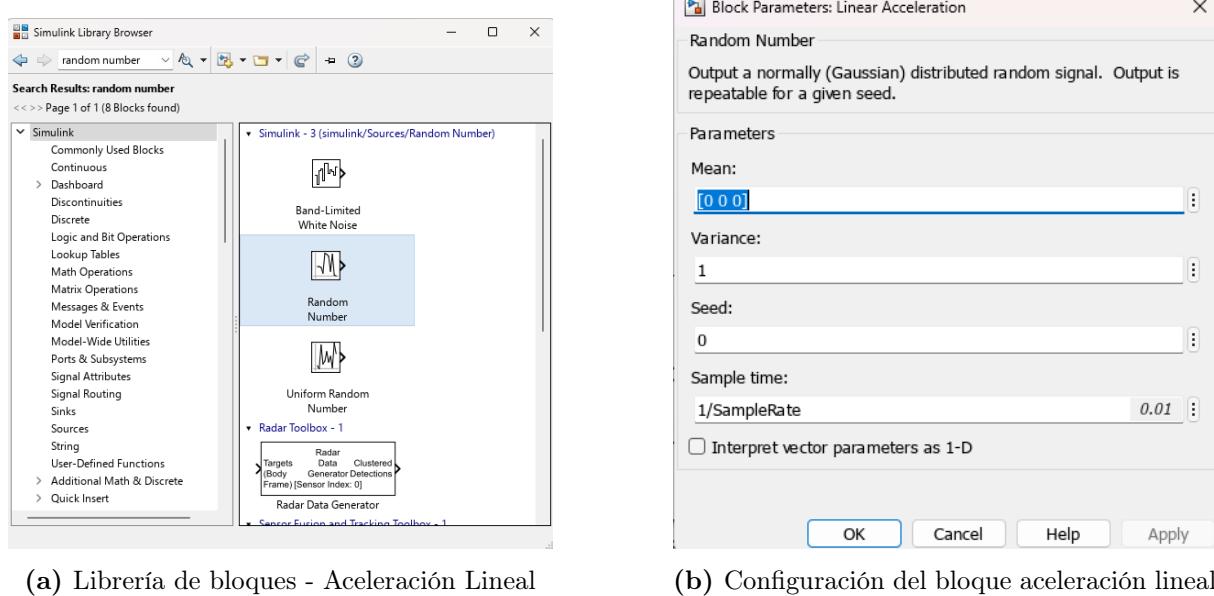


Figura 5.2: Bloque para la aceleración lineal

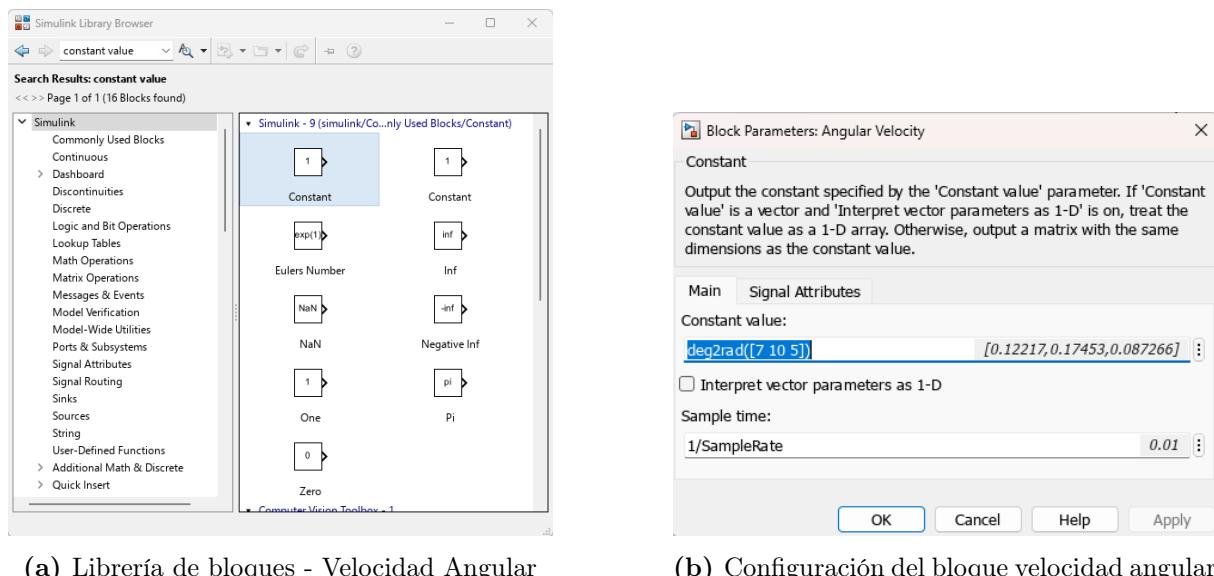


Figura 5.3: Bloque para la velocidad angular

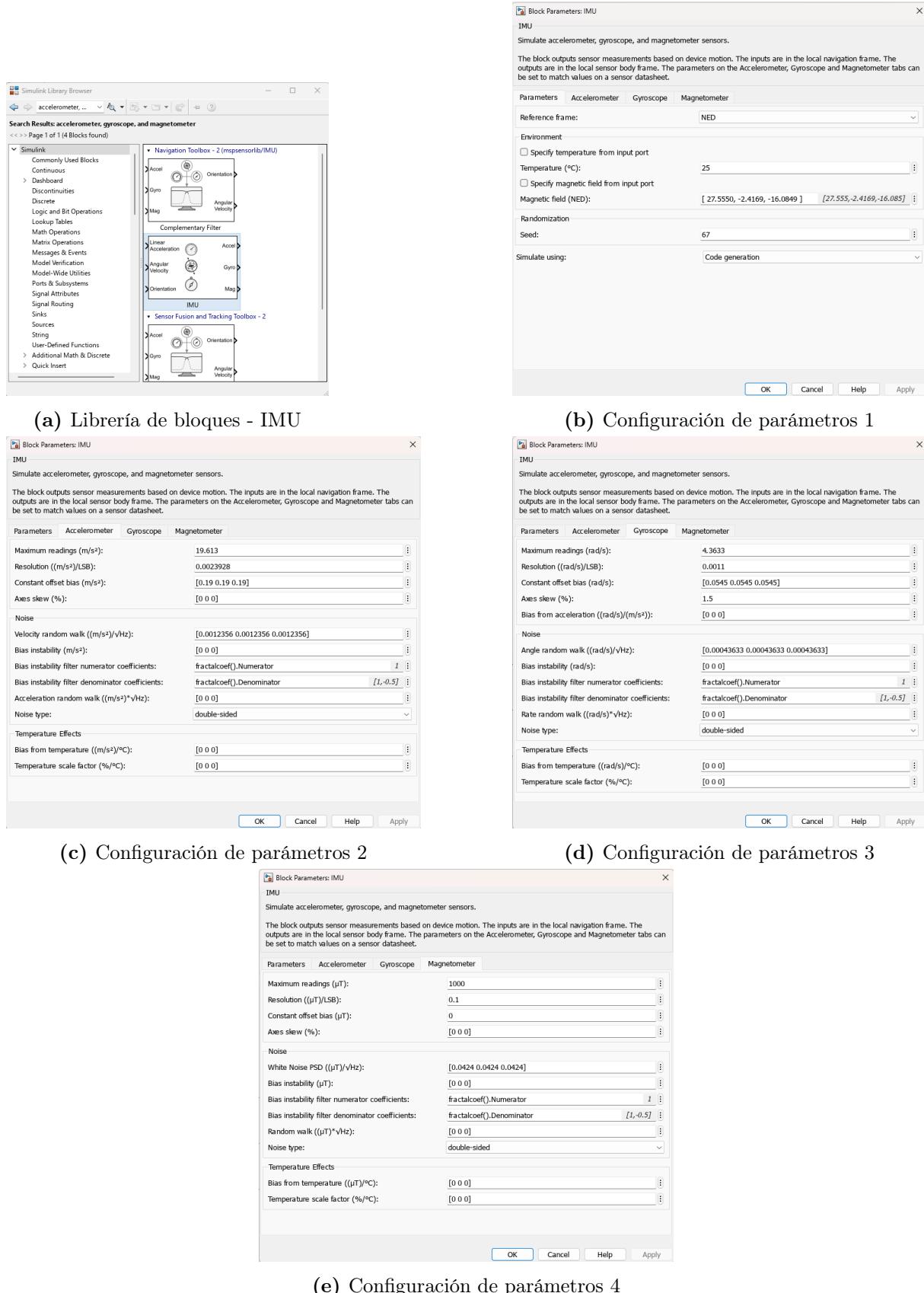


Figura 5.4: Bloque para la simulación del comportamiento de la IMU

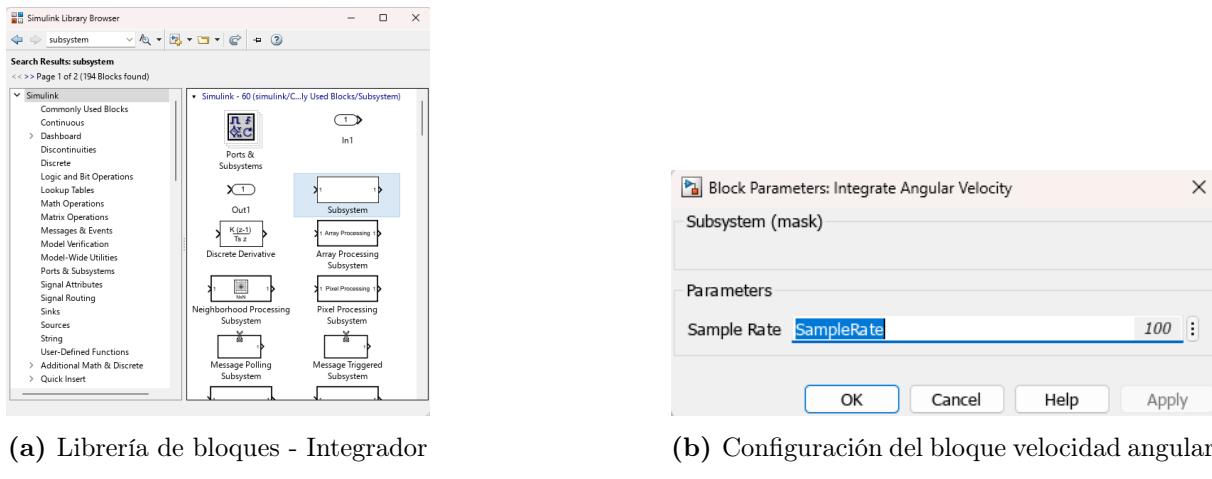


Figura 5.5: Bloque para la integración de la velocidad angular

Sistema para la lectura e interpretación de los archivos generados previamente

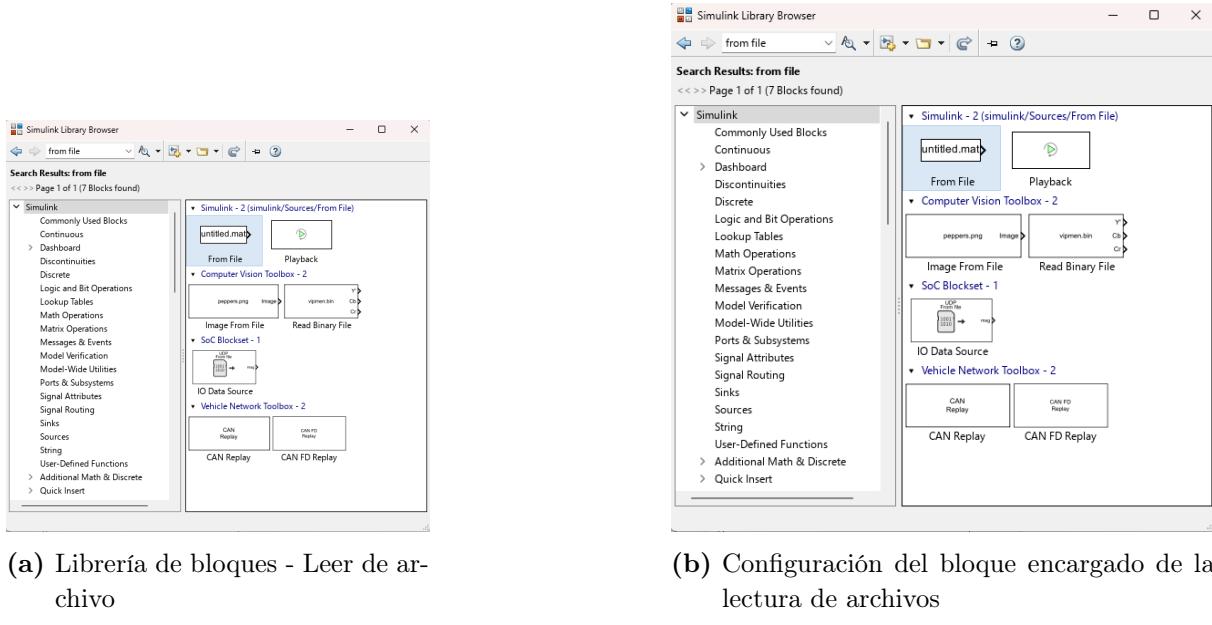


Figura 5.6: Bloque para la lectura de archivos

5.1.3. Resultados de la simulación

5.1.4. Implementación en la Tarjeta de desarrollo mediante EmbedSynthGNC

Para la implementación en la tarjeta de desarrollo ZedBoard se ejecuta el flujo de trabajo que se muestra en la sección ???. El mismo es representado mediante el diagrama que se muestra en la Figura ??.

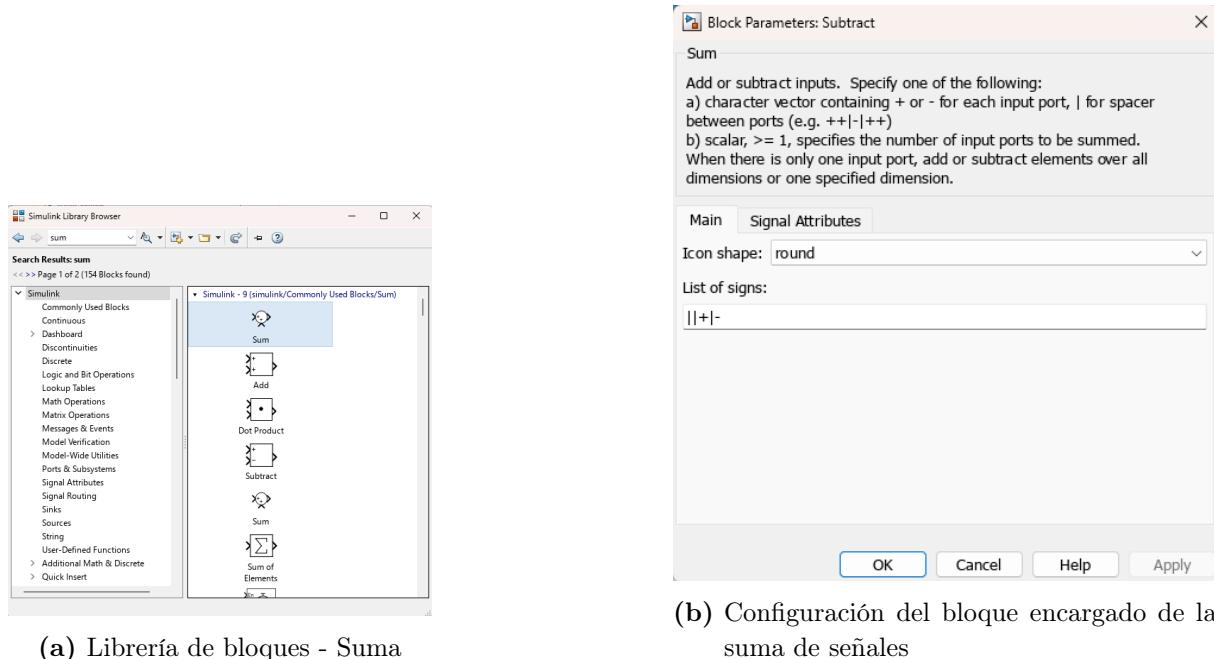


Figura 5.7: Bloque para la suma de señales

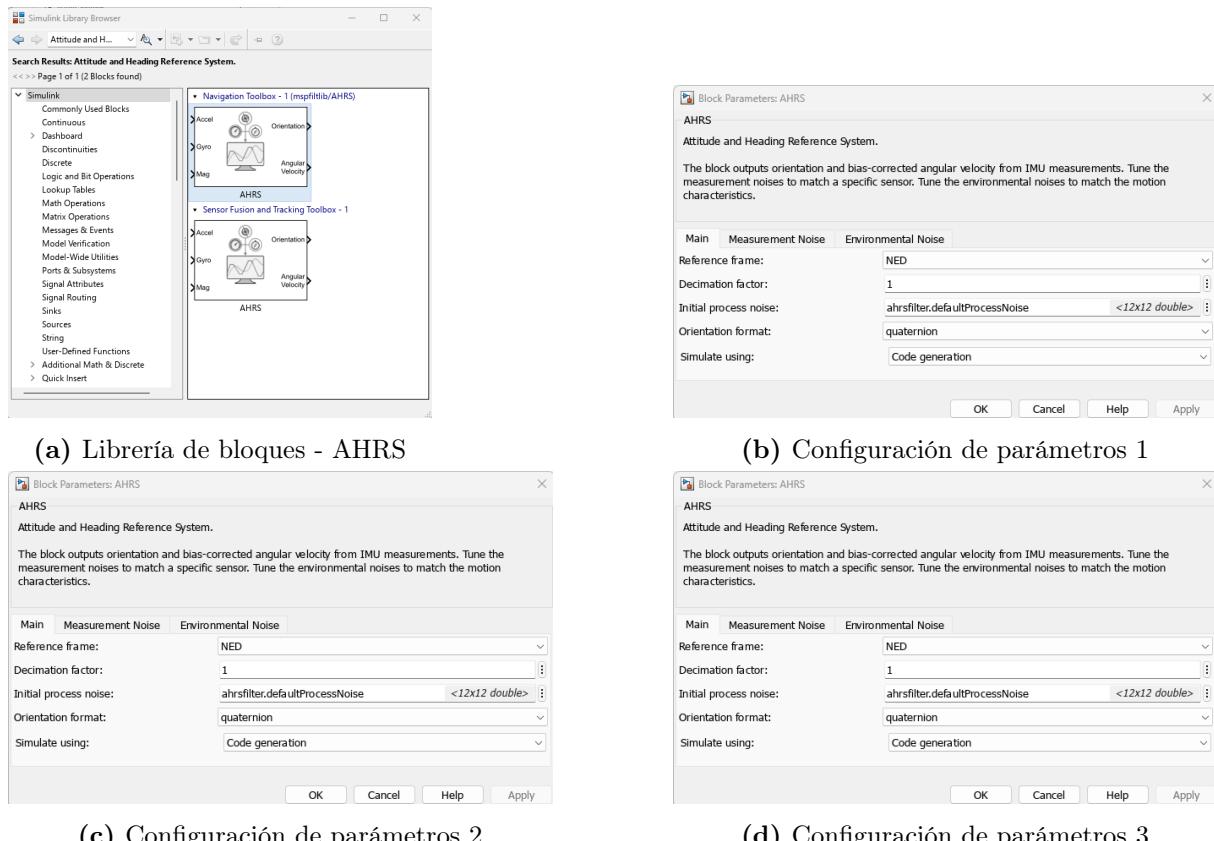
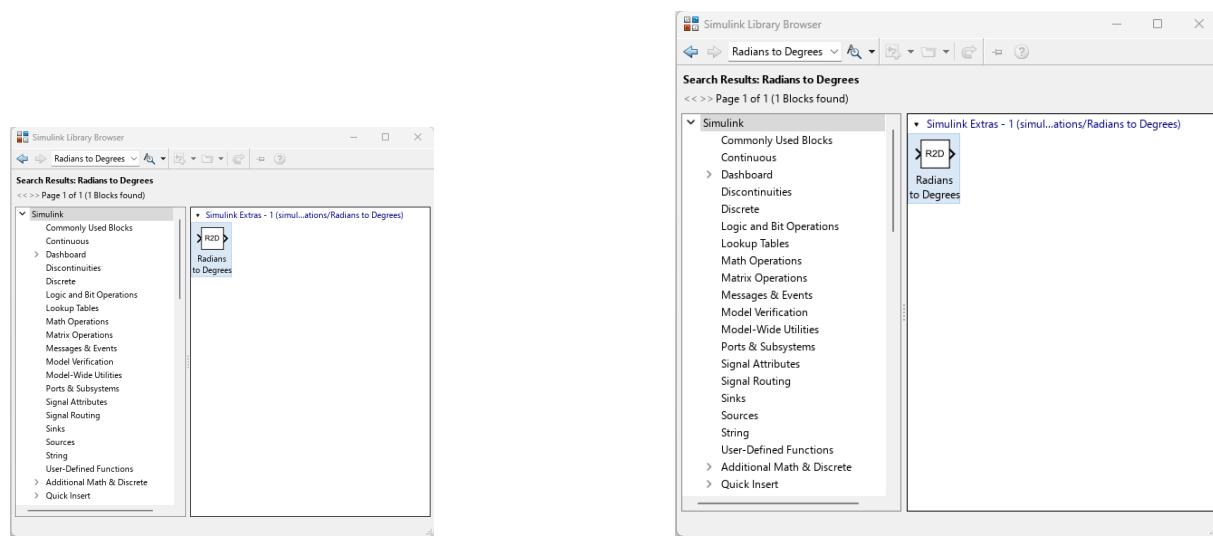
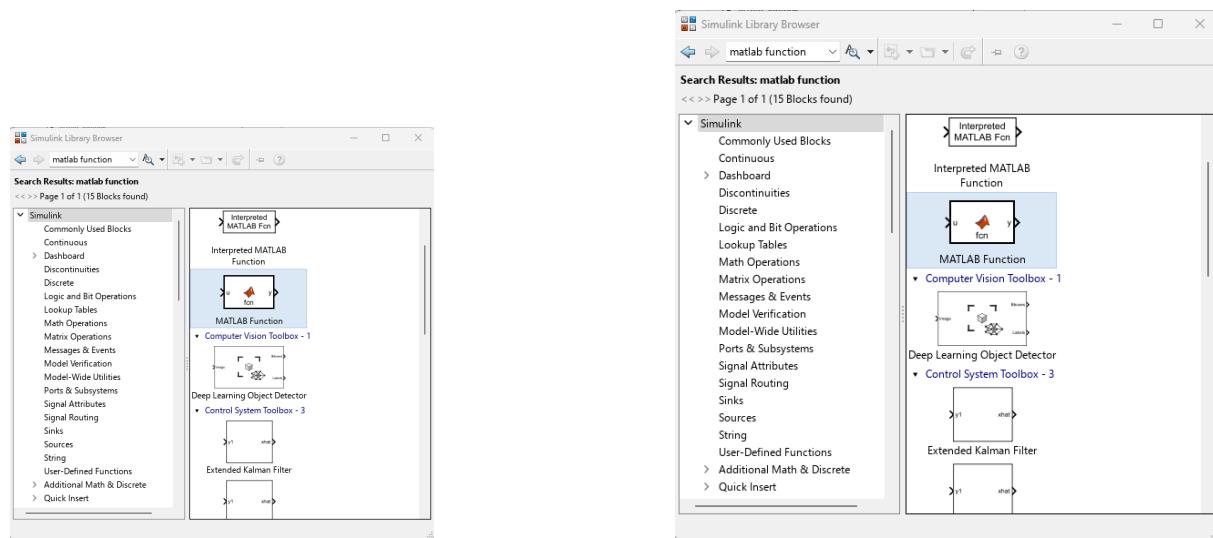


Figura 5.8: Bloque para la simulación del comportamiento de la IMU



(a) Librería de bloques - Conversor de radianes a grados

(b) Configuración del bloque conversor de radianes a grados

Figura 5.9: Bloque para convertir de Radianes a grados

(a) Librería de bloques - Función

(b)

Figura 5.10: Bloque para aplicar una función implementada mediante código

5.1.5. Resultados de la implementación

5.2. Caso de estudio 3 - PID

Finalmente como ultimo caso de estudio se desarrolla un controlador PID (Proporcional-Integral-Derivativo) es una herramienta clave en los sistemas de control automático, diseñada para minimizar el error entre una señal de referencia y la señal de salida real. Su importancia radica en su capacidad para ajustar la respuesta del sistema, logrando un equilibrio entre rapidez y estabilidad. Esto permite que el controlador maneje eficazmente perturbaciones y cambios en el entorno, siendo aplicable a una variedad de sistemas, como motores eléctricos, sistemas de calefacción y procesos industriales complejos.

En el contexto de MATLAB y Simulink, estas herramientas ofrecen una plataforma visual que facilita la implementación y ajuste de controladores PID. A través de bloques específicos en Simulink, los ingenieros pueden modificar en tiempo real los parámetros proporcional, integral y derivativo, observando directamente cómo estos ajustes afectan la salida del sistema. Esta capacidad de simulación y diseño iterativo no solo optimiza el rendimiento del sistema controlado, sino que también proporciona un entorno propicio para la experimentación y el aprendizaje práctico en el campo del control automático.

5.2.1. Implementación en MATLAB Simulink

Como se puede observar en la Figura 5.11, este es el caso de estudio que se propone en [microcontrollerslab_pid_controller_design], a este caso de estudio se le deben de realizar unas modificaciones de acuerdo al funcionamiento deseado que se tiene para este caso de estudio, siempre generando datos en el ámbito de simulación en MATLAB para luego contrastar los mismos con los datos obtenidos en la ejecución del modelo en la tarjeta de desarrollo seleccionada.

5.2.2. Bloques utilizados para la implementación

Los bloques utilizados se obtienen en la librería de bloques de MATLAB Simulink. A continuación se muestran los bloques requeridos, así como la configuración de los mismos para la correcta operación del modelo. La implementación del sistema se divide en dos partes, el primer parte se encarga de generar los archivos necesarios para la operación del sistema mientras que la segunda parte del sistema se encarga de leer los archivos con los datos y generar los dos archivos de salida del programa.

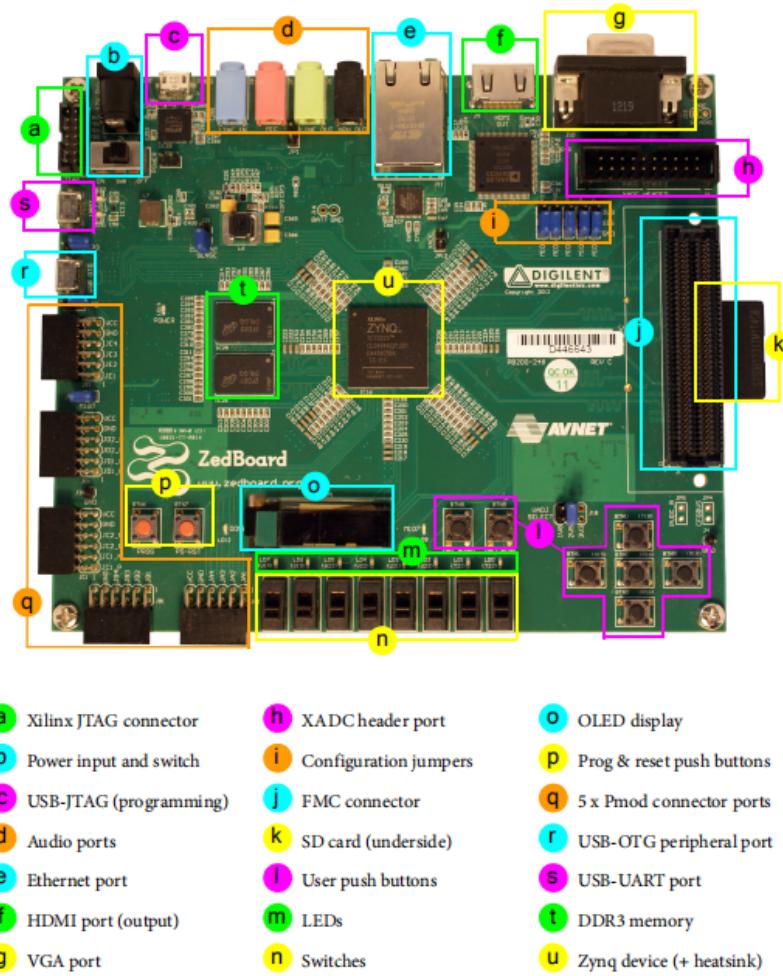


Figure 6.1: ZedBoard layout and interfaces (front)

Figura 5.11: Diagrama completo del caso de estudio 3 - PID

Sistema para la generación de archivos

Este sistema es el encargado de generar los archivos de entrada, estos mismos contienen los datos de tiempo y valores para la correcta implementación del sistema

5.2.3. Implementación en la Tarjeta de desarrollo mediante EmbeddedSynthGNC

Capítulo 6

Conclusiones

Las conclusiones no son un resumen de lo realizado sino a lo que ha llevado el desarrollo de la tesis, no perdiendo de vista los objetivos planteados desde el principio y los resultados obtenidos. En otras palabras, qué se concluye o a qué se ha llegado después de realizado la tesis de maestría. Un error común es “concluir” aspectos que no se desarrollaron en la tesis, como observaciones o afirmaciones derivadas de la teoría directamente. Esto último debe evitarse.

Es fundamental en este capítulo hacer énfasis y puntualizar los aportes específicos del trabajo.

Es usual concluir con lo que queda por hacer, o sugerencias para mejorar los resultados.

Bibliografía

- [1] MATLAB. «MATLAB y Simulink para sistemas espaciales,» visitado 16 de jul. de 2024. dirección: <https://la.mathworks.com/solutions/aerospace-defense/space-systems.html>.
- [2] L. Hewing et al., «Enhancing the Guidance, Navigation and Control of Autonomous Parafoils using Machine Learning Methods,» en *12th International Conference on Guidance, Navigation & Control Systems*, 2023, págs. 1-15.
- [3] «Introducción a la Ingeniería de Sistemas Espaciales,» Agencia Espacial Mexicana, visitado 16 de jul. de 2024. dirección: https://www.educacionespacial.aem.gob.mx/images/convocatorias/images/mem17/04Introducion_ala_ISE.pdf.
- [4] Á. R. Castaño, «Estimación de posición y control de vehículos autónomos a elevada velocidad.,» 2014. dirección: <https://api.semanticscholar.org/CorpusID:170879515>.
- [5] R. D. Culp y G. Bickley, «Guidance and control, 1993; Annual Rocky Mountain Guidance and Control Conference, 16th, Keystone, CO, Feb. 6-10, 1993,» 1993. dirección: <https://api.semanticscholar.org/CorpusID:108791083>.
- [6] A. Falcoz et al., «Guidance, Navigation & Control on-board architecture for Mars Sample Return Rendezvous & Capture,» *Papers of ESA GNC-ICATT 2023*, 2023. dirección: <https://api.semanticscholar.org/CorpusID:267285374>.
- [7] O. Salvador y D. Angolini, *Embedded Linux Development with Yocto Project*. Packt Publishing Ltd, 2014.
- [8] A. Vaduva, *Learning Embedded Linux Using the Yocto Project*. Packt Publishing Ltd, 2015.
- [9] F. Mihalič, M. Trunčič y A. Hren, «Hardware-in-the-loop simulations: A historical overview of engineering challenges,» *Electronics*, vol. 11, n.º 15, pág. 2462, 2022.
- [10] J. Montoya et al., «Advanced laboratory testing methods using real-time simulation and hardware-in-the-loop techniques: A survey of smart grid international research facility network activities,» *Energies*, vol. 13, n.º 12, pág. 3267, 2020.
- [11] J. S. G. Merchán, M. F. Rodríguez, G. J. C. Méndez y M. F. H. Morales, «Evaluación de modelos aproximados para el diseño de control automático en sistemas de riego a canal abierto,» 2019. dirección: <https://api.semanticscholar.org/CorpusID:230388188>.

- [12] F. Mesa, R. Ospina-Ospina y G. Correa-Vélez, «Estimación de variables de estado (LA y LC) en sistemas de control,» *Revista UIS Ingenierías*, 2020. dirección: <https://api.semanticscholar.org/CorpusID:228853996>.
- [13] F. Schwiegelshohn y M. Hübner, «Design of an attention detection system on the Zynq-7000 SoC,» *2014 International Conference on ReConfigurable Computing and FPGAs (ReConFig14)*, págs. 1-6, 2014. dirección: <https://api.semanticscholar.org/CorpusID:18920120>.
- [14] P. G. de Aledo Marugán, «Simulación y verificación de propiedades no-funcionales para sistemas embebidos,» 2017. dirección: <https://api.semanticscholar.org/CorpusID:67290306>.
- [15] J. M. Herrera-López, Á. Galán-Cuenca, I. García-Morales, M. Rollón, I. Rivas-Blanco y V. F. Muñoz, «Entorno de trabajo cíber-físico para cirugía laparoscópica,» *Revista Iberoamericana de Automática e Informática industrial*, 2023. dirección: <https://api.semanticscholar.org/CorpusID:265202759>.
- [16] A. Leppakoski, E. Salminen y T. D. Hämäläinen, «Framework for industrial embedded system product development and management,» *2013 International Symposium on System on Chip (SoC)*, págs. 1-6, 2013. dirección: <https://api.semanticscholar.org/CorpusID:21473510>.
- [17] C. A. Peñaloza-Luna, G. E. Gallego-Rodríguez, J. J. Ramírez-Mateus, K. C. Puerto-López y K. Y. Sánchez-Mojica, «Simulación de un convertidor DC/DC cuadrático elevador con control modo corriente mediante Matlab/Simulink,» *Mundo FESC*, 2022. dirección: <https://api.semanticscholar.org/CorpusID:270944525>.
- [18] J. E. O. García, E. M. de Lourdes Loaiza Massuh y R. A. C. Aguilar, «Matlab como una herramienta informática útil a la contabilidad,» *RECIMUNDO*, 2022. dirección: <https://api.semanticscholar.org/CorpusID:250030773>.
- [19] D. G. Sáenz, J. R. Azagra, S. N. Canal y M. G. Martínez, «Laboratorio para el desarrollo de sistemas de navegación y control para UAVS multirrotor,» *Actas de las XXXIX Jornadas de Automática, Badajoz, 5-7 de Septiembre de 2018*, 2020. dirección: <https://api.semanticscholar.org/CorpusID:198488769>.
- [20] M. A. Chávez-Gudiño, A. Concha-Sánchez, F. M. Maciel-Barboza, S. K. Gadi, S. Thenozhi y R. O. J. Betancourt, «Desarrollo y control de un helicóptero de laboratorio de 2 GDL y de bajo costo,» *Revista Iberoamericana de Automática e Informática industrial*, 2023. dirección: <https://api.semanticscholar.org/CorpusID:259644850>.
- [21] J. E. Z. Daza, D. González-Montoya, E. E. H. Bravo, C. A. Ramos-Paja y D. A. Aponte-Roa, «Plataforma de prototipos de control rápido para sistemas fotovoltaicos basados en Arduino y Simulink,» *Revista EIA*, 2021. dirección: <https://api.semanticscholar.org/CorpusID:236424349>.

- [22] D. O. C. Sarmiento, «Simulation of Isolated Photovoltaic System in Matlab / Simulink,» *Mundo FESC*, 2019. dirección: <https://api.semanticscholar.org/CorpusID:270530814>.
- [23] J. E. Eterovic, M. Cipriano y S. Nicolet, «Análisis de protocolos de comunicaciones para internet de las cosas,» 2018. dirección: <https://api.semanticscholar.org/CorpusID:149955331>.
- [24] C. Barrera-Ramírez, Ó. González-Miranda y J. M. Ibarra-Zannatha, «Sistema de planeación y control de navegación para un vehículo autónomo en un entorno urbano,» *Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI*, 2022. dirección: <https://api.semanticscholar.org/CorpusID:260620410>.
- [25] S. R. Sánchez, «Programación de laboratorios de biología portátiles abiertos basados en Arduino con el lenguaje de programación visual XOD,» 2020. dirección: <https://api.semanticscholar.org/CorpusID:215856445>.
- [26] C. Sacta y K. David, «Desarrollo de un lenguaje de programación gráfico para micro-controladores,» 2011. dirección: <https://api.semanticscholar.org/CorpusID:170649818>.
- [27] D. Casu, V. Dubanchet, H. Renault, A. Comellini y P. Dandré, «EROSS+ Phase A/B1 Guidance, Navigation and Control design for In-Orbit Servicing,» *Papers of ESA GNC-ICATT 2023*, 2023. dirección: <https://api.semanticscholar.org/CorpusID:267272107>.
- [28] A. J. Bordano, G. G. Mcswain y S. T. Fernandes, «Autonomous Guidance, Navigation and Control,» 1991. dirección: <https://api.semanticscholar.org/CorpusID:108853424>.
- [29] P. Lourenço et al., «Verification & validation of optimisation-based control systems: methods and outcomes of VV4RTOS,» *Papers of ESA GNC-ICATT 2023*, 2023. dirección: <https://api.semanticscholar.org/CorpusID:267287945>.
- [30] MathWorks, *IMU Sensor Fusion with Simulink*, Disponible en: <https://la.mathworks.com/help/fusion/ug/imu-sensor-fusion-with-simulink.html> [Accedido: 27-oct-2024], MathWorks, 2024.

Apéndice A

Métricas de comparación de señales

```
1 import h5py
2 import scipy.io
3 import matplotlib.pyplot as plt
4
5 #Cargar los archivos .mat
6 #En la linea 7 se debe de colocar la direccion del archivo de
7 #simulacion generado por MATLAB
8 with h5py.File(r"archivo_simulado.mat", 'r') as archivo:
9     print("Claves del archivo:", list(archivo.keys()))
10    senal_simulada = archivo[list(archivo.keys())[0]][:]
11
12 #En la linea 12 se debe de colocar la direccion del archivo generado
13 #al ejecutar el programa en el ZedBoard
14 senal_experimental = scipy.io.loadmat('archivo_experimental.mat')
15
16 vector1 = senal_simulada
17 vector2 = senal_experimental
18
19 #Calculo del Error promedio absoluto
20 mae = np.mean(np.abs(vector1 - vector2))
21 print(f"Error Promedio Absoluto (MAE): {mae}")
22
23 #Calculo de la raiz del error cuadratico medio
24 rmse = np.sqrt(np.mean((vector1 - vector2) ** 2))
25 print(f"Raíz del Error Cuadrático Medio (RMSE): {rmse}")
26
27 #Calculo del error cuadratico medio
28 mse = np.mean((vector1 - vector2) ** 2)
29 print("Error Cuadrático Medio (MSE):", mse)
```

Listing A.1: Métricas de comparación de señales