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 3 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.

Asegúrse 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/`.

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, 3 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, 3 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 Godínez
Profesor Asesor

Cartago, 3 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 caracter personal, en donde se expresa la gratitud por la oportunidad, el apoyo brindado, la inspiración ofrecida, el acompañamiento moral, etc.

David Duarte Sánchez

Cartago, 3 de octubre de 2024

Índice general

Índice de figuras

Índice de tablas

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 [**MathWorks2024_ss**].

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 [**hewing2023enhancing**].

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 [**AEM2017**].

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. [**Castao2014EstimacinDP**]

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. [**Culp1993GuidanceAC**]

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. [**Falcoz2023GuidanceN**]

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. [**salvador2014embedded**]

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. [**vaduva2015learning**]

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 [**mihalivc2022hardware**] [**montoya2020advanced**].

1.5. Objetivos y estructura del documento

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

El segundo se encarga de Establecer flujos de trabajo para el prototipado de algoritmos de control de orientacion y navegacion para aplicaciones espaciales con hardware en el loop. Y por ultimo el tecero consiste en Evaluar los casos de uso de una computadora de navegacion y control espacial mediante la implementacion de una aplicacion de referencia demostrativa (caso de la IMU)

Este documento incluye lo siguiente: en el capitulo 2 se presenta el marco teorico, donde se esbozan los fundamentos de la propuesta realizada. En el capitulo 3 se detalla la solucion propuesta y el modelo implementado para la solucion del problema. En el capitulo 4 se

presentan los resultados obtenidos. Por ultimo, el capitulo 5 se presenta las conclusiones de la investigacion y trabajo realizado, asi 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 optimiza 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 [1]. 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 diversa condiciones operativas [2].

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 [3]. 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 (FPGA). Esta plataforma es ideal para prototipar aplicaciones en el ámbito de sistemas embebidos. La tabla 2.1 resume las especificaciones que posee la tarjeta de desarrollo ZedBoard.

Tabla 2.1: 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	Conectores Pmod y FMC para módulos adicionales

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 [4]. 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 [5].

2.4.1. YOCTO

Yocto es un marco de trabajo (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 [6].

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 [6].

2.5. 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.

2.5.1. 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. 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.

2.7. 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.

Algunos comandos básicos para el manejo de contenedores son:

Listing 2.1: Lista todos los contenedores en un sistema

```
sudo docker ps -a
```

Listing 2.2: Inicializa el contenedor mediante el ID

```
sudo docker start <nombre-o-id-contenedor>
```

Listing 2.3: Ejecuta el contenedor en segundo plano

```
docker run -d <nombre-o-id-contenedor>
```

2.8. Protocolos de Comunicación

2.8.1. UART

El Universal Asynchronous Receiver-Transmitter (UART) es un protocolo de comunicación serie 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.

2.8.2. 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.

2.9. Revisión literaria

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

2.9.1. 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 [7].

2.9.2. 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 [8].

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 [9].

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.

2.10. 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 [10].

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 [11].

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 [12].

2.10.1. 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 [12].

Capítulo 3

Tarjeta 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 ingles (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, mas precisamente la capacidad de calculo, ya que estos sistemas reuqieren de un procesamiento intensivo para los calculos de trayectoria , estimacion de estado y control. Seguido de esto se debe de considerar que sea un sistema de baja latencia, ademas que el mismo tenga soporte para sensores y actuadores para poder mediir y controlar el sistema, ademas de esto los puertos de entrada y salida deben ofrecer la presicion necesaria ara 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 desiciones criticas en el momento requerido. ademas de tener la capacidad de ejecutar un sistema opereativo de tiempo real (RTOS) o bien Linux en tiempo real. Tambien un aspecto importante a contener por la tarjeta de desarrollo es el consumo de energia, esto sin dejar de lado las capacidades de simulacion y pruebas, ya que, en la interfaz de simulacion la tarjeta se debe de poder conectar a un entorno de pruebas de hardware-in-the-loop por sus siglas en ingles (HIL), ademas de las capacidades de depuracion y monitoreo.

Finalmente se deben de tomar en cuenta aspectos como lo son el Tamanno, peso y forma de la trjeta buscando que las mismas contengan un tamanno 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 anteriomente se eligieron las siguientes tarjetas de desarrollo, las mismas se presentaran con sus caracteristicas.

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 algorimos personalizados.

Por otro aldo esta ofrece una amplia gama de interfaces de comunicacion como: PCIe, Ethernet, I2C, SPI, UART, GPIO. En cuanto a la eficiencia energetica esta opcion contiene mecanismos para el control de energia, ademas de ser compatible con entornos de de simulacion y tiene interfaces JTAG oara una buena depuracion. finalmente es una tarjeta ampliamente utilizada en la industria en Sistemasd de prototipos avanzados y despliegue de HIL.

En sintesis esta opcion ofrece un procesamiento potente y versatil ademas 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 especualmente utilizado para aplicaciones de vision por computador o inteligencia artificial para sistemas GNC.

Las interfases de comunicacion presentes en esta tarjeta son puertos I2C, SPIm UART y

GPIO ademas de contener adicionalmente soporte nativo para camaras y sensores de alta gama. Ademas presenta capacidades en tiempo real ya que se puede implementar con el NVIDIA Jetpack SDK.

En cuanto a la eficiencia energetica, la misma posee un diseno optimizado para bajo consumo. ademas de ser compatible con interfases de pruebas y simulacion 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 vision por computador o bien inteligencia artificial, por otro lado posee un desarrollo potente para el procesamiento de tares 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 logico comparado con las FPGA para aplicaciones en tiempo real extremo, y esta tarjeta de desarrollo se encuentra mas enfocada en el la implementacion de soluciones que requieran inteligencia artificial.

TMS320C6678 Development Kit

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

Sobre la eficiencia energetica, esta tarjeta de desarrollo ofrece herramientas especificas para la optimizacion del consumo de energia, hacinedola adecuada para entornos de consumo energetico restringido, finalmente es compatible con Code Composer Studio, el cual es un entorno de desarrollo integrado que facilita las labores de integracion, simulacion y depuracion. Por tanto podemos decir que es una tarjeta de desarrollo muy adecuada para los sistemas de procesamiento de senañes y control, tiene una gran capacidad para soportar aplicaciones industriarles y aeroespaciales. Por otro lado, podemos ver que es un plataforma menos flexible que una FPGA.

ZedBoard de Avnet

Para la tarjeta ZedBoard en cuanto a procesamiento 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 froma tomando el procesador ARM el cual es ideal para ejecutar algoritmos de control y logica de navegacion en un entrno de RTOS o bien Linux, por otro lado la FPGA Zynq-7000 permite la ejecucion de tareas de procesamiento paralelo en hardware como el filtrado de sennales o algoritmos de estimacion de estad, 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. Ademas de esto contiene puertos Ethernet, micro usb y HDMI los cuales resultan utiles para la comunicacion externa y visualizacion de los sistemas de desarrollo.

La combinacion de un procesador ARM con una FPGA permite un equilibrio en el consumo de energia ya que la mayoria de tareas intensivas se pueden llevar a cabo en la FPGA y el SoC Zynq ofrece opciones de ahorro de energia lo cual siempre representa un beneficio para las aplicaciones embebidas. En cuanto a la 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 depuracion como lo es Jtag para el monitoreo en tiempo real de las aplicaciones ejecutandose en la FPGA y en el procesador ARM.

3.1.3. Criterios de comparación

Una vez presentadas las terjetas de desarrollo candidatas se procede con la definicion de los criterios de comparacion: 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 energia: 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. Caracteristicas Fisicas: 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.

3.2. Matriz de Pugh

Tabla 3.1: Matriz de Pugh para seleccionar la tarjeta de desarrollo que mejor de 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	15	15	15	15	15
Soporte de sistemas de tiempo real	20	20	20	15	20
Caracteristicas Fisicas	10	4	7	10	10
Costo de la tarjeta	15	7	10	10	15
Escalabilidad del sistema	10	10	6	6	10
Suma general		86	88	79	95
Posicion		3	2	4	1

Como se pudo observar en la Tabla 3.1, un claro ganador segun los requerimientos establecidos para este proyecto ha sido la tarjeta de desarrollo zedbaord ya que es la mejor opcion en cuanto a caracteristicas como lo son la capacidad de procesamiento y ...

3.3. Plataforma seleccionada

Como se pudo observar en la Tabla 3.1, la tarjeta de desarrollo seleccionada fue la Trajeta 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 meciono en el capitulo 2 en la Tabla 2.1 y anteriormente en este capitulo la tarjeta en cuestion 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 capitulo se analizaron los requerimientos de hardware que se deben de tomar en cuenta para el desaarrollo de este proyecto, seguido de esto se tomaron cuatros tarhjetas de desarrolo con el fin de elegir entre las que presentaran las prestaciones adecuadas para la tarea a realizar. Por un lado se tenian tarjetas muy potentes como xxxx y xxxx y por otro lado se tenian tarjetas de grandes dimenciones como la xxx y xxxx. Segun los parametros definidos se elige la tarjeta xxxx con numero de parte xxx para continuar con el desarrollo de este proyecto.

Capítulo 4

Flujo de trabajo para la implementación de software para GNC embebido

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

Es por esto que en este capítulo se pretenden establecer los flujos de trabajo para el prototipado de algoritmos de control de orientación y navegación para aplicaciones espaciales. Esto mediante el uso de MATLAB Simulink para tomar un caso de estudio como ejemplo.

Una vez seleccionado el caso de estudio se convierte el código por medio de la transformación de modelo de Simulink a un modelo de código C, esto con el objetivo de poder embeber el código C por medio del flujo de trabajo de Yocto Project y finalmente probar el mismo en la tarjeta de desarrollo seleccionada.

De esta forma se puede comparar los resultados obtenidos y el tiempo de ejecución que llevo la tarea en el computador y en el sistema embebido.

En la Figura 4.1, se muestra un diagrama del flujo de trabajo general. En este capítulo se trabajará en la sección remarcada en rojo la cual engloba la generación del modelo utilizado como caso de estudio, la validación del mismo en MATLAB Simulink, la generación de un código en lenguaje C y la incorporación del mismo en el flujo de trabajo de Yocto Project.

4.1. Selección del caso de estudio

Como caso de estudio se seleccionó una aplicación la cual permitiera una comparación de resultados antes del procesado y después del mismo, es por esto que se decidió implementar un filtro de tipo paso bajo haciendo uso de los siguientes bloques de MATLAB Simulink.

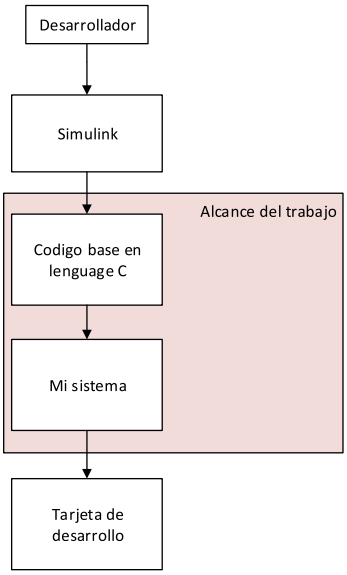


Figura 4.1: Diagrama general del flujo de trabajo propuesto

- Onda seno
- Suma
- Función de transferencia
- Generador de archivo de salida

La configuración seleccionada para el primer generador de onda seno es:

- Amplitud = 1
- Bias = 0
- Frecuencia = 1 rad/s
- Fase = 0
- Tiempo de muestreo = 0

Por otro lado, para la segunda onda se tiene la configuración:

- Amplitud = 1
- Bias = 0
- Frecuencia = 12 rad/s
- Fase = 0

- Tiempo de muestreo = 0

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.

$$y(t) = \sin(t) + \sin(12t) \tag{4.1}$$

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

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

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.

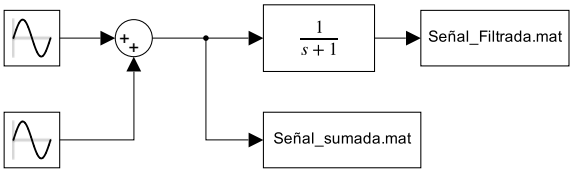


Figura 4.2: Diagrama MATLAB Simulink

Estos bloques mencionados anteriormente se colocan como se muestra en la Figura 4.2 de modo que se obtienen como salida del sistema dos archivos, uno llamado señal sumada el cual contiene los datos crudos de la suma de las dos señales y otro denominado señal filtrada el cual contiene los datos de la señal filtrada por la función de transferencia.

4.1.1. Simulación del caso de estudio en MATLAB Simulink

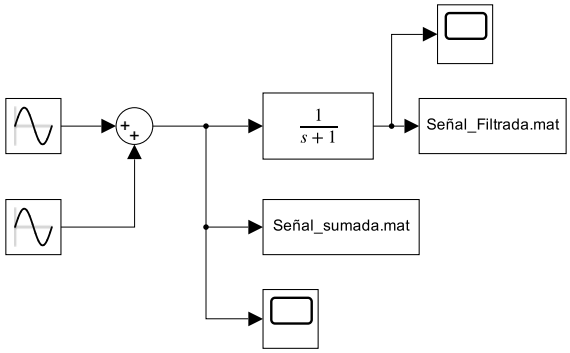
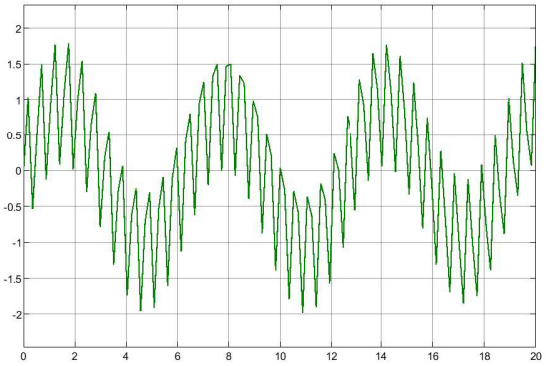


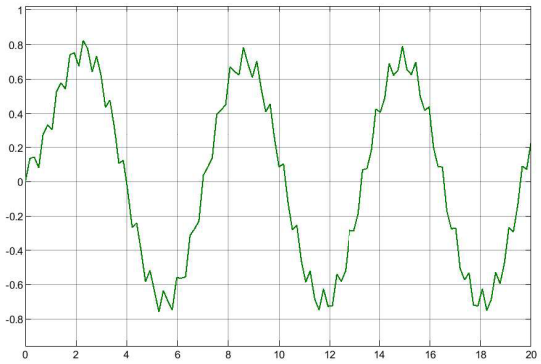
Figura 4.3: Diagrama MATLAB Simulink para poder observar las salidas

Utilizando el diagrama de la Figura 4.2, además de los parámetros configurados anteriormente se colocan dos bloques de gráfico en el diagrama como se muestra en la Figura 4.3,

esto con el objetivo de poder observar las señales de salida en cada uno de los puntos de interés.



(a) Ondas Moduladas



(b) Onda resultante luego de la función de transferencia

Figura 4.4: Salida resultante del diagrama mostrado en la Figura 4.3

Como se puede observar en la Figura 4.4a se puede observar la salida de la suma de las dos señales senoidales, por otro lado en la Figura 4.4b se puede observar la salida de la función de transferencia.

Resultados obtenidos con la ejecución de la simulación

Como se mencionó anteriormente los resultados obtenidos se pueden observar en la Figura 4.4, 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.

4.2. 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 del MATLAB Simulink Coder, el cual tiene la capacidad de convertir un sistema de control generado en MATLAB Simulink, en un código C. 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.

A lo largo de este capítulo se definirán los parámetros que se deben de utilizar y el funcionamiento de estos dentro de la generación del código C.

4.2.1. Simulink Coder

Una vez comprobado el comportamiento esperado por el caso de estudio se puede proceder con la ejecución del flujo de trabajo de MATLAB Simulink Coder, esto con el fin de transformar el modelo generado en 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 4.2, ya que, este solamente contiene como salida los archivos con los datos numéricos del sistema y no contiene las salidas gráficas agregadas en 4.1.1.

4.2.2. Definición de parámetros

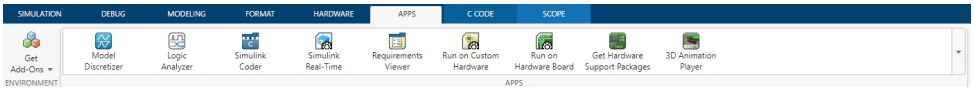


Figura 4.5: Pestaña Aplicaciones

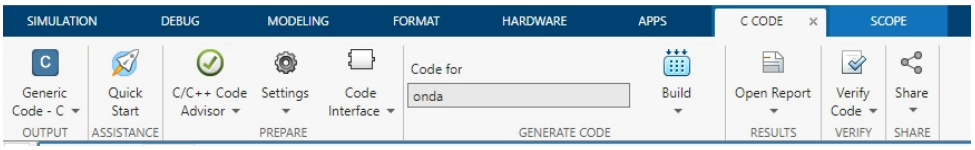


Figura 4.6: Pestaña código C

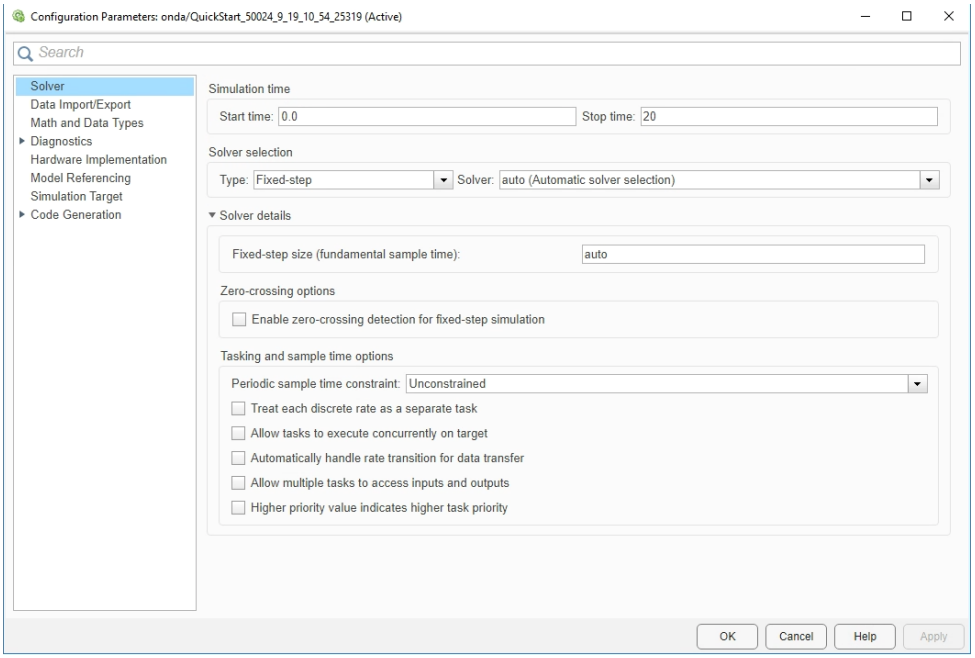


Figura 4.7: Configuración de parámetros

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 4.5, 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 4.6.

Una vez estemos en la pestaña de código C, debemos de ir a la opción de configuración de parámetros, en la Figura 4.6 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 4.7, en la pestaña denominada Solver se deberán de proporcionar los datos sobre el tiempo de ejecución de la prueba.

Selección del procesador objetivo

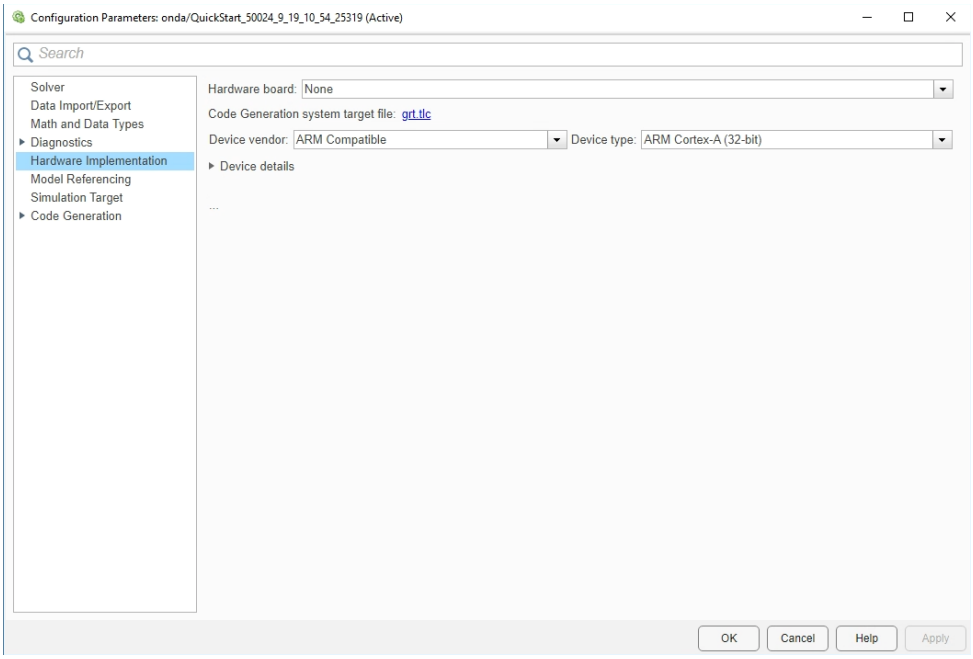


Figura 4.8: Selección del procesador y la familia del procesador

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 4.8.

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 utilizara 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 4.9, 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.

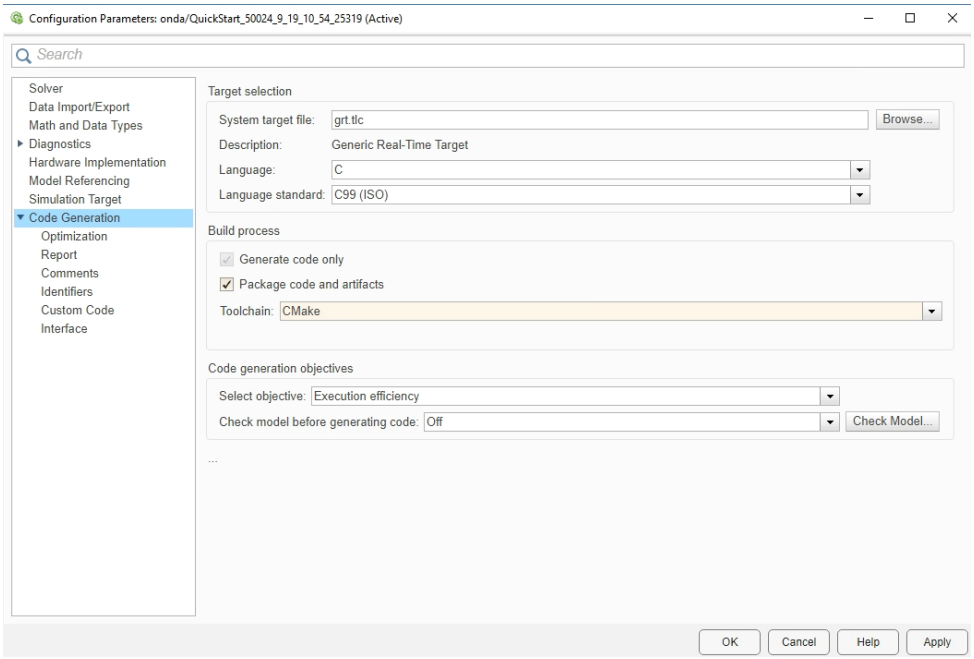


Figura 4.9: Selección del tipo de archivo de construcción

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 4.6 bajo el nombre de Build.

4.2.3. Contenedor para compilación de los binarios

Para poder generar la compilación cruzada es necesario crear un contenedor para poder utilizar Ubuntu 20.04, ya que esta versión es la que presenta mayor compatibilidad con las dependencias contenidas en el flujo de trabajo de Yocto Project que se utiliza.

Listing 4.1: Instalacion de docker, Linux

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

Listing 4.2: Instalacion de Ubuntu 20.04, Linux

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

Para poder generar este contenedor primeramente se debe de satisfacer la dependencia de tener instalado docker.io, esto se logra por medio del comando que se muestra en 4.1,seguido de esto, se puede hacer uso del comando 4.2 el cual se encarga de construir la máquina dentro del entorno 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 puedes ejecutar comandos como si fuera una máquina virtual con Ubuntu 20.04.

Instalación de programas en el contenedor

Listing 4.3: Instalacion del compilador cruzado, Linux

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

Listing 4.4: Instalacion de CMake, Linux

```
sudo apt install cmake
```

Listing 4.5: Instalacion de build essential, Linux

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

Una vez generado el contenedor debemos de instalar en el mismo el compilador cruzado que para nuestros efectos será arm-linux-gnueabihf-gcc, esto lo podemos hacer mediante el comando que se muestra en 4.3, CMake el cual es una herramienta de construcción multiplataforma y de código abierto que se utiliza para gestionar la construcción de software utilizando un enfoque basado en proyectos y finalmente build-essential las cuales son herramientas que nos ayudaran a compilar el programa generado en 4.2.1 para la arquitectura del procesador.

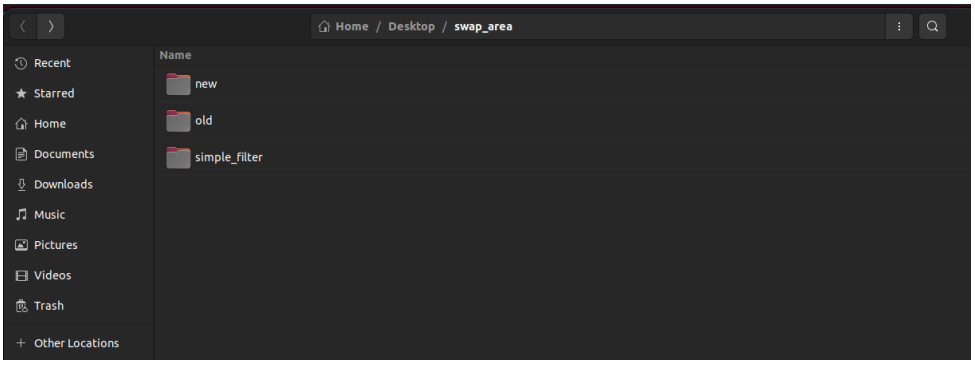


Figura 4.10: Archivo comprimido en el directorio swap_area

Listing 4.6: Copiar archivos al contenedor, Linux

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

Listing 4.7: Copiar archivos del contenedor, Linux

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

Seguido de esto se debe de copiar el archivo comprimido generado en 4.2.1, al contenedor con Ubuntu. Primeramente colocaremos el archivo comprimido en un directorio llamado swap_area, tal y como se muestra en 4.10. 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 4.6.

4.2.4. Compilación de los binarios

Listing 4.8: Compilacion del programa, Linux

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

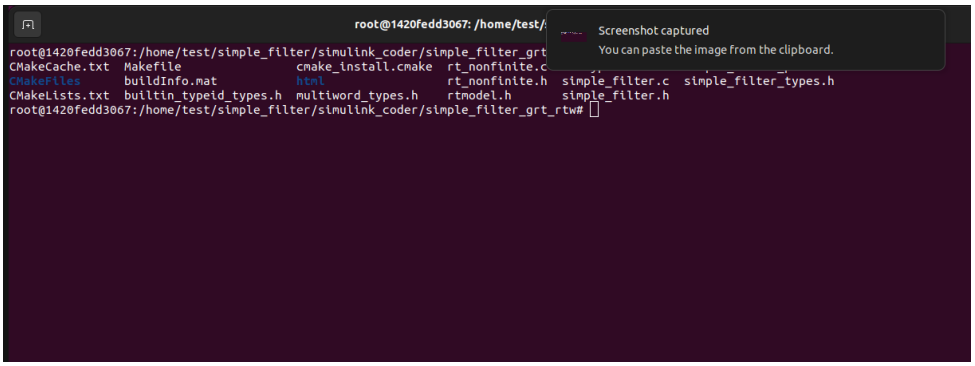


Figura 4.11: Make File

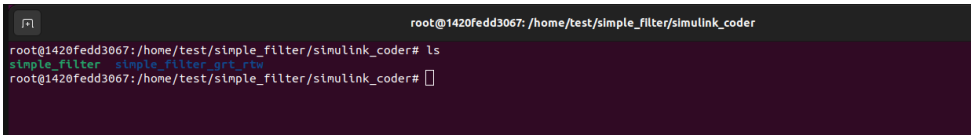


Figura 4.12: Binario llamado simple_filter

Para la compilación del archivo binario se deberá hace uso del comando 4.8 por el cual se construirá el Makefile, tal y como se muestra en la Figura 4.11, 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 4.12.

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.

4.3. Flujo de Trabajo Herramienta desarrollada por mi persona

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.

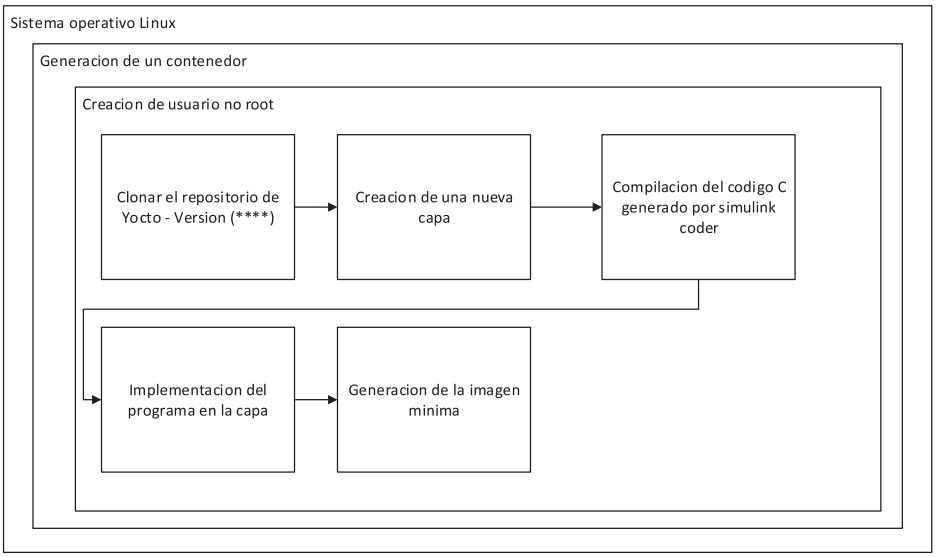


Figura 4.13: Flujo de trabajo Yocto

En el desarrollo de esta seccion se muestran los pasos que se siguieron para la generación de una imagen mínima, la integración de una capa personalizada con el binario generado en 4.2.4 y la implementación de la misma en la tarjeta de desarrollo seleccionada.

4.3.1. Sistema operativo para desarrollo

Como sistema operativo de desarrollo se utilizó Ubuntu 22.04 LTS, en una computadora con las siguientes características:

- Procesador - Intel Corei9-13980HX
- Almecenamiento - 500 GB
- Memoria RAM - 16 GB

4.3.2. Generación de un contenedor

Para el desarrollo del marco de trabajo de Yocto se decidió implementar un contenedor, esto debido a que la versión de Yocto para la cual se encontraba un paquete de soporte para la tarjeta de desarrollo es Yocto Zeus 3.0, este fue liberado en octubre del 2019, por tanto no era soportado por la versión de Linux del computador de desarrollo, es por esto que se tomó la decisión de utilizar un contenedor el cual ejecutara la versión de Ubuntu 16.04 LTS.

Listing 4.9: Instalacion de Ubuntu 16.04

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

Para la generación del contenedor se hace uso del comando que se muestra en 4.9, el cual se encarga de darnos un contenedor con Ubuntu 16.04 LTS.Ademas de esto debemos de instalar algunos requerimientos como lo son:

- tree : para poder visualizar árboles de dependencia
- vim : para poder editar archivos de texto
- etc ...

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:

Listing 4.10: Generacion de usuario no root, Linux

```
apt - get install -y sudo
useradd - ms/bin/bash myuser
echo "myuser:password" | chpasswd
usermod - aG sudo myuser
```

De forma que según el comando observado en 4.10, en la primera línea instalamos sudo, 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 çhpasswd", finalmente se agrega en el archivo usermod el nuevo usuario.

Listing 4.11: Iniciar usuario no root, Linux

```
su - myuser
```

Cada vez que iniciemos el contenedor siempre lo haremos como usuario root, para poder iniciar con el usuario no root llamado "myuser"se debe de hacer uso del comando que se muestra en 4.11.

4.3.3. 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.

Listing 4.12: Requerimientos Yocto Zeus, Linux

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

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.

Listing 4.13: Version de Yocto

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

Listing 4.14: BSP para Zedboard

```
git clone -b zeus https://github.com/Xilinx/meta-xilinx
git clone -b zeus https://github.com/openembedded/meta-openembedded.gi
```

La versión de yocto a utilizar se debe de clonar de 4.13, seguido de esto se debe de ir a la rama de la versión Yocto Zeus. Además de clonar este repositorio se debe de ingresar al directorio denominado poky y clonar dentro del repositorio 4.14 el cual contiene en su rama llamada Zeus la versión de paquete de soporte para la tarjeta requerida para generar una imagen para la tarjeta de desarrollo seleccionada.

Listing 4.15: Configuraciones adicionales, Yocto

```
source oe-init-build-env

echo "MACHINE ??=\"zedboard-zynq7\"" >> conf/local.conf
echo "IMAGE_FEATURES +=\"package-management\"" >> conf/local.conf
echo "DISTRO_HOSTNAME =\"zynq\"" >> conf/local.conf

bitbake-layers add-layer ../meta-xilinx/meta-xilinx-bsp/
bitbake-layers add-layer ../meta-openembedded/meta-oe/
```

Algunas configuraciones adicionales que se deben de realizar se muestran en 4.15.

4.3.4. Creación de una capa de yocto

Listing 4.16: "Print Working Directory",Linux

```
pwd
```

Listing 4.17: Inicializar ambiente, Yocto

```
source oe-init-build-env
```

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 4.16, seguido de esto se debe de inicializar el entorno de desarrollo esto mediante el comando que se muestra en 4.17, este se encarga de generar todos los archivos necesarios para poder hacer uso de las variables de entorno con las cuales opera el marco de trabajo de Yocto.

Listing 4.18: Generar nueva capa, Yocto

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

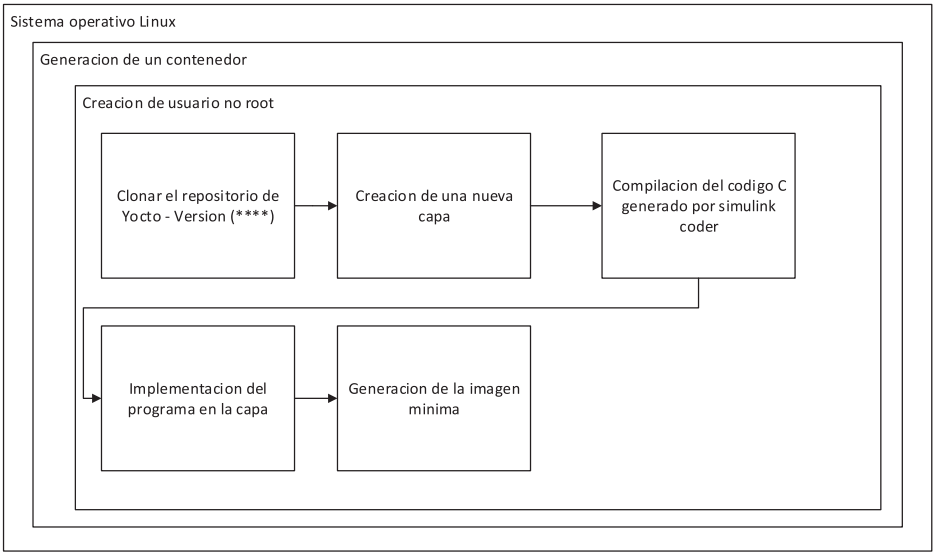


Figura 4.14: Árbol de directorios de la capa

Listing 4.19: Agregar nueva capa, Yocto

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

Seguido de esto se debe de utilizar el comando que se muestra en 4.18 el cual se encargara de generar el árbol de directorios que se puede observar en la Figura 4.14. Una vez implementado este comando se debe de hacer uso del comando que se muestra en 4.19 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.

4.3.5. Caso de estudio

En esta sección se integrará el caso de estudio generado en 4.2.4, a un sistema operativo a la medida por medio del marco de trabajo de Yocto Project, primeramente se generara una capa personalizada, seguido de esto se generara el archivo de instalación con el cual la capa personalizada pasara a ser parte del sistema de archivos del sistema embebido, también se generara la imagen mínima la cual consiste en un sistema de arranque y el sistema de archivos y finalmente se implementaran estos en la tarjeta de desarrollo seleccionada en 3.

4.3.6. Integración del programa generado a la capa de Yocto

Para la implementación del binario generado en 4.2.4, se deben de generar algunos directorios, esto con el objetivo de mantener un entorno 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 "sistema_control.[el] cual se encargara de contener el archivo de configuración de la capa llamado "sistema_control.bb"; este archivo se genera mediante el comando que se observa en 4.18, además de generar este archivo se debe de crear un directorio llamado "files"que será el encargado de contener el archivo binario compilado en 4.2.4.

sistema_control.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 encontraran los binarios en el sistema de archivos de la imagen del sistema embebido.

```
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 4.15: Estructura del archivo sistema_control.bb

La estructura que debe de contener ese archivo para instalar binarios en el sistema son las que se pueden observar en la Figura 4.15.

4.3.7. Generación de la imagen mínima

Listing 4.20: Generar archivos de desarrollador, Yocto

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

Antes de generar la imagen mínima se debe de tener en consideración ejecutar la línea de comando que se muestra en 4.20, 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 4.17, seguido de esto se deberá de ejecutar el comando de "bitbake core-image-minimal.[el] cual se encarga de comenzar a generar la imagen mínima.

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

Para la implementación de la imagen mínima desarrollada en 4.3.7 en la tarjeta de desarrollo, se deben de desarrollar los siguientes pasos en la máquina Host:

1. 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 4.7, esto con el objetivo de poder enviar los archivos a la tarjeta SD más adelante.

Listing 4.21: Copiar archivos root, Linux

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

Listing 4.22: Copiar sistema de archivos, Linux

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

Para el sistema Root se deberán de copiar los archivos mediante el comando que se muestra en 4.21, por otro lado en la partición denominada FileSystem se debe de copiar el archvivo çore-image-minimal-zedboard-zynq7.tar.gz"mediante el uso del comando que se muestra en 4.22.

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

Como protocolo de comunicación se establece de primera mano UART, el cual como se mencionó en 2.8 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 4.16, 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.

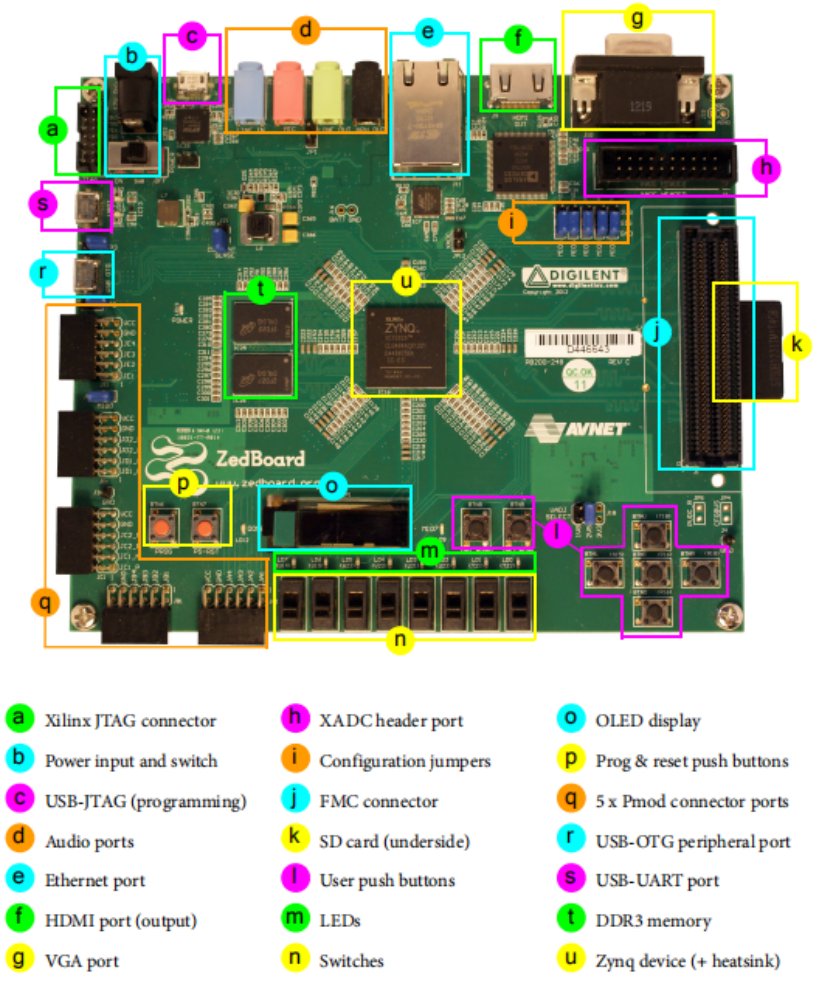


Figure 6.1: ZedBoard layout and interfaces (front)

Figura 4.16: Puertos tarjeta de desarrollo Zedboard

Una vez que se estableció la conexión por medio de SSH el cual como se menciona en 2.8, 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.

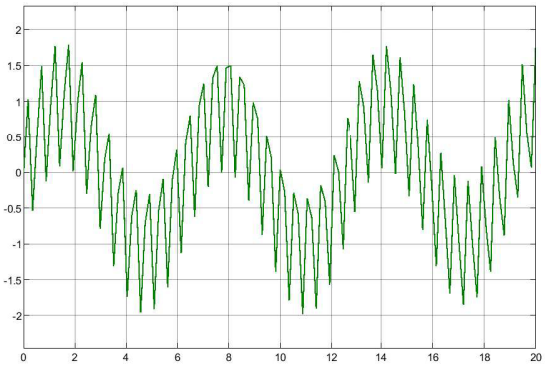
El diagrama de este protocolo de comunicación se puede observar en 4.16, el cual se logra mediante la conexión al puerto denominado en el diagrama como .

4.3.10. Ejecución del caso de estudio y resultados

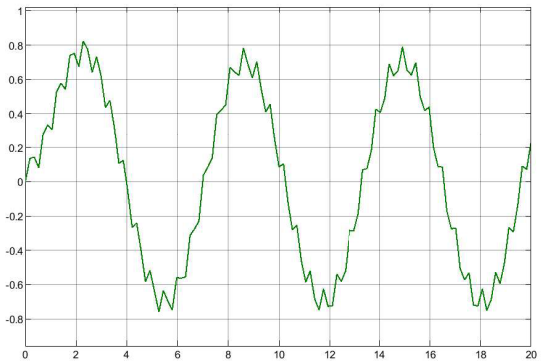
Una vez implementada la imagen de Yocto en la tarjeta de desarrollo se puede ejecutar el caso de estudio. Para esto será necesario dirigirnos al directorio en el cual se instaló el archivo binario, el mismo se encuentra en la ruta xxxxx. Una vez encontrado el archivo basta con ejecutarlo mediante el uso del nombre del mismo "simple_filter". Cuando el archivo se ejecuta genera dos archivos de salida llamados xxx y xxx, 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 4.23.

Listing 4.23: Copiar archivo por protocolo SSH, Linux

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



(a) Ondas Moduladas



(b) Onda resultante luego de la función de transferencia

Figura 4.17: Salida resultante del diagrama mostrado en la Figura 4.3

4.3.11. Comparación de resultados

Como se pudo observar, con la comparación realizada, obtenemos que los datos tiene una desviación de xxxxx además de xxxxxx, es por esto que podemos determinar que la implementación de sistemas de control es viable según lo demuestra el caso de estudio.

4.4. Reflexión final

Capítulo 5

Solución propuesta

En este capítulo se exponen los diseños experimentales realizados para comprobar el funcionamiento correcto del sistema. Por ejemplo, si se realiza algún sistema con reconocimiento de patrones, usualmente esta sección involucra las llamadas *matrices de confusión* donde se compactan las estadísticas de reconocimiento alcanzadas. En circuitos de hardware, experimentos para determinar variaciones contra ruido, etc. También pueden ilustrarse algunos resultados concretos como ejemplo del funcionamiento de los algoritmos. Puede mostrar por medio de experimentos ventajas, desventajas, desempeño de su algoritmo, o comparaciones con otros algoritmos.

Recuerde que debe minimizar los "saltos" que el lector deba hacer en su documento. Por tanto, usualmente el análisis se coloca junto a tablasy figuras presentadas, y debe tener un orden de tal modo que se observe cómo los objetivos específicos y el objetivo general del proyecto de tesis se han cumplido.

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] 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.

[2] 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.

[3] 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.

[4] P. G. de Aledo Marugán, «Simulación y verificación de propiedades no-funcionales para sistemas embebidos,» 2017. dirección: https://api.semanticscholar.org/CorpusID:67290306.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] C. Sacta y K. David, «Desarrollo de un lenguaje de programación gráfico para microcontroladores,» 2011. dirección: https://api.semanticscholar.org/CorpusID:170649818.

[10] 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.

[11] A. J. Bordano, G. G. Mcswain y S. T. Fernandes, «Autonomous Guidance, Navigation and Control,» 1991. dirección: https://api.semanticscholar.org/CorpusID:108853424.

[12] 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.

Apéndice A

Demostración del teorema de Nyquist

El título anterior es solo un ejemplo ilustrativo. Éste teorema no ameritaría un apéndice pues es parte normal del currículum de Electrónica, pero apéndices usualmente involucran aspectos de esta índole, que se salen de la línea de la tesis, pero que es conveniente incluir por completitud.

Los anexos contienen toda información adicional que se considere pertinente agregar, como manuales de usuario, demostraciones matemáticas que se salen de la línea principal de la tesis, pero que pueden considerarse parte de los resultados del trabajo.

Índice alfabético