UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MASTER
ELECTRÓNICA INDUSTRIAL

Detección y seguimiento ópticos de objetivos para apuntamiento de RIS

Autor
Ángel Manuel Sánchez Zarco

Directores
Antonio García Ríos
Luis Parrilla Roure

Escuela Internacional de Posgrado

—

Granada, julio de 2025

Detección y seguimiento ópticos de objetivos para apuntamiento de RIS

Ángel Manuel Sánchez Zarco

Palabras clave: RIS, 6G, electrónica, inteligencia artificial.

Resumen

Las superficies inteligentes reconfigurables (RIS) se perfilan como una tecnología clave para la transformación de los sistemas de radio, tanto en aplicaciones de comunicaciones como de radar, frente a los desafíos que plantea la evolución hacia la 6G. Para alcanzar todo su potencial, estos dispositivos deben incorporar una electrónica de control eficiente y capacidades inteligentes que les permitan reconfigurarse de manera autónoma y adaptativa según el entorno.

Con este objetivo, en este trabajo fin de máster se diseñará la electrónica de control en VHDL, lo que permitirá la reconfiguración completa del dispositivo y la comunicación entre los elementos de la RIS y el módulo encargado de gestionar el sistema RIS a configurar. Además, se implementará un sistema de seguimiento visual mediante cámara, capaz de detectar y seguir objetivos para ajustar dinámicamente la dirección del haz reflejado por la RIS. Este sistema, desplegado en distintas plataformas, se encargará de la captura y procesamiento de imágenes para estimar la posición del usuario y orientar la RIS de forma dinámica, facilitando así una transmisión eficiente.

Optical Detection and Tracking of Targets for RIS Beamforming

Ángel Manuel Sánchez Zarco

Keywords: RIS, 6G, electronics, artificial intelligence

Abstract

Reconfigurable Intelligent Surfaces (RIS) are emerging as a key technology for transforming radio systems, both in communication and radar applications, in response to the challenges posed by the evolution toward 6G. To fully realize their potential, these devices must incorporate efficient control electronics and intelligent capabilities that enable autonomous and adaptive reconfiguration according to the environment.

With this objective, this master's thesis will focus on designing the control electronics using VHDL, which will allow for complete device reconfiguration and communication between the RIS elements and the module responsible for managing the panel to be configured. Additionally, a visual tracking system using a camera will be implemented, capable of detecting and following targets to dynamically adjust the direction of the beam reflected by the RIS. This system, deployed on different platforms, will handle image capture and processing to estimate the user's position and dynamically orient the RIS, thus enabling efficient transmission.

D. **Antonio García Ríos**, Catedrático Universitario del Área de Tecnología Electrónica del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada.

D. **Luis Parrilla Roure**, Profesor titular universitario del Área de Electrónica del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Detección y seguimiento ópticos de objetivos para apuntamiento de RIS*, ha sido realizado bajo su supervisión por **Ángel Manuel Sánchez Zarco**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 11 de julio de 2025.

Los directores:

Antonio García Ríos **Luis Parrilla Roure**

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a mis tutores, Antonio García Ríos y Luis Parrilla Roure, así como a mi mentor, Rubén Padial Allué, quienes me han acompañado a lo largo del desarrollo de este trabajo. Su paciencia, compromiso y orientación en momentos clave han sido invaluables e indispensables. Y por quienes, a nivel personal y profesional, profeso admiración y respeto a partes iguales.

También me gustaría agradecer al grupo SWAT-UGR, y en particular a Juan Francisco Valenzuela Valdés y Marcos Baena Molina, por su colaboración en el despliegue técnico y material, una parte esencial para la realización de este Trabajo Fin de Máster.

Por último, me gustaría agradecer a mi familia por apoyar mi pasión en este campo y por brindarme su aliento incondicional a lo largo de todo este proceso. Su confianza y comprensión han sido un pilar fundamental para alcanzar esta meta.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción, Objetivos y Motivación

En esta primera sección se plantea el propósito general del trabajo fin de máster, así como los objetivos específicos que orientan su desarrollo. Además, se expone la motivación principal que impulsó la elección del tema.

1.1. Introducción

A lo largo de los últimos años, el tráfico de datos móviles ha experimentado un crecimiento exponencial, impulsado principalmente por el aumento de usuarios de tecnologías móviles. El tráfico de información total pasó de apenas 470,1 megabytes por línea en 2014 a más de 7.000 megabytes en 2021 (Figura 1.1). Este incremento no solo refleja un aumento en la cantidad de datos transmitidos, sino también una mayor demanda de conectividad por parte de los usuarios, evidenciada por el fuerte crecimiento del tráfico mensual por línea, que se multiplicó por más de 15 veces en el mismo periodo [1].

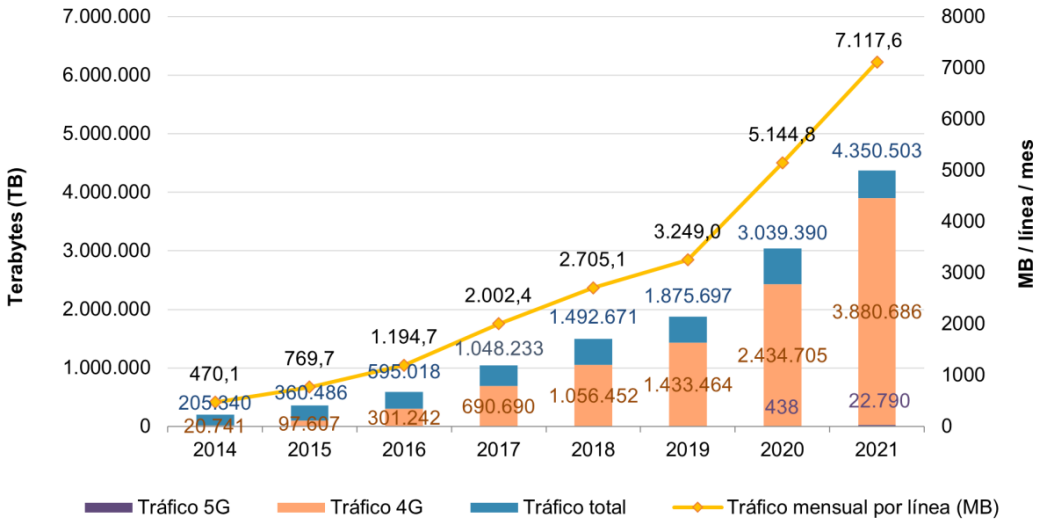

Figura 1.1: Tráfico total y mensual por línea (TB y MB/línea/mes). CNMC[1]

En este escenario, las demandas sobre las infraestructuras de telecomunicaciones son cada vez más exigentes, obligando a repensar el diseño de los sistemas inalámbricos desde una perspectiva más flexible, eficiente y adaptable. Tecnologías emergentes como Massive MIMO, beamforming adaptativo y acceso al espectro dinámico han permitido avances significativos, pero también imponen mayores niveles de complejidad en el hardware, el consumo de energía y la gestión del canal.

Por ello, las superficies inteligentes reconfigurables (RIS) se perfilan como una tecnología clave para la transformación de los sistemas de radio, tanto en aplicaciones de comunicaciones como de radar, frente a los desafíos que plantea la evolución hacia la 6G. A diferencia de tecnologías pasivas o estáticas como los *reflactarray*, la tecnología RIS puede adaptarse a condiciones variables del entorno y colaborar activamente en la optimización de los enlaces de comunicación, mejorando la cobertura, reduciendo la interferencia y aumentando la eficiencia espectral.

La implementación efectiva de estas superficies requiere una electrónica de control distribuida capaz de actuar en tiempo real sobre múltiples elementos de la superficie, en función de parámetros como la localización del usuario, las condiciones del canal o las necesidades del sistema. Además, la integración con sensores y sistemas de percepción, como cámaras o redes neuronales, abre la puerta al desarrollo de RIS inteligentes, capaces de operar de forma autónoma en entornos dinámicos y complejos.

Este avance tecnológico no solo representa una mejora incremental, sino un cambio de paradigma en el diseño de las redes inalámbricas del futuro. La posibilidad de transformar el entorno electromagnético en una entidad programable convierte a las RIS en una piedra angular para el despliegue de la próxima generación de redes móviles, donde la adaptabilidad y la inteligencia del sistema serán tan importantes como su capacidad de transmisión.

1.2. Motivación

El crecimiento exponencial del tráfico de datos móviles y las exigencias que plantea la evolución hacia redes 6G evidencian la necesidad de nuevas soluciones tecnológicas que permitan mejorar la eficiencia espectral, la cobertura y la adaptabilidad de los sistemas de comunicación inalámbricos. En este escenario, las Superficies Inteligentes Reconfigurables (RIS) emergen como una alternativa prometedora para transformar la forma en que se gestiona la propagación de las señales, al convertir el entorno en un canal de comunicación programable.

Sin embargo, para que esta tecnología alcance un nivel de madurez adecuado para su integración en entornos reales, es fundamental desarrollar sistemas capaces de operar en tiempo real. La posibilidad de integrar sensores, algoritmos de percepción y electrónica de control en un único sistema plantea un reto multidisciplinario que involucra conocimientos de comunicaciones, hardware embebido y visión por computador.

Este proyecto surge, por tanto, de la motivación por contribuir al avance de la teconología RIS desde un enfoque práctico e integrador, mediante el diseño y desarrollo de una plataforma que no solo permita la reconfiguración dinámica de estas superficies, sino que también incorpore capacidades inteligentes de seguimiento y adaptación al entorno.

1.3. Objetivos

A partir de los retos identificados y la motivación descrita anteriormente, se han definido los siguientes objetivos que guiarán el desarrollo de este trabajo fin de máster.

OE1 **Diseñar un sistema** para la reconfiguración de un dispositivo RIS, considerando los requisitos de robustez, latencia y control en tiempo real.

OE2 **Diseñar e implementar hardware** que gestione la comunicación con el dispositivo RIS y permita configurar sus estados operativos de forma eficiente y confiable.

OE3 **Integrar un sistema de tracking basado en cámaras** capaz de detectar y rastrear en tiempo

real la posición de objetivos, para alimentar al sistema de reconfiguración RIS con información contextual que permita una adaptación dinámica y dirigida del haz.

OE4 **Validar el sistema** mediante pruebas funcionales y de rendimiento, evaluando tiempos de respuesta, precisión en la configuración y estabilidad durante operaciones en tiempo real.

OE5 **Documentar el desarrollo** para asegurar la reproducibilidad y facilitar futuras mejoras o expansiones del sistema.

1.4. Planificación

En esta sección se refleja el conjunto de tareas realizadas para abordar el desarrollo del proyecto. A continuación, se detallan los recursos materiales y humanos involucrados, así como la planificación temporal que ha seguido la ejecución de cada fase del trabajo.

1.4.1. Tareas

Para la planificación del proyecto se ha estructurado en una serie de tareas que permiten abordar de forma ordenada y progresiva los distintos componentes del sistema propuesto. Cada tarea responde a un objetivo específico dentro del proceso de diseño, implementación e integración de una solución completa para la reconfiguración dinámica de superficies inteligentes reconfigurables.

TA1 **Diseño de un hardware de comunicación** encargado de recibir y transmitir información utilizando el protocolo de comunicación SPI. Este componente constituye la base del sistema de control del dispositivo RIS, garantizando la correcta interpretación y envío de datos binarios entre módulos..

TA2 **Diseño de un protocolo específico** de comunicación basado en SPI, para la reconfiguración de dispositivos RIS con resolución de 2 bits. Este protocolo establecerá una estructura de tramas eficiente que permita configurar el dispositivo de manera rápida y eficiente.

TA3 **Diseño hardware de una memoria volátil**, encargada de almacenar los valores equivalentes a cada estado del dispositivo RIS. Aumentando la capacidad adaptativa de la RIS.

TA4 **Simulación y evaluación** del hardware desarrollado, mediante técnicas de *test-bench* y el uso de analizadores lógicos.

TA5 **Implementación de un sistema de seguimiento activo en CPU**, que permita controlar dinámicamente la configuración del dispositivo RIS en función de la posición del objetivo.

TA6 **Implementación de un sistema de seguimiento activo en aceleradores hardware**, optimizado para ofrecer baja latencia en el control dinámico del dispositivo RIS.

TA7 **Integración completa** del sistema de reconfiguración RIS, combinando hardware, protocolo de comunicación y sistema de seguimiento.

TA8 **Evaluación** completa del sistema en un entorno de prueba controlado. Esta tarea permitirá medir el rendimiento, la robstez y la capacidad de adaptación de la solución propuesta en condiciones reales.

TA9 **Recopilación bibliográfica** que ilustre sobre las tecnologías a utilizar, su estado actual y las principales líneas de investigación relacionadas. Esta tarea permitirá fundamentar las decisiones de diseño y seleccionar las soluciones más adecuadas.

TA10 **Elaboración de la memoria** que documente el trabajo realizado, incluyendo los objetivos alcanzados, la metodología empleada, los resultados obtenidos y las aportaciones derivadas.

TA11 **Revisión y corrección** que oriente la mejora de la memoria final, mediante la evaluación del contenido, redacción, estructura y formato, con el fin de garantizar la claridad, coherencia y calidad del documento a entregar.

1.4.2. Temporización

Se presenta a continuación el cronograma que resume el tiempo dedicado a cada una de las tareas contempladas en el desarrollo del proyecto.

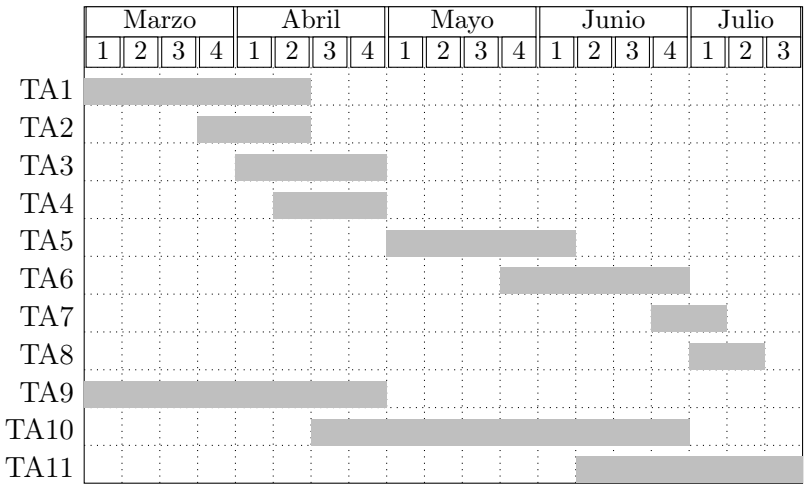

Figura 1.2: Temporización del proyecto

Dificultades en la temporización

TA1 Está tarea extendió su duración 2 semanas a causa de la complejidad del hardware desarrollado, encontrando problemas en sincronización de los componentes, síntesis del código en VHDL y depurado de *glitches*.

TA3 Está tarea necesitó de un día de test extra del tiempo planificado, para el depurado de *latches* y desarrollo de distintos *testbenches*.

TA4 Esta tarea se extendió un día más, porque hubo que realizar distintas verificaciones y depurado para sincronizar la memoria con el hardware desarrollado previamente.

TA5 La complejidad asociada y los pocos recursos disponibles para la integración de esta tecnología necesitó de una semana extra de investigación para completar la tarea.

1.4.3. Recursos humanos y técnicos

Para la correcta ejecución del proyecto, ha sido fundamental contar con un conjunto de recursos materiales y técnicos adecuados a las exigencias de cada tarea.

Recursos materiales

Se presentan los recursos materiales utilizados en la elaboración de este proyecto:

- 9x Cmod A7-35T: Breadboardable Artix-7 FPGA Module
- 225x Step Motor Driver TMC5130
- 1x Raspberry Pi 4
- 1x Arria 10 Development Kit
- 1x Kinect V1

Recursos humanos

Se presentan los recursos humanos empleados en la elaboración de este trabajo fin de máster (Tabla 1.1):

Tarea	Horas estimadas	Horas empleadas
Diseño de hardware SPI	60	80
Diseño de un protocolo	10	10
Diseño de hardware Memoria volátil	20	22
Simulación y evaluación del hardware	5	8
Implementación de un sistema de seguimiento activo en CPU	30	30
Implementación en aceleradores hardware	40	50
Integración	15	15
Evaluación	15	15
Recopilación bibliográfica	15	20
Elaboración de la memoria	30	40
Revisión y corrección	25	25
Total	265	315

Tabla 1.1: Recursos humanos necesarios

Coste asociado al proyecto

Se presenta el coste asociado a los recursos empleados en la elaboración de este trabajo fin de máster (Tabla 1.2):

Recurso	Precio
9x FPGA Cmod A7-35T	990€
225x Step Motor Driver TMC5130	4.740,75€
1x Raspberry Pi 4	68,40€
1x Arria 10 Development Kit	3.995€
1x Kinect V1	35€
Coste del ingeniero	40€/hora x 315 horas = 12.600€
Total	22.429,15€

Tabla 1.2: Coste del proyecto

Capítulo 2

Estado del arte

En este capítulo se presentan los fundamentos teóricos y los antecedentes académicos que respaldan el desarrollo de este proyecto. Se examinará en detalle el funcionamiento de Yolo (You Only Look Once), así como los principios teóricos y prácticos de una Superficie Inteligente Reconfigurable (RIS, por sus siglas en inglés). Finalmente, se analizará un estudio que integra las capacidades de las superficies reconfigurables con algoritmos de reconocimiento de imágenes como Yolo.

2.1. Reconfigurable Intelligent Surfaces (RIS)

Las superficies inteligentes reconfigurables (por sus siglas en inglés RIS) son antenas alimentadas capaces de adaptar las ondas electromagnéticas (EM) que inciden sobre ellas. Su construcción consiste en una superficie comúnmente plana dividida en múltiples elementos individuales que alteran su respuesta electromagnética. Estos elementos o celdas unidad pueden ser mecánicos o eléctricos (por ejemplo, guías de ondas, parches microstrip, dipolos, etc.), permitiendo ajustar dinámicamente el estado de cada celda según las condiciones del entorno o los requisitos del sistema.

Los dispositivos RIS se utilizan principalmente en sistemas de telecomunicación avanzados, como redes 5G y 6G, con el objetivo de mejorar la cobertura, aumentar la eficiencia espectral y reducir el consumo energético. Esto se consigue gracias a su capacidad de reflejar la señal incidente de forma inteligente, optimizando la energía y dirección del haz, incluso en escenarios donde las señales directas entre transmisor y receptor se ven bloqueadas por obstáculos (figura 2.1), a diferencia de las antenas radiantes que emiten la señal en todas direcciones o en haces fijos, sin capacidad de adaptación al entorno.

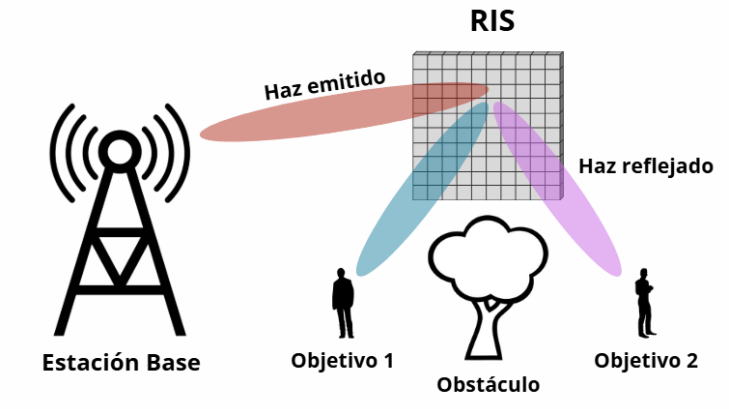

Figura 2.1: Imagen conceptual del uso de un dispositivo RIS

Este dispositivo es una evolución del concepto de *reflactarray* (RA) desarrollado por D.G. Berry en la década de 1960 [2]. En este artículo, el autor presenta una nueva clase de antenas que combina la simplicidad estructural de los reflectores con la versatilidad de las antenas de fase controlada. La superficie reflectante se compone de múltiples elementos pasivos cuya impedancia superficial está diseñada para controlar la fase del frente de onda reflejado. Esto permite construir distintos patrones de radiación sin necesidad de un sistema de alimentación complejo.

Para conseguir esto, Berry ofrece un prototipo compuesto por una matriz de guías de onda cortocircuitadas (Figura 2.2). Cada guía de onda actúa como un elemento reflectante, y su profundidad (es decir, la distancia entre la apertura y el cortocircuito en cada guía) determina la fase con la que se refleja la onda incidente, consiguiendo ajustar la fase del campo reflejado.

En la actualidad, el prototipo de D.G. Berry ha quedado relegado a una prueba de concepto en el contexto académico; sus grandes dimensiones y el desarrollo/descubrimiento de nuevas tecnologías han dado paso a la utilización de nuevos elementos que, además de tener un menor tamaño, permiten manipular con gran flexibilidad las ondas electromagnéticas en los dominios temporal, frecuencial y espacial.

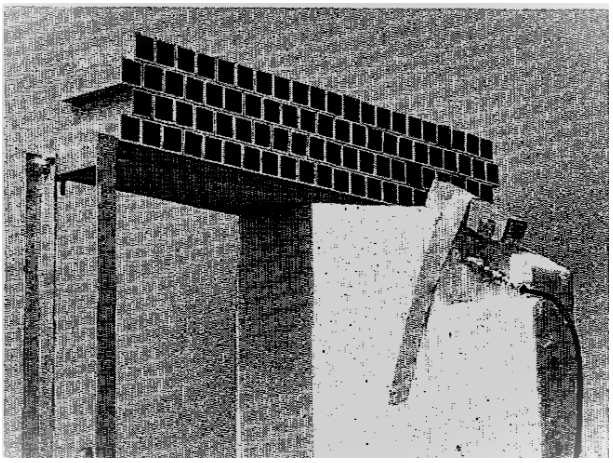

Figura 2.2: Propuesta de *Reflactarray* Berry década de 1960 [2]

2.1.1. Integración hardware para RIS

Para cumplir con su característica principal, las RIS (Reconfigurable Intelligent Surfaces), tal como indica su nombre, deben ser reconfigurables. Para lograrlo, es necesario que cuenten con una electrónica independiente del apartado de radiofrecuencia (RF), la cual permita modificar el estado de las celdas del dispositivo. La electrónica encargada de esta tarea está directamente influida por el diseño del elemento que constituye la celda del dispositivo. Además, en ciertas propuestas se han completado los demostradores con sistemas de identificación de objetos para aplicaciones de reconfiguración automática.

1-bit Mechanically Reconfigurable Metasurface [SWAT-UGR 2025]

Este primer prototipo fue diseñado en la Universidad de Granada y su publicación está datada del año 2025. El dispositivo consta de una matriz 10×10 cuya unidad elemental es un electroimán y los estados de la celda pueden ser únicamente '0' o '1' [3].

El dispositivo consta de las siguientes partes (ver Figura 2.3):

1. La primera sección del dispositivo está compuesta por una PCB con conectores macho de 2.54

mm dispuestos en una matriz de 10×10. Su función es configurar el panel, permitiendo seleccionar entre dos estados: cortocircuito, que activa el electro-imán y retrae el émbolo, y circuito abierto, que lo desactiva, permitiendo que un muelle interno expanda el émbolo.

2. La segunda etapa de la RIS incluye otra PCB que actúa como pasarela entre los electroimanes y la PCB de configuración. Para ello, incorpora una matriz 10 × 10 de conectores hembra JST de 2 mm, diseñada específicamente para facilitar la conexión de los electroimanes.

3. La etapa final del dispositivo consiste en una matriz 10×10 formada por electro-imanes, montados sobre una estructura impresa en 3D que los mantiene organizados y separados, evitando que se imanten entre sí.

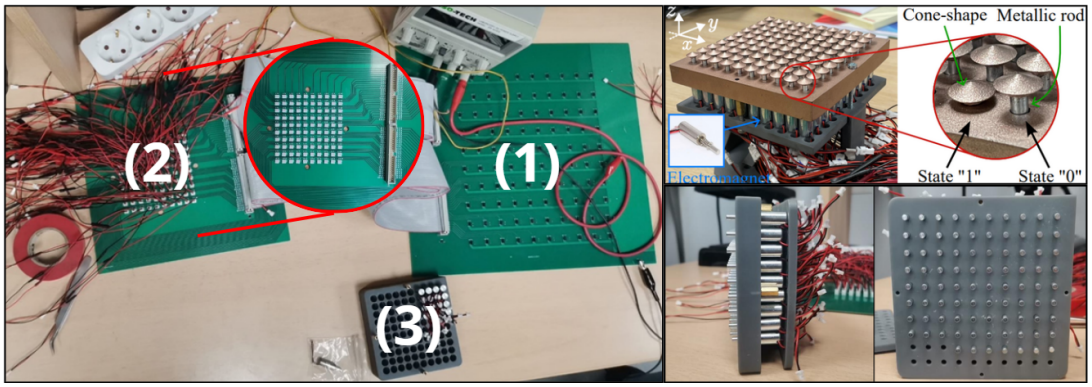

Figura 2.3: Prototipo RIS-3D SWAT-UGR 2025

Este prototipo carece de un hardware encargado de configurar el panel, ya que el operario debe configurar el panel de la RIS manualmente mediante *Jumpers* cortocircuitando la alimentación del electroimán para alterar su estado. Por lo tanto, el tiempo necesario para reconfigurar el panel depende directamente de cuántos elementos se modifiquen respecto a la configuración anterior y de la destreza del operario.

1-bit Diode Reconfigurable Intelligent Surface [Universidad de Xi'an 2022]

Este prototipo, desarrollado por la Universidad de Xi'an en 2022, se compone de celdas unidad diseñadas a partir de diodos PIN organizadas en una matriz de 20×20 (400 unidades en total) [4]. Al igual que el primer prototipo 2.1.1, consta de celdas de 1 bit (únicamente '0' o '1'), su frecuencia óptima es de 5.4 GHz y tiene un ancho de banda reducido por la respuesta no ideal de los diodos.

Este demostrador cuenta con un sistema de reconfiguración autónoma del panel para el control del estado de las celdas, implementado mediante dos FPGAs Intel Cyclone IV. La arquitectura está organizada jerárquicamente, con una FPGA maestra (FPGA M) y una FPGA esclava (FPGA S).

La FPGA Maestra recibe, a través de una interfaz de comunicación SPI, los datos correspondientes al panel que se desea reconfigurar. Una vez recibido el conjunto de datos, este se divide en dos bloques:

- El primero, compuesto por 200 elementos, es configurado directamente por la FPGA M una vez recibido el panel.
- El segundo bloque, también de 200 elementos, es enviado desde la FPGA M a la FPGA S para lo configure.

Se optó por este enfoque principalmente por la limitación de pines que poseen estas FPGAs, por otra parte dividir el panel permite una distribución eficiente para la configuración paralela de las celdas, optimizando el tiempo total de reconfiguración del sistema a 85 μs.

Por otra parte, para calcular el panel óptimo a configurar, el demostrador combina el uso de dispositivos de visión con modelos de detección de objetos. En este caso, se implementa la cámara estereoscópica ZED II y el modelo de detección YolovX desde una CPU. A partir de la imagen capturada por la cámara, el modelo de Yolo identificará el objeto objetivo, en este caso antenas receptoras, y usando visión estéreo calcula la profundidad del objetivo, pudiendo así estimar los ángulos de salida óptimos de haz reflejado en la RIS obteniendo latencias menores a 1 ms [4].

El prototipo de RIS asistido por visión desarrollado por la Universidad de Xi'an representa un avance significativo en el campo de las superficies inteligentes reconfigurables, siendo el primer demostrador que emplea información visual (mediante una cámara binocular) para determinar la posición del objetivo respecto a la RIS, configurando el panel refleje el haz de forma óptima.

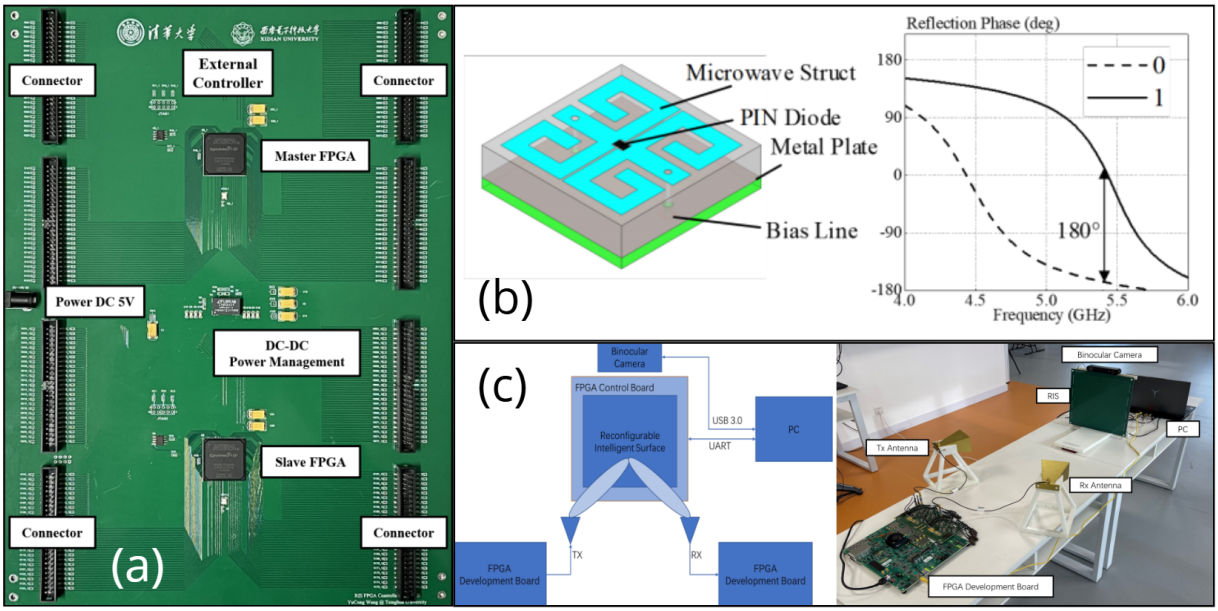

Figura 2.4: Prototipo RIS 1-bit Diode a) Placa de control RIS con dos FPGAs Intel Cyclone IV. b) Celda unidad del demostrador c) Demostrador con diagrama de funcionamiento

2.2. Detección de objetos

En la actualidad, el reconocimiento de objetos se ha convertido en una herramienta clave dentro del campo de la inteligencia artificial, especialmente en aplicaciones de visión por computador. Esta tecnología permite identificar, clasificar y localizar objetos dentro de imágenes o secuencias de video, aportando información valiosa sobre la escena en la que nos encontramos.

Estos modelos evolucionan de los modelos de detección de características implementados a finales de los años 90 y principios de los 2000:

- **SIFT (Scale-Invariant Feature Transform)** presentado por David Lowe en 1999 [5], es un algoritmo diseñado para detectar y describir puntos clave (keypoints) en una imagen de manera que sean invariantes a transformaciones como escala, rotación, y cambios de iluminación o perspectiva. El algoritmo SIFT comienza generando un espacio-escala, el cual se construye aplicando filtros de suavizado Gaussiano a la imagen original en distintos niveles de resolución mediante la aplicación de filtros de suavizado Gaussiano. Luego, se calcula la diferencia entre estas imágenes para identificar extremos locales (máximos y mínimos) que corresponden a los puntos clave (keypoints), que serán utilizados posteriormente para describir características distintivas de la imagen.
- **HOG (Histograms of Oriented Gradients)** propuesto por Dalal y Triggs en 2005 [6], es un descriptor de características que se enfocó principalmente en la detección de objetos, destacando en la detección de peatones. A diferencia de SIFT, que trabaja con puntos clave específicos, HOG analiza toda la imagen (o ventanas dentro de ella) dividiéndola en pequeñas regiones llamadas celdas. En cada celda se calcula un histograma de orientaciones de los gradientes de intensidad, lo que permite capturar la estructura de bordes del objeto.

En el año 2013 con el auge del aprendizaje profundo, OverFeat [7] fue uno de los primeros modelos en usar redes convolucionales tanto para clasificación como para localización. Este trabajo marcó una transición importante hacia los métodos basados en deep learning, y sentó las bases para la aparición de redes convolucionales complejas como Yolo y ResNet.

2.2.1. Deep Learning

Deep Learning es un tipo de *Machine Learning* que utiliza redes neuronales artificiales para resolver tareas complejas. Mientras que el *Machine Learning* tradicional se basa en una forma de estadística aplicada, que usa computadoras para estimar funciones complejas y encontrar patrones de datos para su posterior clasificación, el *Deep Learning* se destaca por poder aprender patrones y características directamente de los datos, sin que sea necesario definirlos manualmente.[8]

Esto representa una gran ventaja frente al enfoque tradicional del *Machine Learning*, especialmente en aplicaciones prácticas como el reconocimiento de imágenes. En los métodos clásicos, era necesario que el usuario definiera manualmente qué características debían extraerse de una imagen, por ejemplo, bordes o formas específicas, para que el algoritmo pudiera clasificarlas correctamente. Este proceso compromete al usuario a elegir características suficientemente representativas que permitan al modelo clasificar correctamente los datos. Si estas características no son las idóneas para el modelo, se limitará el rendimiento, provocando en muchos casos un proceso de prueba y error en la selección de estas características.

En cambio, con el uso de *Deep Learning*, los modelos son capaces de aprender automáticamente estas representaciones a partir de grandes cantidades de datos, extrayendo patrones cada vez más complejos a medida que se profundiza en las capas de la red neuronal. Esto permite una mayor precisión y

adaptabilidad a distintos tipos de datos de entrada sin perder precisión en la clasificación. Los modelos *Deep Learning* se clasifican en dos grandes grupos Modelos de aprendizaje supervisado y Modelos de aprendizaje no supervisado:

Modelos de aprendizaje supervisado:

Los algoritmos de aprendizaje supervisado son, en esencia, algoritmos que aprenden a asociar una entrada con una salida, dado un conjunto de datos de entrenamiento. En muchos casos, las salidas pueden ser difíciles de procesar automáticamente y deben ser proporcionadas por un supervisor humano, pero el término sigue siendo válido incluso cuando los objetivos del conjunto de entrenamiento se obtienen automáticamente.

Existen numerosos modelos de aprendizaje supervisado, estos se diferencian tanto por las entradas del modelo como por los algoritmos/principios que implementan:

- **Regresión Lineal** La regresión lineal es el método más extendido dentro del contexto de algoritmos de aprendizaje supervisado. Consiste en encontrar una relación lineal entre una variable dependiente continua y una o varias variables independientes. El modelo asume que el valor de salida Y puede expresarse como una combinación lineal de las entradas X, más un término de error.[8]

- **Regresión Logística** Este método se utiliza principalmente para problemas de clasificación binaria. Se basa en la función *sigmoide* para mapear una combinación lineal de las características de entrada a un valor entre 0 y 1, interpretado como una probabilidad.[9]

- **Árboles de Decisión** Los árboles de decisión segmentan el espacio de características en regiones jerárquicas utilizando clasificación binarias. Son interpretables y fáciles de visualizar, aunque propensos al sobreajuste.[10]

- **Redes Neuronales Artificiales** Este método se inspira en el cerebro humano, las redes neuronales consisten en capas de neuronas artificiales que transforman la entrada mediante funciones de activación no lineales. Son altamente expresivas y capaces de modelar relaciones complejas. Su rendimiento aumenta con grandes volúmenes de datos y forman la base del aprendizaje profundo actual, siendo utilizadas en tareas de visión por computador.[11]

2.2.2. Redes neuronales convolucionales

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son una clase especializada de redes neuronales artificiales diseñadas para el procesamiento de datos con una estructura matricial, como las imágenes [12].

Una CNN está compuesta por diferentes tipos de capas, entre las que destacan las capas convolucionales, capas de activación no lineal (como ReLU), capas de agrupamiento (pooling) y capas completamente conectadas. Las capas convolucionales aplican filtros (también conocidos como kernels) que se deslizan sobre la imagen de entrada para extraer características locales, como bordes, formas o texturas. A medida que se avanza en la red, estas características se combinan para formar representaciones cada vez más abstractas y complejas.

Este tipo de redes están muy extendidas en distintos campos de la ingeniería, algunas implementaciones en proyectos actuales son las siguientes:

CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning [13]

En este artículo publicado por investigadores de la Universidad de Stanford en el año 2017, tiene como objetivo principal desarrollar un sistema basado en redes neuronales convolucionales (CNN) capaz de detectar neumonía en radiografías de tórax con una precisión comparable a la de radiólogos expertos.

Los autores utilizaron una arquitectura de red neuronal convolucional basada en DenseNet-121, una variante optimizada de ResNet, entrenada con el conjunto de datos ChestX-ray14, que contiene más de 112,000 radiografías etiquetadas con 14 patologías torácicas diferentes, incluyendo neumonía. Durante el entrenamiento, el sistema aprendió a identificar patrones asociados a neumonía en las imágenes médicas.

Los resultados demostraron que CheXNet superó el rendimiento promedio de cuatro radiólogos en la tarea de detección de neumonía, alcanzando un área bajo la curva ROC (es una métrica que muestra la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos) de 0.80, en comparación con 0.76 obtenido por los especialistas humanos. El modelo logró una precisión del 90.1 % y mostró una mayor sensibilidad que los radiólogos en casos ambiguos. Con el fin de ser utilizado como una herramienta para los especialistas, los investigadores ajustaron la salida del modelo para que precisara las regiones de la imagen que más contribuyen al diagnóstico, facilitando así la interpretación de los resultados por parte de los médicos (Figura 2.5).

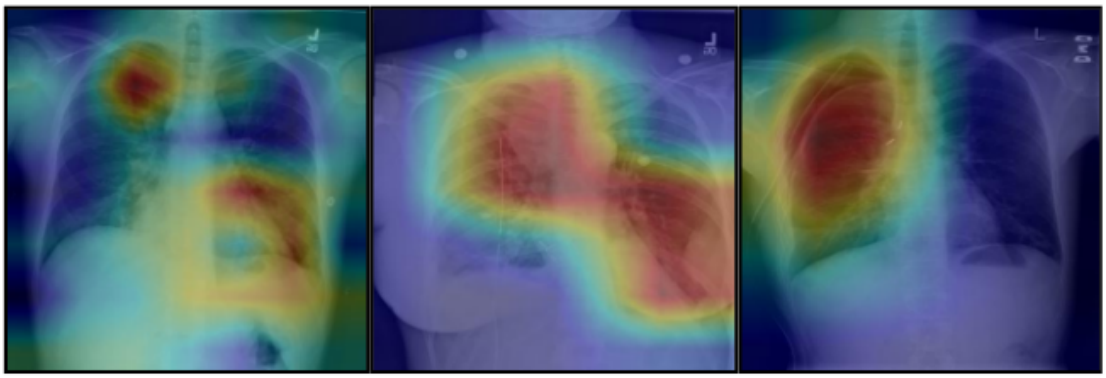

Figura 2.5: Detecciones realizadas con cheXnet a) Paciente con neumonía multifoca. b) Paciente con neoplasia pulmonar primaria c) Paciente con neumotórax derecho

FaceNet (Google, 2015) [14]

FaceNet es un sistema avanzado de reconocimiento facial desarrollado por Google en 2015, destaca por su arquitectura basada en una red neuronal convolucional (CNN) del tipo Inception, optimizada específicamente para el procesamiento de características faciales. El verdadero avance radica en su enfoque de aprendizaje métrico mediante la función de pérdida por tripletas (Triplet Loss). Cada iteración utiliza tres imágenes (rostro base de referencia, otro rostro de la misma persona y rostro de una persona diferente), forzando al modelo a minimizar la distancia entre las representaciones (embeddings) vectoriales del mismo rostro mientras maximiza la separación entre rostros diferentes (Figura 2.6).

Técnicamente, FaceNet superó los métodos tradicionales al eliminar la necesidad de pasos intermedios como alineamiento facial o extracción manual de características, logrando una precisión del 99.63 % con tiempos de inferencia de apenas 1.6ms por imagen.

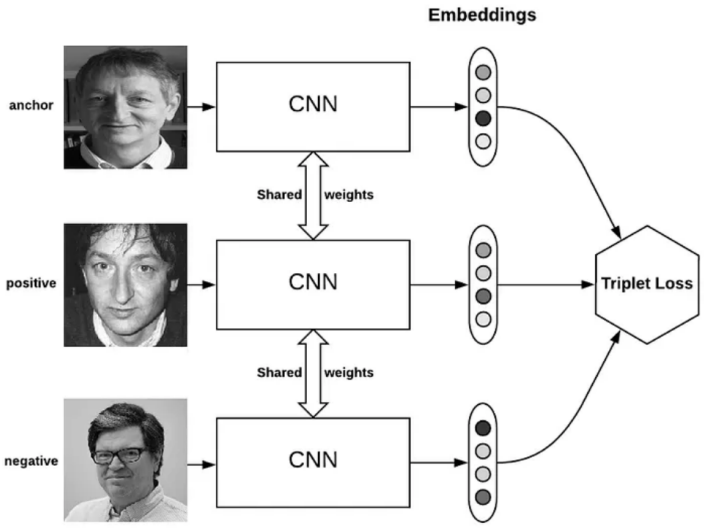

Figura 2.6: Técnica de *Triplet Loss* implementada en FaceNet

2.2.3. Yolo (You Only Look Once)

Yolo (You Only Look Once) es un algoritmo de detección de objetos en imágenes y video. A diferencia de otros métodos tradicionales que primero generan propuestas de regiones y luego las clasifican, Yolo trata la detección como un problema de regresión directa: toma una imagen completa y, en una sola pasada por la red neuronal, predice las ubicaciones de los objetos y sus respectivas clases.

A nivel conceptual, Yolo divide la imagen en una cuadrícula y, para cada celda, predice un número fijo de cajas delimitadoras (bounding boxes) junto con las probabilidades asociadas a cada clase a la que pueda corresponder. Esto le permite identificar múltiples objetos de diferentes tipos dentro de la misma imagen, lo que lo hace ideal para aplicaciones como vigilancia, seguimiento activo y robótica.

Actualmente existen numerosas versiones de este modelo, mejorando la precisión, velocidad y capacidad de detección en condiciones complejas. Cada versión implementa técnicas que, aunque similares, presentan diferencias que las distinguen significativamente. Por lo tanto, en sincronía con el proyecto, nos centraremos en Yolov3.

Este modelo se ha extendido a numerosas aplicaciones gracias a su equilibrio entre velocidad y precisión, así como a su capacidad de adaptación al ser reentrenado con distintos datasets, incluso cuando estos presentan características radicalmente distintas.

Pedestrian Counting Using Yolov3 [2021] [15]

Esta implementación de Yolo fue presentada *International Conference on Innovative Trends in Information Technology (ICITIIT)* en el año 2021, en ella se propone un sistema de conteo de peatones basado en el modelo Yolov3 para aplicaciones en seguridad urbana y gestión de multitudes.

El modelo fue entrenado con datasets públicos como COCO (Common Objects in Context) y CityPersons, utilizando técnicas de aumento de datos (ej. cambio de brillo y contraste) para mejorar la robustez en escenarios reales donde no se pueden controlar las condiciones de entorno.

Los resultados mostraron una precisión del 88.5 % en detección y un conteo eficiente incluso en multitudes de densidad media, con un rendimiento de 30 FPS en una GPU NVIDIA. El sistema incluyó un algoritmo de seguimiento basado en la asociación de detecciones entre fotogramas para evitar conteos redundantes. Las principales limitaciones fueron la disminución de precisión en aglomeraciones muy densas y la dependencia de la calidad del video de entrada.

Automatic detection of pulmonary nodules on CT images with Yolov3: Development and evaluation using simulated and patient data [2020] [16]

El artículo presenta un método automatizado para la detección de nódulos pulmonares en imágenes de tomografía computarizada (TC) utilizando la arquitectura de red neuronal Yolov3.

Para el entrenamiento se utilizó un *dataset* que combinaba tanto datos simulados como clínicos. Esto porque los datos simulados permitieron controlar variables como el tamaño, la forma y la ubicación de los nódulos, mientras que los datos de pacientes reales proporcionaron un escenario clínico más fiel a la realidad.

Este tipo de entrenamiento consiguió que el modelo tuviera una alta robustez frente a tamaños pequeños de nódulos, siendo capaz de detectar estas características a tamaños de 3–5 mm con una sensibilidad del >87 % (Figura 2.7).

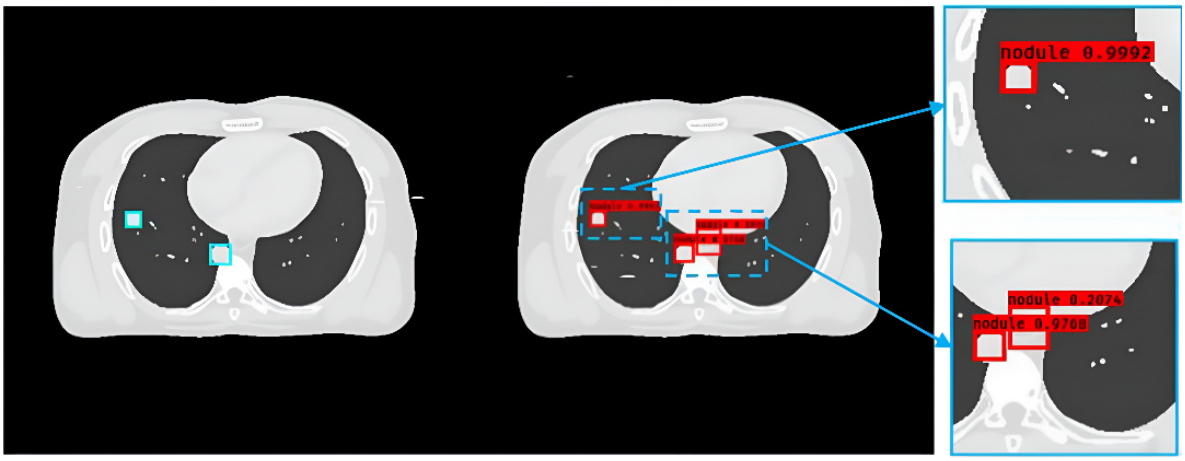

Figura 2.7: Predicción de nódulo realizada por el modelo de YoloV3 [16]

2.3. Aceleradores hardware para algoritmos de IA

En la actualidad, con el creciente auge de la integración de la Inteligencia Artificial (IA) en la automatización de procesos, también ha surgido la necesidad de desarrollar tecnología que pueda alojar estos modelos, reduciendo latencias para implementaciones *real-time*.

Una de las tecnologías que se barajan para estas tareas son los aceleradores hardware para IA. Estos aceleradores están diseñados específicamente para optimizar operaciones matemáticas intensivas, como la multiplicación de matrices y convoluciones, fundamentales en redes neuronales. Ofrecen mayor velocidad de procesamiento en comparación con las CPU, además, al ser una implementación directa en hardware, esta tecnología puede ser implementada en dispositivos FPGA lo que los hace ideales para sistemas compactos basados en SoC FPGA (System on Chip FPGA).

Una característica importante de estos aceleradores hardware es el formato para representar los datos del modelo, principalmente se usan dos formatos de representación, y su uso implica diferencias en el rendimiento, ancho de banda y precisión [17]:

- **Formato de 32 bits:** Su estructura consta de:
  - 1 bit para el signo (positivo/negativo).
  - 8 bits para el exponente (rango dinámico).
  - 23 bits para la mantisa (precisión fraccionaria).

  Tiene una alta precisión, ideal para entrenamiento de modelos complejos y un rango dinámico amplio para la representación de números muy grandes o muy pequeños. Aunque esto provoca un mayor consumo de memoria y ancho de banda, además de un coste mayor en operaciones hardware.

- **Formato de 16 bits** Su estructura consta de:
  - 1 bit para el signo (positivo/negativo).
  - 5 bits para el exponente (rango dinámico).
  - 10 bits para la mantisa (precisión fraccionaria).

  Este modelo es más conservador en cuanto a rendimiento, pero mejora la velocidad de operación y su eficiencia energética al necesitar menos hardware para su funcionamiento.

Los dos grandes proveedores para implementaciones de Aceleración Hardware en FPGA son Xilinx® con Vitis® e Altera® con Intel® FPGA AI Suite.

- **Intel® FPGA AI Suite.** es una plataforma completa para el desarrollo de aceleración hardware en FPGAs de la serie de Altera. Compatible con plataformas como Agilex 5/7, Arria 10, Cyclone 10, etc... Utiliza para la integración de modelos de IA, frameworks populares (TensorFlow, PyTorch, Caffe, MXNet, ONNX) mediante integración con el OpenVINO™ toolkit.

- **Vitis® AI** es la plataforma de aceleración de inteligencia artificial de AMD/Xilinx que permite desplegar modelos de aprendizaje profundo optimizados sobre FPGAs y ACAPs (Adaptive Compute Acceleration Platforms), como los Zynq UltraScale+ y los Versal AI Core. También es compatible con los frameworks más populares.

High-performance FPGA-based CNN accelerator with block-floating-point arithmetic [18]

El artículo presenta un acelerador basado en FPGA para redes neuronales convolucionales (CNN) que utiliza aritmética de punto flotante por bloques (BFP). El acelerador se implementa en una placa Xilinx VC709 empleando formatos de 16 bits para mapas de características y 8 bits para parámetros del modelo, reduciendo los requisitos de memoria y ancho de banda en un 50 % y 75 %, respectivamente, en comparación con el formato de 32 bits, solo perdiendo precisión con un valor inferior al 0.12 %.

Con estos cambios, el acelerador alcanza un rendimiento de 760.83 GOP/s (Gigaoperaciones por segundo) y una eficiencia energética de 82.88 GOP/s/W (Gigaoperaciones por Watio) a 200 MHz.

El artículo presenta una línea de investigación innovadora enfocada en optimizar aceleradores FPGA para redes neuronales convolucionales (CNN), abordando tanto el rendimiento computacional como la eficiencia energética. La clave de este aporte radica en el uso de aritmética de punto flotante por bloques (BFP), que reduce significativamente los requisitos de memoria y ancho de banda sin comprometer la precisión del modelo.

Briefly Analysis about CNN Accelerator based on FPGA [19]

El artículo analiza la aceleración de redes neuronales convolucionales mediante el uso de herramientas como High-Level Synthesis (HLS) y Vitis AI, que facilitan el diseño y despliegue de modelos en FPGA sin requerir conocimientos profundos de hardware.

El trabajo presenta dos enfoques principales para implementar CNN en FPGA: cuantización y reducción de pesos. La cuantización reduce el tamaño de los datos y pesos, mientras que la reducción de pesos elimina parámetros redundantes mediante técnicas como poda neuronal, eliminando neuronas o conexiones con pesos cercanos a cero, simplificando la estructura de la red.

Este enfoque logró mejorar la velocidad de inferencia del modelo Yolov4, alcanzando un rendimiento 72.5 veces más rápido que una CPU en el estudio, al mismo tiempo que reducía significativamente el consumo de recursos lógicos.

Hardware Implementations of a Deep Learning Approach to Optimal Configuration of Reconfigurable Intelligence Surfaces [20]

El artículo presenta una solución para la reconfiguración óptima de superficies inteligentes reconfigurables utilizando técnicas de inteligencia artificial (IA) y aprendizaje profundo (DL). Utilizando la herramienta Intel® FGPA AI Suite.

En este, los autores proponen una red neuronal convolucional (CNN) personalizada, capaz de determinar el desplazamiento de fase óptimo para cada celda de la RIS basándose en un ángulo de reflexión deseado. Para mejorar el entrenamiento, se desarrolla una función de pérdida personalizada que considera configuraciones equivalentes de la RIS, donde múltiples combinaciones de celdas pueden producir el mismo resultado.

El estudio compara implementaciones de su red neuronal en diferentes dispositivos de edge computing, incluyendo CPUs, GPUs, TPUs y FPGAs. Los resultados muestran que las FPGAs ofrecen el mejor rendimiento, con un aumento de hasta 20 veces en velocidad comparado con opciones aceleradas por GPU y casi 3 veces más que las implementaciones cuantizadas en INT8 con TPUs.

Capítulo 3

Métodos y materiales

En este capítulo se profundizará en las distintas tecnologías que forman parte de este proyecto, definiendo conceptos que se aplicarán en el capítulo 4, proporcionando un análisis detallado de su relevancia y aplicación en el contexto de este Trabajo Fin de Máster.

3.1. ROS

ROS, por sus siglas en inglés, Robot Operating System, es un entorno de desarrollo de software de código abierto para sistemas robóticos [21]. Una de sus principales virtudes radica en la abundancia de librerías y repositorios que facilitan la programación y la extracción de datos del robot.

ROS se caracteriza por su arquitectura modular basada en nodos, que facilita la reutilización y el intercambio de código entre diferentes proyectos y equipos. Cada nodo en ROS es un proceso que realiza una función específica y puede comunicarse con otros nodos a través de mensajes. Esta comunicación se gestiona mediante un sistema de publicación-suscripción y servicios que permite una integración fluida y escalable de múltiples componentes robóticos

3.1.1. El ecosistema de ROS en el proyecto

El módulo de seguimiento de este proyecto se divide a su vez en dos submódulos con tareas distintas pero dependientes entre sí, por una parte, el submódulo de captura compuesto por una Raspberry Pi 4 y una cámara RGBD Kinect V1, será el encargado de capturar las imágenes en el formato clásico RGB y las imágenes capturadas por la cámara infrarrojos, por consiguiente, el tratamiento y procesamiento de estás se realizará en una máquina virtual que aplicará técnicas de reconocimiento de objetos (Yolo) y cálculos de triangulación óptica para estimar la posición relativa del objetivo a rastrear.

Esto crea la necesidad de conectar en tiempo real estos dos submódulos para conseguir un rastreamiento con la máxima precisión posible. Como solución, ROS ofrece la posibilidad de establecer una jerarquía de dispositivos dentro de un mismo ecosistema, siendo el maestro de esta, el que establece el reloj de tiempo común encargado de sincronizar los módulos.

Para este proyecto reloj de sincronización será gestionado por el submódulo compuesto por la cámara y la Raspberry Pi ya que es quién captura el dato y por tanto quién inicia el proceso de rastreamiento. A partir de este punto, nos referiremos al submódulo de captura (Kinect V1 + Raspberry Pi 4) cómo 'Master o maestro' y el submódulo de procesamiento (Yolo + CPU ó Acelerador Hardware) cómo 'Slave o esclavo'

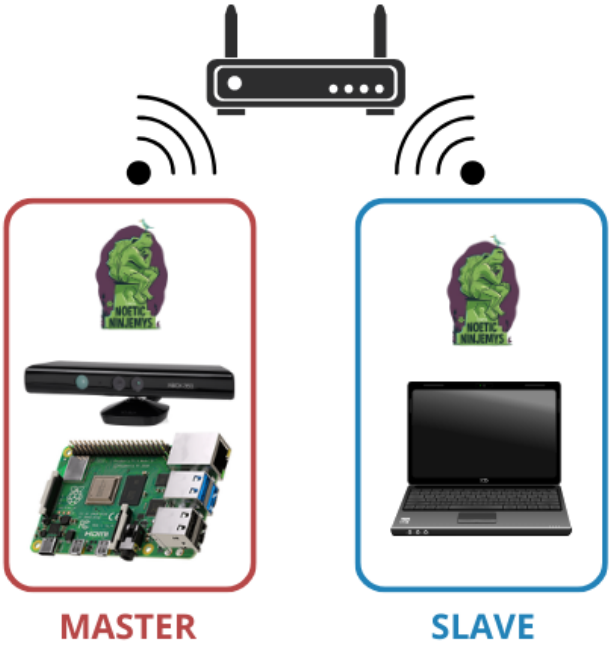

Figura 3.1: Infraestructura de red del proyecto

3.1.2. Nodos y tópicos

Un aspecto fundamental del funcionamiento de ROS son los Nodos y los Tópicos, elementos indispensables dentro de este entorno y que son a su vez base fundamental del control de los distintos elementos de los dispositivos.

Nodos:

Un nodo en ROS es un proceso diseñado para llevar a cabo una acción específica. Esto significa que, dentro de ROS, cada elemento puede ser comprendido como una entidad única que opera de forma independiente, ya sea recibiendo y/o enviando datos. Este enfoque favorece una programación modular, facilitando así la gestión de cada uno de los dispositivos, por ejemplo, en nuestro caso el Kinect V1 será gestionado por un nodo que activará los *drivers* de la cámara capturando y emitiendo datos a toda la red de ROS mediante los tópicos.

- **Nodo publicador:** la labor de este tipo de nodos es la emitir datos mediante los tópicos, comúnmente estos nodos son reservados para módulos relacionados con la extracción de datos.

- **Nodo subscriptor:** este tipo de nodos solo reciben datos suscribiéndose a tópicos de la red, normalmente generados por otros nodos, su funcionalidad principal es la de procesar esa información para accionar un actuador dentro del sistema.

- **Nodo publicador-subscriptor:** estos nodos son una mezcla de los dos anteriores, reciben datos de otros nodos y emiten otros datos a la red una vez procesados.

Tópicos:

Un tópico en ROS es un bus virtual de datos con nombre mediante el cuál los nodos de ROS intercambian información, estos buses son unidireccionales lo que implica que nodo publicador no conoce que nodos se suscriben a su tópico y por ende no recibe ningún *feedback* sobre este.

3.2. Kinect V1

El Kinect V1 es un periférico de la consola Xbox 360 lanzado por Microsoft en junio de 2011, este dispositivo forma parte de la familia de periféricos para consolas tipo *Motion Capture* que empezaron a promocionarse en el medio a mediados de la primera década del siglo XXI. En esta familia encontramos al famoso WiiMotion o al PlayStation Move, cada uno implementa tecnologías distintas entre sí, aunque todos buscan una misma aplicación, la captura de movimiento del usuario [22].

Centrándonos en el Kinect V1 este es un dispositivo tipo cámara RGBD (Red Green Blue + Distance) esto implica que además de las capacidades de una cámara convencional que captura el patrón de luz de una escena, las cámaras RGBD disponen de sensores que pueden medir la profundidad de la escena permitiendo generar un mapa 3D de esta en tiempo real sin necesidad de más sensores.

3.2.1. Características del dispositivo

El Kinect V1 consta principalmente de tres sensores que permiten capturar el tipo de datos previamente mencionados [23].

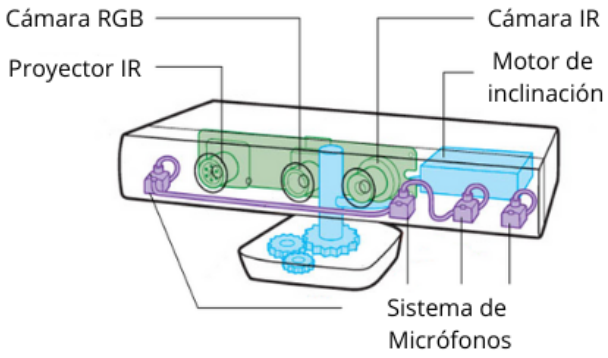

Figura 3.2: Estructura interna del Kinect V1

- **Proyector IR:** El dispositivo es un proyector láser no modulado, lo que significa que el haz de luz emitido se mantiene constante, sin cambios en su frecuencia o amplitud. La frecuencia de onda del haz es de 830 nm, siendo subdividida por un sistema de difracción óptico que proyecta un patrón de puntos pseudo-aleatorio.
- **Cámara IR:** El dispositivo es un sensor CMOS Aptina MT9M001 cuya banda de paso es de aproximadamente 830 nm lo que imposibilita usarlo como sensor térmico. La estimación de profundidad la realiza de la siguiente forma:
  - Una vez proyectados los puntos en la escena, la cámara IR captura el patrón generado.
  - El dispositivo tiene almacenado un 'patrón base' generado a una profundidad fija conocida. Este patrón se utiliza como referencia para comparar con la nueva imagen.

- Sabiendo la separación entre proyector IR y cámara IR (7.5 cm), el dispositivo buscará la coincidencia más cercana entre la ventana actual y la referencia, proporcionando un valor de desplazamiento (disparidad), que indica cuán lejos está un píxel en la imagen con respecto a la profundidad conocida.

- **Cámara RGB:** La cámara RGB es un sensor CMOS Aptina MT9M112, cuya función es la misma que la de cualquier cámara RGB convencional.

3.3. Raspberry Pi 4

La Raspberry Pi 4 es un minicomputador desarrollado por la Fundación Raspberry Pi, está construida a partir de un procesador Broadcom BCM2711, quad-core Cortex®-A72 (ARM® v8) de 64 bits a 1.5 GHz. Dispone de distintas versiones respecto a la memoria RAM de la que dispone, en el caso de nuestro proyecto se trata de la versión de 4GB LPDDR4.

El sistema operativo construido en su interior es Ubuntu 20.04 (Focal Fossa), la cual es compatible con la versión Noetic de ROS (necesaria para el despliegue de nuestro sistema).

La principal función de la Raspberry Pi dentro del proyecto es actuar como intermediario entre el hardware de la RIS y la red neuronal Yolo. Esto se logra mediante el envío de las imágenes capturadas por la cámara Kinect V1 hacia la red Yolo, utilizando el sistema de publicación-suscripción de ROS. La incorporación de este dispositivo aporta funcionalidades de tipo middleware, facilitando la conexión e integración de periféricos con el sistema general.

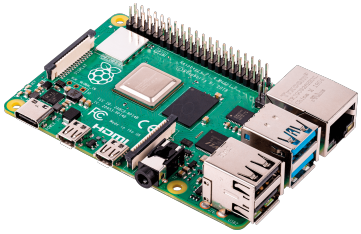

Figura 3.3: Raspberry Pi 4

3.4. Kit de desarrollo SoC Arria® 10 SX

Intel Arria 10 es una familia de dispositivos FPGA (Field Programmable Gate Array) y SoC (System on Chip) que se caracteriza por ofrecer un alto rendimiento y bajo consumo de energía. Están fabricados con tecnología de 20 nanómetros y representan una mejora importante respecto a generaciones anteriores, ya que ofrecen mayor velocidad y eficiencia [24].

El SoC está construido a partir de un procesador ARM Cortex-A9 dual-core a 1.2 GHz (hasta 1.5 GHz). Por otra parte, la FPGA interna consta de una arquitectura lógica basada en hasta 660 mil elementos lógicos (dependiendo del modelo), más de 250 mil ALMs, y una gran cantidad de registros y bloques de memoria integrados, como M20K y MLAB, que proporcionan almacenamiento interno eficiente. Además, incluye bloques DSP de precisión variable, transceptores de alta velocidad (hasta 17.4 Gbps).

Su propósito será el de alojar los aceleradores hardware para realizar las inferencias con el modelo de Yolo.

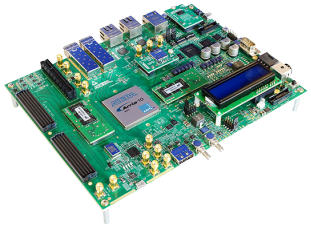

Figura 3.4: Kit de desarrollo SoC Arria® 10 SX

3.5. Step Motor Driver TMC5130

El TMC5130A es un circuito integrado que combina funciones de controlador y driver para motores paso a paso, con soporte para comunicación serial [25]. Incorpora un generador de rampas programable que permite realizar movimientos hacia posiciones objetivo mediante perfiles de velocidad y aceleración definidos por el usuario. Esto permite que el motor se desplace de forma controlada y autónoma una vez configurados los parámetros de movimiento.

El dispositivo mantiene un registro interno de la posición actual del motor en micro-pasos, lo que permite calcular con precisión cuántos pasos ha ejecutado y cuántos faltan para alcanzar una posición de destino. Por último, gracias al sistema StealthChop, el control de corriente es dinámico y adaptativo, lo que contribuye a un funcionamiento silencioso y estable, especialmente en aplicaciones sensibles al ruido o a la vibración.

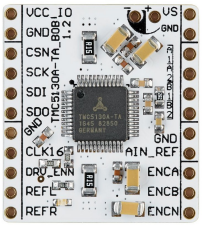

Figura 3.5: Step Motor Driver TMC5130-TA BOB [26]

3.6. Cmod A7-35t: Breadboardable Artix-7 FPGA Module

El Cmod A7 es un módulo compacto (0.7"x 2.75") con formato DIP de 48 pines, diseñado para facilitar la integración de una FPGA Xilinx Artix-7 en proyectos de diseño de circuitos y prototipado [27].

Este módulo integra una FPGA XC7A35t-1CPG236C, fabricada con tecnología de 28 nanómetros, que proporciona un equilibrio óptimo entre rendimiento, consumo energético y coste. El dispositivo cuenta con aproximadamente 20.800 celdas lógicas, 225 KB de memoria en bloques RAM y 41.600 flip-flops.

El Cmod A7-35t dispone de una memoria flash Quad-SPI de 128KB, un reloj de sistema de 12 MHz, y una interfaz USB-JTAG/UART integrada, lo que facilita tanto la programación como la depuración del sistema sin necesidad de hardware adicional (Tabla 3.1).

En este proyecto, el Cmod A7 se encargará de gestionar las posiciones individuales de cada elemento del sistema a partir de símbolos de 2 bits, permitiendo formar el patrón deseado en el panel de la RIS (Reconfigurable Intelligent Surface).

Para ello, se cuenta con el uso de nueve módulos Cmod A7-35t, cada uno encargado de controlar

25 motores paso a paso mediante el protocolo SPI, lo que da un total de 225 motores. La función principal de cada módulo será traducir los símbolos de 2 bits, generados por el sistema de detección y seguimiento, en tramas de configuración interpretables por el controlador TMC5130 (Step Motor Driver).

En el siguiente capítulo se detallará la implementación de los módulos descritos en VHDL necesarios para cumplir con esta tarea, incluyendo la lógica de decodificación, el generador de tramas SPI y la secuencia de sentencias para gestionar el TMC5130.

Las capacidades técnicas del dispositivo se resumen en la Tabla 3.1:

Característica	**Especificación**
FPGA	XC7A35t-1CPG236C
LUTs	20.800
Flip-Flops	41.600
Block RAM	225 KB
Clock Management Tiles (CMTs)	5 (MMCM/PLL)
Reloj interno	Cristal de 12 MHz

Tabla 3.1: Resumen técnico del módulo FPGA Cmod A7-35t [27]

3.7. Intel® FGPA AI Suite

Intel® FGPA AI Suite es una herramienta desarrollada por Intel que facilita a ingenieros en FPGA, ML e IA el diseño de componentes optimizados para la inferencia de modelos de IA implementados en hardware dentro de una FPGA.[28]

Una de sus principales ventajas es la compatibilidad con frameworks de IA ampliamente utilizados, como TensorFlow, PyTorch, Caffe, ONNX y el ecosistema OpenVINO. Esto permite a los desarrolladores importar modelos preentrenados sin necesidad de reentrenarlos ni modificarlos.

Esta herramienta es ideal para aplicaciones en tiempo real donde se requiere baja latencia, como visión artificial, control industrial, dispositivos médicos, robótica y otros sistemas inteligentes. Es compatible con una amplia gama de dispositivos FPGA de Intel, incluidos los Agilex 5 y 7, Cyclone 10 GX y Arria 10, lo que la convierte en una herramienta versátil para distintas soluciones que requieran del uso de modelos de inteligencia artificial.

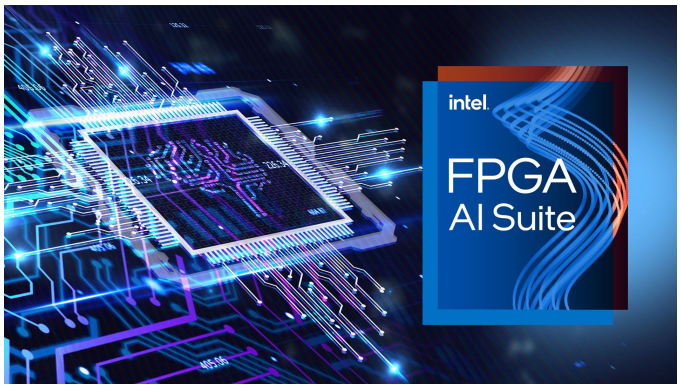

Figura 3.6: Logotipo FPGA AI Suite

3.8. Yolov3

El algoritmo encargado/implementado para la tarea de reconocimiento de objetos en este proyecto es el modelo de YoloV3. Tal y como se explica en la sección 2.2.3, Yolo es un algoritmo de detección de objetos que en una sola iteración por la red neuronal predice las ubicaciones de los objetos dentro de la escena y sus respectivas clases .

3.8.1. Arquitectura del modelo

El modelo de Yolov3 se inspira en una red llamada Darknet-53 como su arquitectura principal, encargada de la extracción de características de la imagen. Esta red tiene 53 capas convolucionales e incorpora conexiones residuales, una técnica heredada de ResNet, que ayuda a evitar el sobreajuste en redes de *deep learning* al permitir el flujo de información a través de saltos entre capas de la red [29].

Cada capa convolucional emplea un stride de 2, lo que significa que el filtro se desplaza dos píxeles a la vez sobre el mapa de características durante la operación de convolución. Este desplazamiento en pasos de dos unidades permite reducir tanto el tamaño de la salida como el tiempo de procesamiento, al disminuir la cantidad de operaciones a realizar. Seguidas por una normalización por lotes (batch normalization) y una función de activación ReLU. Esta normalización es necesaria para escalar la salida de la capa anterior, estabilizando la distribución de los datos al interior de la red. Por otra parte, las capas ReLU introducen no linealidad al modelo, lo cual es esencial para que la red aprenda representaciones complejas. ReLU transforma los valores negativos en cero y deja pasar los positivos sin modificar, lo que además ayuda a evitar saturaciones y mejora la eficiencia del entrenamiento.

	Type	**Filters**	**Size**	**Output**
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3/2$	128×128
	Convolutional	32	1×1	
$1\times$	Convolutional	64	3×3	
	Residual			128×128
	Convolutional	64	1×1	
$2\times$	Convolutional	128	3×3	
	Residual			64×64
$3\times$	Convolutional	256	$3 \times 3/2$	32×32
	Convolutional	128	1×1	
$8\times$	Convolutional	256	3×3	
	Residual			32×32
$8\times$	Convolutional	512	$3 \times 3/2$	16×16
$4\times$	Convolutional	512	$3 \times 3/2$	8×8
	Convolutional	512	3×3	
	Convolutional	512	3×3	
	Convolutional	512	3×3	
	Avgpool		Global	
	Connected	1000		
	Softmax			

Tabla 3.2: Arquitectura del modelo de Yolov3 [29]

3.8.2. Formato de salida del modelo

Una de las características más importantes de Yolov3 es que realiza detecciones en tres escalas diferentes. Esto soluciona el problema de detectar objetos pequeños, ya que encadena las salidas de capas previas para preservar características más detalladas (fine-grained features).

Yolov3 utiliza tres anclas por escala, por lo que predice tres cajas delimitadoras (bounding boxes) por celda (n_anchors = 3). Por cada una de estas cajas, se genera un vector de tamaño 5 + número de clases. El valor 5 corresponde a las cuatro coordenadas de la caja (bx, by, bh, bw) junto con una probabilidad de confianza (pc) que indica la posibilidad de que exista un objeto en esa celda. A esto se le suman las probabilidades asociadas a cada una de las clases posibles (Figura 3.7).

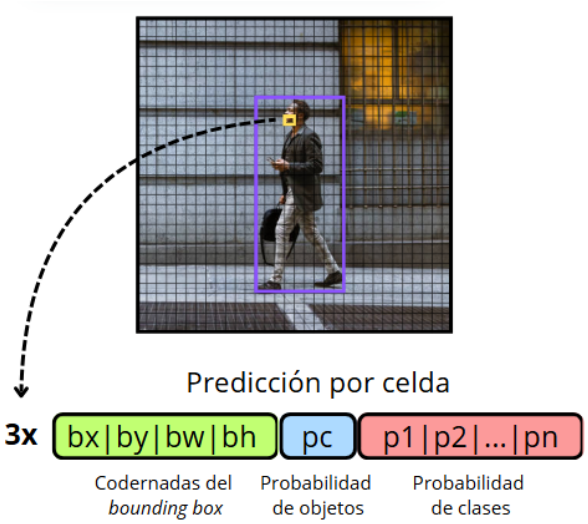

Figura 3.7: Formato de salida para Yolov3

Escalas de detección

La red reserva 3 escalas para la detección de características de distinto tamaño. Estas escalas deben ser divisoras del tamaño de la imagen de entrada. El tamaño estándar de entrada para el modelo YoloV3 es de 416×416 píxeles, aunque existen variaciones del modelo que permiten entradas de 640×640.

Al depender de una entrada de 416×416 y el tamaño del grid será de 13×13, 26×26 y 52×52, que generarán 169, 676 y 2704 celdas de 32×32, 16×16 y 8×8 píxeles respectivamente, estas celdas llevan asignadas 3 anclas distintas por escala de salida. Las anclas en Yolov3 representan formas y tamaños típicos de objetos que se esperan detectar en una imagen. En lugar de que la red prediga desde cero la posición y tamaño de cada objeto, parte de estas anclas predefinidas y aprende a ajustarlas (es decir, a predecir desplazamientos y escalas) para que se ajusten a los objetos reales en la imagen. Las anclas asociadas a cada formato de salida son las siguientes:

Tamaño del grid (Shape)	Anclas (Anchors)
13 × 13	(116,90), (156,198), (373,326)
26 × 26	(30,61), (62,45), (59,119)
52 × 52	(10,13), (16,30), (33,23)

Tabla 3.3: Resumen de anclas por tamaño del grid

3.8.3. Cálculo de *bounding boxes*

En Yolov3, el cálculo del bounding box (caja delimitadora) parte de predicciones relativas a cajas ancla (anchor boxes) y a las coordenadas de caja (bx, by, bh, bw) resultado de la inferencia. La red no predice directamente las coordenadas absolutas del objeto, sino transformaciones que luego se ajustan a una celda del mapa de salida.

$$BB_x = \sigma(b_x) + c_x \tag{3.1}$$

$$BB_y = \sigma(b_y) + c_y \tag{3.2}$$

$$BB_w = p_w \cdot e^{b_w} \tag{3.3}$$

$$BB_h = p_h \cdot e^{b_h} \tag{3.4}$$

Estas fórmulas caracterizan el *bounding box* donde:

- BB_x, BB_y : coordenadas del centro del *bounding box*, relativas a la celda en el mapa.
- BB_w, BB_h : ancho y alto del *bounding box*.
- b_x, b_y, b_w, b_h : salidas crudas de la red.
- $\sigma()$: Representa la función *sigmoide* que unifica el valor en un rango de [0-1].
- c_x, c_y: Posición de la celda (fila, columna).
- p_x, p_y: Valor de ancla correspondiente.

Este cálculo es post-procesamiento ya que el modelo genera únicamente los valores b_x, b_y, b_w, b_h.

3.8.4. Cálculo de puntuaciones y detección final

En la detección de imagen de tamaño 416 × 416, Yolo predice:

$$N_{Cajas} = (52 \cdot 52) \cdot (26 \cdot 26) \cdot (13 \cdot 13) = 10.647\, cajas$$

Sin embargo, en la práctica, la imagen contiene solo unos pocos objetos reales. Por ello, es necesario reducir esas miles de predicciones a las verdaderas cajas usando lo siguiente:

Filtrado por umbral

Tal y cómo se ha comentado, en muchas ocasiones las *bounding boxes* generadas no contienen nada, por lo que para evitar post-procesar información no relevante utilizaremos la variable de probabilidad de confianza (pc). A partir de un umbral diseñado por el usuario, se establece qué valor de confianza es límite para que se considere esa caja como útil. Esta variable está en formato normalizado, por lo que un umbral de 0.7 nos indicará una probabilidad de objeto del 70 %. Dependiendo del entrenamiento y el tipo de imagen que vamos a procesar se elegirá un valor más alto o más bajo.

Non-Maximum Suppression (NMS)

Una vez filtradas las cajas que aportan información, aún existen cajas que quedan superpuestas para un mismo objeto. Esto genera redundancia, y si no se maneja adecuadamente, puede afectar la precisión y claridad del sistema de detección. Para eliminar esto se aplica NMS:

- 1. Selecciona la caja con la puntuación de confianza más alta entre todas las detectadas.
- 2. Calcula el IoU (Intersección sobre Unión) entre esta caja seleccionada y todas las demás cajas.

$$IoU(A, B) = \frac{Area_{interseccion}(A \cap B)}{Area_{union}(A \cup B)} \tag{3.5}$$

- 3. Descarta todas las cajas cuya IoU con la caja seleccionada sea mayor a un umbral, ya que se consideran detecciones duplicadas del mismo objeto.

Capítulo 4

Desarrollo e implementación

4.1. Descripción del sistema completo

Cómo punto de partida de este proyecto, se plantea la necesidad de desarrollar e implementar un sistema que permita configurar de forma dinámica un dispositivo tipo RIS. Tal y como se explora en el capítulo 2, una de las propuestas presentadas como hardware integrado para RIS, es la reconfiguración mediante el uso de cámaras/sensores visuales.

El flujo de diseño debe contar con la capacidad de detectar el objetivo, definirlo espacialmente y reconfigurar la RIS con el panel óptimo para la situación. Para lograr esto, el diseño debe comenzar por definir el tipo de imagen que se recibirá y procesará. En nuestro caso, al utilizar una cámara Kinect V1 (ver Sección 3.2), la entrada estará compuesta por dos tipos de imágenes: una imagen RGB convencional y una imagen de profundidad (este tipo de imagen es una matriz de datos que asocia los píxeles con un valor de profundidad medido).

Combinando ambas imágenes y la implementación de un modelo de detección de objetos se realizará una estimación espacial del objetivo mediante óptica. Descrito el objetivo y calculado su ángulo respecto a la RIS, se escogerá un panel (ya sea por un dataset predefinido o mediante un modelo de IA) y se enviará vía SPI al dispositivo para su reconfiguración.

Estas tareas se dividirán en dos módulos principales. El primero se enfocará en la recepción del panel a reconfigurar mediante comunicación SPI, lo que implicará el desarrollo e implementación de diversos componentes hardware sobre FPGA. El segundo módulo se encargará de las funciones relacionadas con la reconfiguración de la RIS, incluyendo la captura de imágenes, la detección de objetivos, la estimación espacial y la selección del panel óptimo. Ambos siempre siguiendo el flujo de trabajo de la Figura 4.1

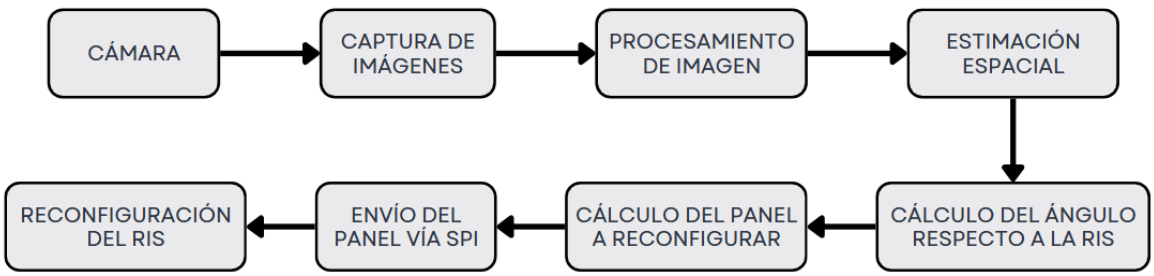

Figura 4.1: Diagrama de funcionamiento del sistema

4.2. Diseño en VHDL del módulo de comunicación SPI

Para conseguir lo anteriormente descrito, el proyecto requiere de la implementación de una tecnología que actúe como interfaz de comunicación entre el sistema encargado de generar la configuración y los elementos que integran la superficie inteligente reconfigurable (RIS).

La RIS que actúa de plataforma en este proyecto, era una RIS motorizada, es decir, los elementos cambian de estado (posición) mediante el movimiento de motores. Esto implica que su funcionamiento depende directamente de la precisión espacial de cada motor. Además, la posibilidad de que los motores adopten múltiples posiciones genera un elevado número de configuraciones potenciales a caracterizar. Con el fin de simplificar y reducir el volumen de configuraciones posibles, se opta por diseñar el control simbolizando los estados posibles en 2-bits (4 posiciones posibles).

Esto implica que cada panel recibido por el módulo estará compuesto por valores en el rango [0–3], los cuales deben ser interpretados y traducidos a una posición espacial individual para cada elemento que pueda ser interpretada por el driver TMC5130 (sección 3.5).

Finalmente, para maximizar las capacidades de la RIS, los valores espaciales asignados a los símbolos de 2-bits de cada elemento deben ser configurables individualmente, esto con el fin de paliar posibles defectos mecánicos o con el fin de probar distintas caracterizaciones del dispositivo. Partiendo de estas premisas, el sistema se dividirá en un módulo *SLAVE* encargado de recibir las configuraciones del panel mediante el protocolo SPI y un módulo *MASTER* que enviará las posiciones a alcanzar a los elementos de la RIS:

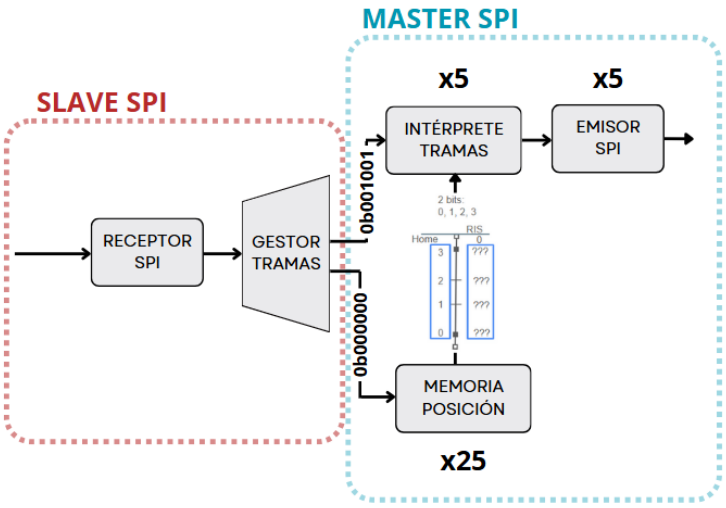

Figura 4.2: Diagrama módulo de comunicación SPI

4.2.1. Diseño de tramas de comunicación

Una vez planteado cuáles van a ser los retos a cumplir en este módulo, necesitamos crear un sistema de tramas que nos permita identificar qué información estamos mandando o recibiendo.

Tramas del generador de configuraciones al Módulo Slave SPI

El primer aspecto a abordar es la forma en que se recibirá la información de reconfiguración correspondiente a los elementos de la RIS. El principal reto consiste en desarrollar un protocolo de tramas que permita la configuración individual de 225 motores. Este protocolo debe ser eficiente tanto en

términos de tiempo como de volumen de datos, y además, debe mantener un tamaño razonable para minimizar el riesgo de pérdida de datos durante la transmisión o errores en el procesamiento por parte del receptor.

Como se mencionó anteriormente, los 225 motores son controlados por 9 FPGAs CMOD A7-35t (ver sección 3.6), por lo que cada FPGA se encargará de gestionar 25 motores. Esta distribución permite simplificar considerablemente el diseño de las tramas, pudiendo reducirlas a paquetes de 7 bytes (es decir, 56 bits por trama).

Además, debido a limitaciones mecánicas, el recorrido útil de cada motor tras el montaje está acotado al rango de [0–510 pasos]. Por ello, cada símbolo convertido a pasos puede representarse con una codificación de 9 bits. Dado que se emplean 4 símbolos por trama, se requieren 36 bits en total, lo cual se ajusta perfectamente al tamaño estándar de 7 bytes por trama. A partir de estas premisas, podemos estandarizar la trama, tal y como se presenta en la Figura 4.3.

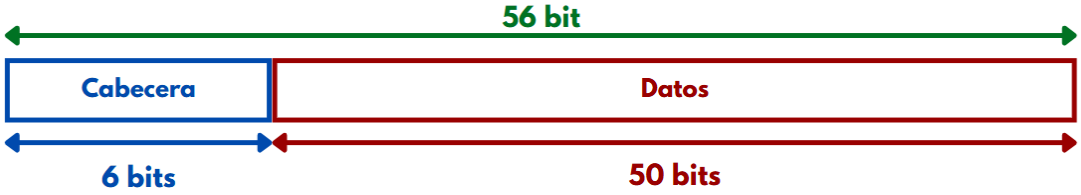

Figura 4.3: Trama estándar para comunicación

En concreto, para las tramas destinadas a transmitir los datos de reconfiguración del panel, se utilizará la cabecera "`0b001001`", la cual indicará que la trama recibida corresponde específicamente a este tipo de información. Por otra parte, el bloque de datos estará compuesto por los símbolos correspondientes a los 25 motores adscritos a la FPGA, asignando los bits menos significativos a los motores con una numeración menor y viceversa (2 bits por motor), Figura 4.4.

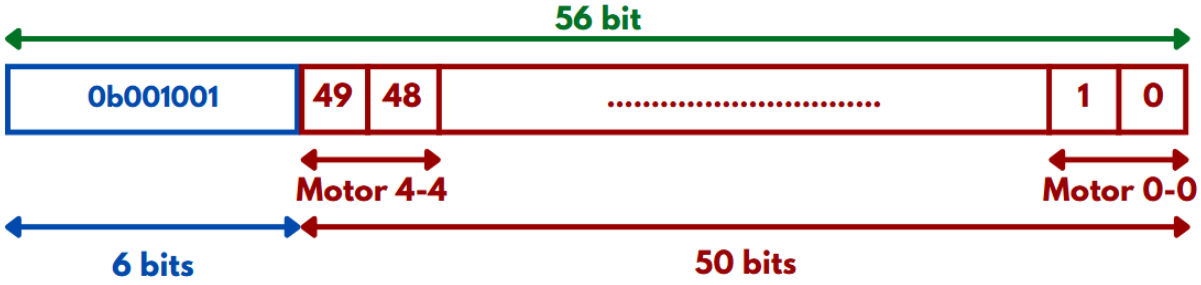

Figura 4.4: Trama de reconfiguración del panel

En el caso de las tramas destinadas a la reconfiguración de símbolos, el diseño resulta más complejo, ya que es necesario identificar específicamente a qué motor se le aplicará el cambio de valor. Por lo tanto, además de transmitir los nuevos valores de reconfiguración, la trama debe incluir un identificador del motor correspondiente; para diferenciarlas de las tramas anteriores, su cabecera tendrá el valor "`0b000000`".

Dado que cada FPGA gestiona 25 motores, una solución lógica sería utilizar 5 bits para codificar dicho identificador ($2^5 = 32$ posibles combinaciones, suficientes para cubrir los 25 motores). Esto, junto con los 36 bits destinados a los nuevos símbolos, encajaría dentro del límite de los 56 bits (7 bytes) por trama.

Sin embargo, aunque este enfoque es conceptualmente sencillo, su implementación puede resultar compleja. Para ello se propone usar 10 bits, los 5 más significativos de este grupo asignados al

número de MASTER (se profundizará en este concepto en la sección 4.2.3) y los 5 menos significativos a número de motor. Concretando el **Motor 0-0** tendrá asignado en este bloque el valor binario "`0b0000100001`"mientras que el **Motor 4-4** "`0b1000010000`". Mediante esta lógica se referencian todos los elementos de cada FPGA resultando en el siguiente modelo trama:

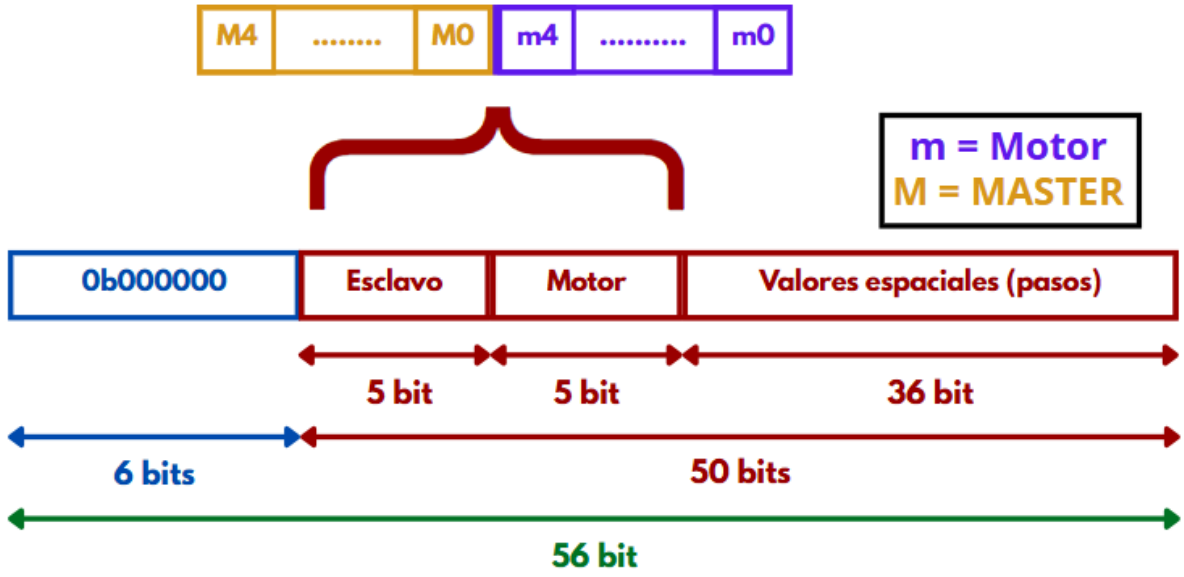

Figura 4.5: Trama de reconfiguración de símbolos

Tramas del Módulo Master SPI hacia el Driver TMC5130

Para comunicarnos con el driver TMC5130, utilizamos el conjunto de tramas descrito en su hoja de datos [25]. Este dispositivo admite más de 50 instrucciones de configuración, cada una identificada por una cabecera de 8 bits. Para simplificar este apartado, nos enfocaremos únicamente en las tramas relacionadas con las transiciones entre posiciones. Estos son los registros de configuración ***XACTUAL*** y ***XTARGET***.

El registro ***XACTUAL*** es una variable interna del driver que gestiona cuál es la posición relativa en pasos en la que se encuentra el motor; esta variable se calcula a partir de las conmutaciones del **puente H** que posee el driver. Esta variable puede ser escrita o leída y permite reiniciar el origen del que parte el motor, función realmente útil para establecer una calibración en los elementos mecánicos.

El registro ***XTARGET*** es una variable interna del driver que le indica a este cuáles son los pasos objetivo a dar. Es decir, a partir del valor de ***XACTUAL*** indica al driver de qué forma debe conmutar el **puente H** para llegar a los pasos deseados.

4.2.2. Módulo SPI SLAVE

Dentro del hardware implementado para la comunicación SPI, el módulo *SLAVE* es el encargado de recibir y almacenar las tramas que llegan a través del bus. Este módulo se compone de dos subcomponentes, cada uno con funciones simplificadas. Conceptualmente, el módulo se representa mediante el esquemático de la Figura 4.6.

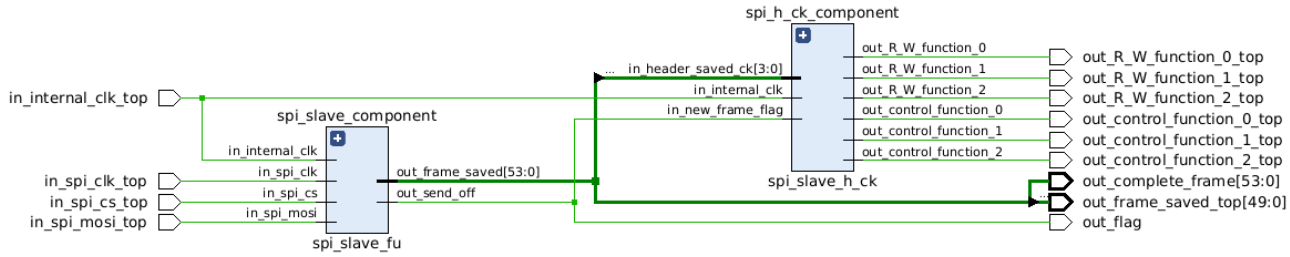

Figura 4.6: Esquemático RTL módulo SLAVE

Componente *spi_slave_fu*:

Este componente recibe su nombre por ser la unidad funcional (Functional Unit, FU) del módulo SLAVE. Su función principal es adquirir los 56 bits transmitidos a través del bus SPI. Su descripción funcional es (se refleja en la Figura 4.7):

1. Se establece un STD_LOGIC_VECTOR del tramaño de la trama a recibir (en nuestro caso 56 bits) con valor lógico 0. Además se diseña un contador interno de valor 56 que registrará cuantos bit quedan por recibir.

2. Mediante una máquina de estados síncrona (controlada por una señal de reloj), se supervisa el canal CS (Chip Select) del bus SPI. Cuando CS se encuentra en nivel bajo (lógico 0), se interpreta como el inicio de una comunicación, activando así la máquina de estados. En cambio, cuando CS está en nivel alto (lógico 1), la recepción permanece deshabilitada. Una vez activada, la máquina de estados alterna entre sus dos estados en cada semi-ciclo del reloj.

3. **Primer estado "*Reading*":** En este primer estado, ejecutado en el flanco ascendente (Low to High) de la señal de reloj, el componente sobrescribirá todos los bits del STD_LOGIC_VECTOR desde la posición indicada por el contador de bits restantes hasta el bit menos significativo. Cada uno de estos bits se actualizará con el valor lógico recibido a través del canal MOSI de la comunicación SPI. De este modo, solo se reescriben los bits que aún no han sido recibidos, preservando los ya almacenados.

4. **Segundo estado: "*checking*"** En este segundo estado, que se ejecuta durante el flanco descendente (High to Low) de la señal de reloj, el componente verifica el valor del contador. Si el contador es igual a cero (lo que indica que se ha recibido la trama completa), se envía un '1' lógico a través de un canal interno para notificar al resto de los componentes que hay una nueva trama disponible. En caso contrario, se decrementa el contador en una unidad y el sistema permanece a la espera del siguiente flanco ascendente del reloj para almacenar el próximo bit recibido.

Componente *spi_slave_h_ck*:

Este es el segundo y último componente del módulo *SLAVE*, recibe su nombre por ser el componente encargado de comprobar la cabecera de la trama (header checker), su funcionamiento es relativamente sencillo. A partir de los 6 bits más significativos de la trama recibida, activa una señal interna encargada de encender o apagar los componentes a utilizar en el módulo MASTER, el sistema está preparado para detectar 16 cabeceras distintas aunque en la versión actual solo están en uso 2.

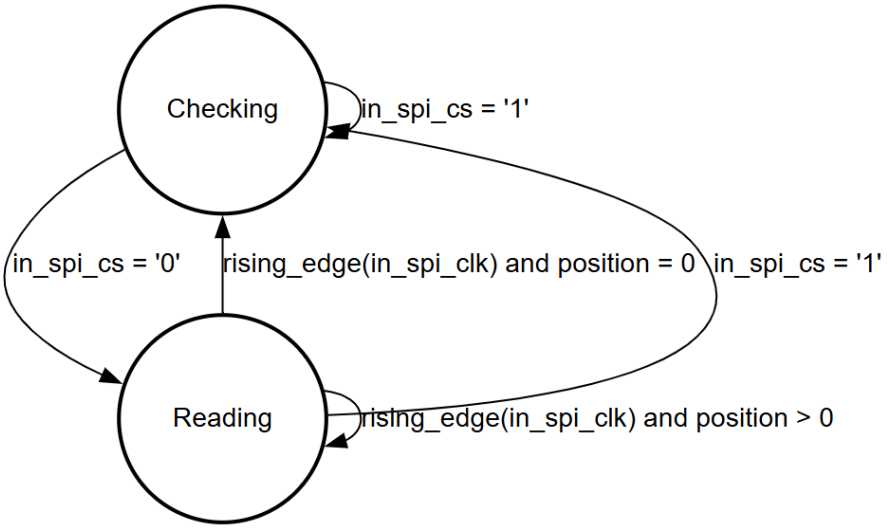

Figura 4.7: Diagrama de estados de funcionamiento del módulo SLAVE

4.2.3. Módulo SPI MASTER-MEMORIA

El módulo SPI MASTER se compone de 6 componentes con funciones individualizadas, con el fin de amenizar la explicación y haciendo referencia a la Figura 4.2, dividiremos el módulo SPI MASTER en 3 submódulos.

Tal como indica el título de esta sección, el primer submódulo a analizar es el submódulo de memoria, cuya función es almacenar los valores de paso correspondientes a los símbolos del panel. Cada unidad de memoria está diseñada para gestionar los datos de 5 motores. Esta decisión responde a la necesidad de optimizar los recursos disponibles en la FPGA Cmod A7-35t y afrontar el elevado número de dispositivos que deben ser controlados.

Para lograr esto, los módulos MASTER fueron concebidos para manejar únicamente **5 motores** cada uno. Como consecuencia, en el diseño de la trama de reconfiguración de símbolos, es necesario identificar a qué módulo MASTER se dirige la comunicación, ya que cada módulo solo conoce los motores que gestiona directamente.

En resumen, cada submódulo **SPI MASTER-MEMORIA** debe almacenar los valores en pasos correspondientes a los 4 símbolos del panel para 5 motores, lo que implica un total de **180 bits** de memoria por submódulo.

Descripción Funcional

Este submódulo, al tener un funcionamiento similar al de una memoria, debe cumplir con las funciones básicas asociadas a esta: almacenamiento, lectura y escritura de datos. El componente principal de esta memoria son los 180 biestables (flip-flops), encargados de conservar los 180 bits necesarios para representar los valores en pasos de los 4 símbolos del panel correspondientes a 5 motores.

A estos biestables se les añade la lógica de control necesaria para gestionar adecuadamente las operaciones de lectura y escritura, garantizando así que el submódulo funcione conforme a las especificaciones previamente mencionadas de una memoria digital. Se puede observar la descripción RTL del submódulo en la Figura 4.8.

A partir de las especificaciones previamente mencionadas, se procede a explicar cómo las resuelve este submódulo 4.8:

1. **Escritura de datos:** Esta funcionalidad implica el uso de la trama para reconfiguración de símbolo. Cuando el módulo esclavo SPI recibe la trama y detecta su cabecera, el componente `spi_slave_h_ck` activa el flag que certifica una trama de este tipo. A continuación, mediante puertas lógicas AND, se verifica que tanto el flag como el identificador del motor estén activos en el mismo ciclo de reloj. Si ambos se cumplen simultáneamente, la memoria almacena el conjunto de bits correspondiente al campo de datos de esa trama. El almacenamiento puede ser simultáneo permitiendo que una misma configuración de símbolos se guarde para distintos motores.

2. **Almacenamiento de datos:** Esta funcionalidad se basa en la definición lógica del biestable. La salida de las puertas AND previamente mencionadas se utilizará como la señal de control ENABLE del biestable clásico. La entrada, o estado actual, corresponderá al bit del bloque de datos recibido, mientras que la salida, o estado siguiente, representará el valor almacenado.

   Cuando la señal ENABLE coincida en un valor lógico alto con el flanco ascendente de la señal de reloj, el estado siguiente se cargará con el estado actual, almacenando el nuevo bit y sobrescribiendo el anterior. Mientras la señal ENABLE y el reloj no cumplan la condición anterior, el valor almacenará.

3. **Lectura de datos:** Para leer los datos de esta memoria se utiliza un multiplexor, tal y cómo se ha comentado, cada motor/elemento tiene asignados 36 bits de memoria por lo qué la salida de este multiplexor será también de 36 bits. Mediante una señal selectora de 5 bits, el multiplexor seleccionará una de 5 entradas de 36 bits y la dirigirá a su salida.

Cada salida está relacionada con la posición del bit que la selecciona, siendo el bit menos significativo el correspondiente al motor con la menor numeración.

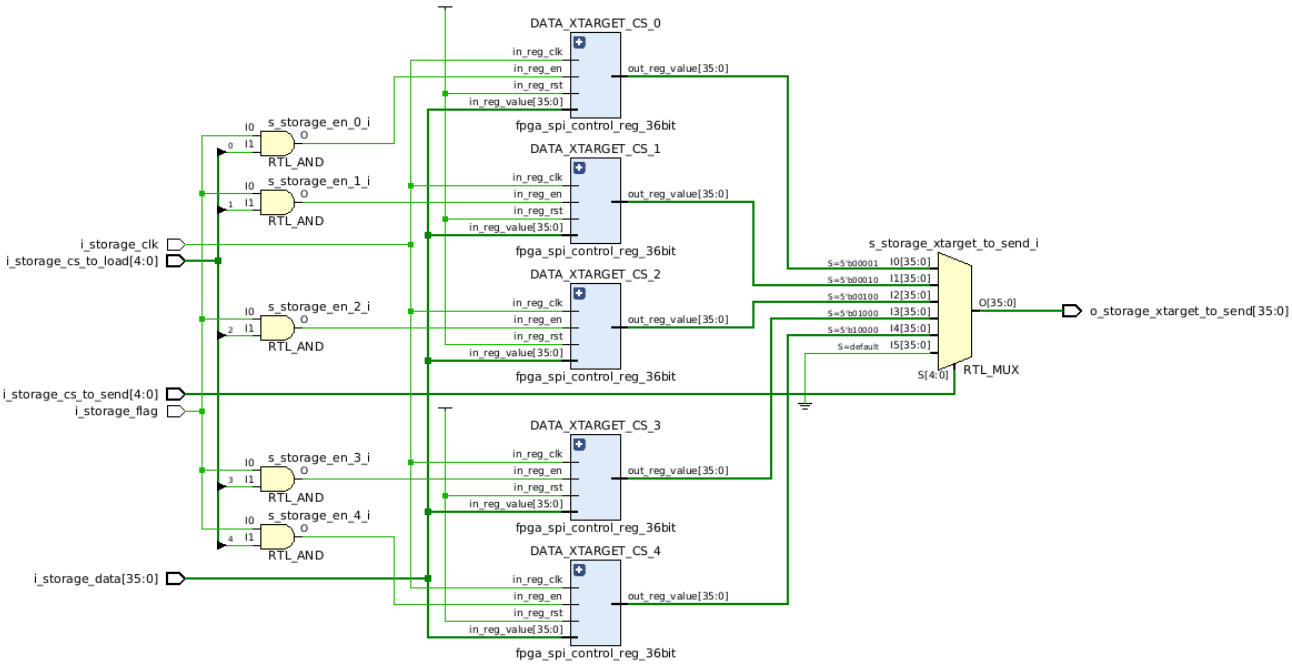

Figura 4.8: Esquemático RTL submódulo SPI MASTER-MEMORIA

4.2.4. Módulo SPI MASTER-INTERPRETE

Este módulo, únicamente es excitado al recibir la trama de reconfiguración del panel, su principal función es la de traducir el símbolo recibido en un valor en pasos que pueda entender el driver TMC5130. Para ello se implementan los siguientes componentes:

Target Manager (TM)

La función de este componente es la de dividir una entrada de 50 bits (el campo de la trama dedicado a datos) en 5 paquetes destinados a cada MASTER SPI (10 bits/MASTER SPI). Cada bloque de 10 bits contiene la información necesaria para configurar los símbolos de cada motor (Figura 4.9). La codificación binaria empleada asigna a cada símbolo una cadena de 2 bits: `0b00`" representa el símbolo '0', `0b01`" el símbolo '1', y así sucesivamente hasta el símbolo '3'.

Su descripción funcional es la siguiente (Figura 4.10):

1. **Estado inicial y de espera:** En este estado, el módulo permanece inactivo a la espera de una nueva solicitud. Al recibir una nueva configuración por el bus SPI, el componente se activa y cambia al siguiente al siguiente estado.
2. **División de paquetes:** En este segundo estado, una vez recibido el paquete de 50 bits, lo divide en 5 paquetes dirigidos a la entrada de cada MASTER SPI y activa una flag de salida indicando que hay nueva información a tratar.
3. **Comprobación y espera:** En este último estado el componente comprueba que los MASTER SPI se encuentre inactivos, en esta caso cambiará el valor de la flag de salida a '0´ y permanecerá a la espera de una nueva trama. En caso contrario, esperará a que los MASTER terminen su actividad para apagarse.

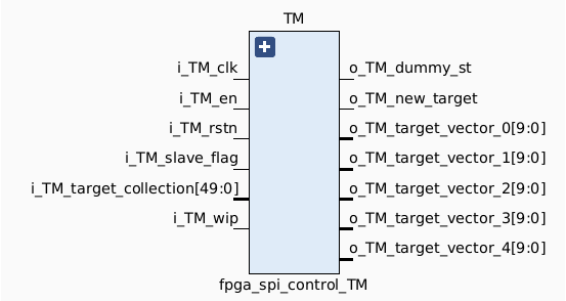

Figura 4.9: Esquemático RTL Target Manager (TM)

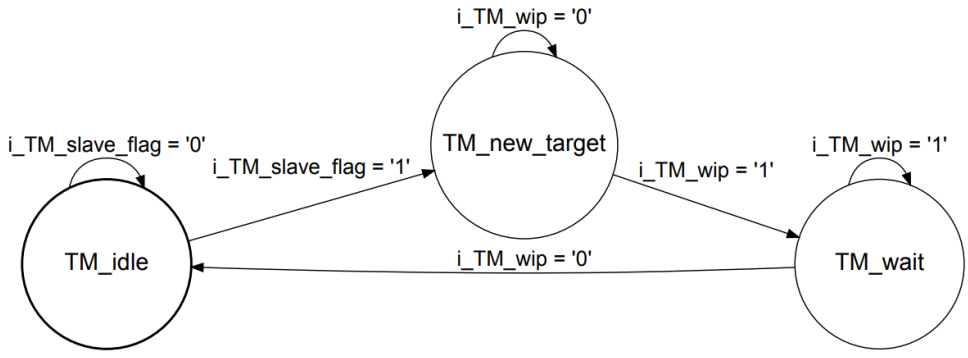

Figura 4.10: Diagrama de estados del módulo Target Manager (TM)

SPI Target Selector (TS)

Este componente es el encargado de descodificar los símbolos de entrada y transformarlos a un valor en pasos comprensible por el driver TMC5130. Esto lo realiza gracias a los valores guardados en la memoria descrita en la sección 4.2.3.

Al recibir el paquete de 10 bits con los símbolos de cada driver, el componente *TS* selecciona mediante una señal de entrada (codificada en 3 bits) con qué driver va a comunicarse, y extrae el símbolo correspondiente a transformar.

Mediante una señal selectora llamada *'o_ TS_ cs_ select'*, el componente se comunica con la memoria indicándole qué registro de 36 bits quiere leer. Una vez recibidos los valores guardados en la memoria, selecciona qué valor corresponde al símbolo a enviar, lo convierte a un paquete de 32 bits y lo envía por la salida *'o_ TS_ target_ value'*.

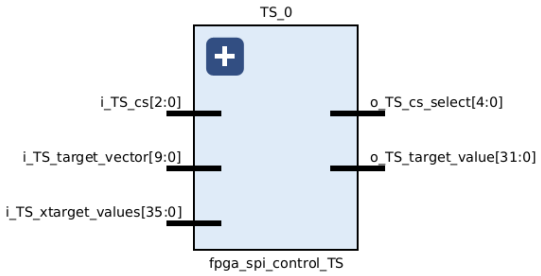

Figura 4.11: Esquemático RTL Target Selector (TS)

SPI Frame Builder (FB)

El módulo Frame Builder (FB) es un componente asíncrono encargado de construir la trama SPI completa de 40 bits para transmitir el número de pasos a recorrer por el motor en un formato entendible por el driver TMC5130. Esta trama se basa en la señal *'o_ TS_ target_ value'*, proveniente del componente Target Selector (ver sección 4.2.4). Para generar la trama, el módulo añade la dirección de instrucción correspondiente a *XTARGET* del controlador TMC5130. Finalmente, se ensambla la trama completa y la envía por el bus *o_ SPI_ data*.

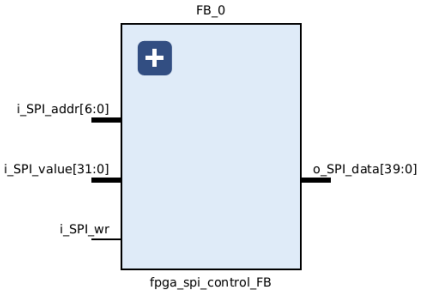

Figura 4.12: Esquemático RTL Frame Builder (FB)

4.2.5. Módulo SPI MASTER-EMISOR

Una vez construida la trama SPI para el TMC5130, se inicia el proceso de transmisión de datos por el bus SPI hacia el esclavo correspondiente. Este módulo está formado por el componente *SPI top*.

El componente *SPI top* tiene dos sub-componentes, una unidad de control y una unidad de procesamiento. Encargadas de gestionar el envío de las tramas generadas por SPI:

SPI top Unidad de Control

La Unidad de Control es una máquina de estados síncrona, en la Figura 4.13 se muestra el diagrama temporal de la secuencia de estados que describe su funcionamiento :

1. **Idle:** Estado inicial. En esta fase, el módulo permanece en espera de una señal de activación que indique el comienzo de la comunicación. Mientras se encuentra en este estado, mantiene en *High* lógico las líneas correspondientes a la interfaz SPI, indicando que no hay actividad de transmisión en curso.

2. **Pre_cs:** Antes de iniciar la transmisión, el selector de esclavo correspondiente debe activarse en estado bajo unos ciclos de reloj antes de enviar los datos. En este estado, el módulo cambia el estado de los pines a un *Low* lógico indica futura actividad en la línea.

3. **Tx:** Estado de transmisión de datos. Tanto el reloj del bus SPI como el selector del esclavo están habilitados, permitiendo el envío de datos al esclavo.

4. **Post_cs:** Este estado la transmisión aún se encuentra activa y el esclavo se encuentra procesando la información recibida. Para garantizar que lea el último dato en el último flanco de subida del reloj y deshabilitándolo unos ciclos después.

5. **Pause:** Estado de pausa entre la transmisión y recepción de datos. Al igual que en el estado **Idle** los pines de la comunicación SPI se encuentra desactivados.

6. **Rx:** Estado de recepción de datos. En este estado se habilitan los pines dedicados al *Chip Select* para permitir la recepción de datos.

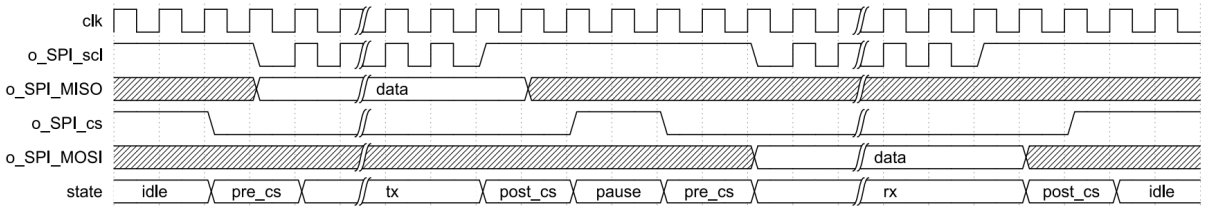

Figura 4.13: Diagrama temporal del módulo SPI TOP

SPI top Unidad de Procesamiento

La unidad de procesamiento se encarga de gestionar el envío y recepción de datos a través del bus SPI, así como de seleccionar el esclavo correspondiente para la comunicación (Figura 4.14). Sus funciones son las siguientes:

1. **Enviar datos:** A partir de la trama recibida por el módulo FB 4.2.5, transmite los datos a través de la salida o_SPI_MOSI.

2. **Recibir datos:** Captura los datos de la entrada i_SPI_MISO.

3. **Seleccionar esclavo:** Selecciona con que esclavo establece la comunicación.

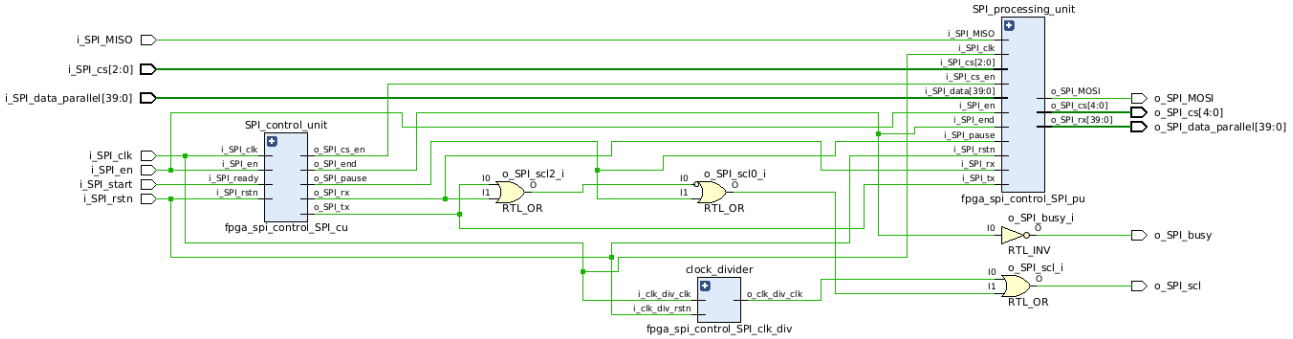

Figura 4.14: Diagrama de funcionamiento del módulo SPI TOP

4.3. Primera propuesta del Módulo de Seguimiento: Implementación en CPU

El módulo de seguimiento es el sistema que implementa el flujo de trabajo de la Figura 4.1 desde la tarea de **Captura de imágenes** hasta la tarea de **Envío del panel vía SPI**.

En esta primera propuesta, se optó por un sistema simple pero eficaz, que permitiera probar las capacidades adaptativas de los dispositivos RIS. Para el diseño del sistema, se partió de las necesidades de este: el sistema debe ser compatible con el protocolo SPI y debe poder estimar la posición espacial del objetivo. A partir de estas premisas se diseña el siguiente ecosistema (Figura 4.15):

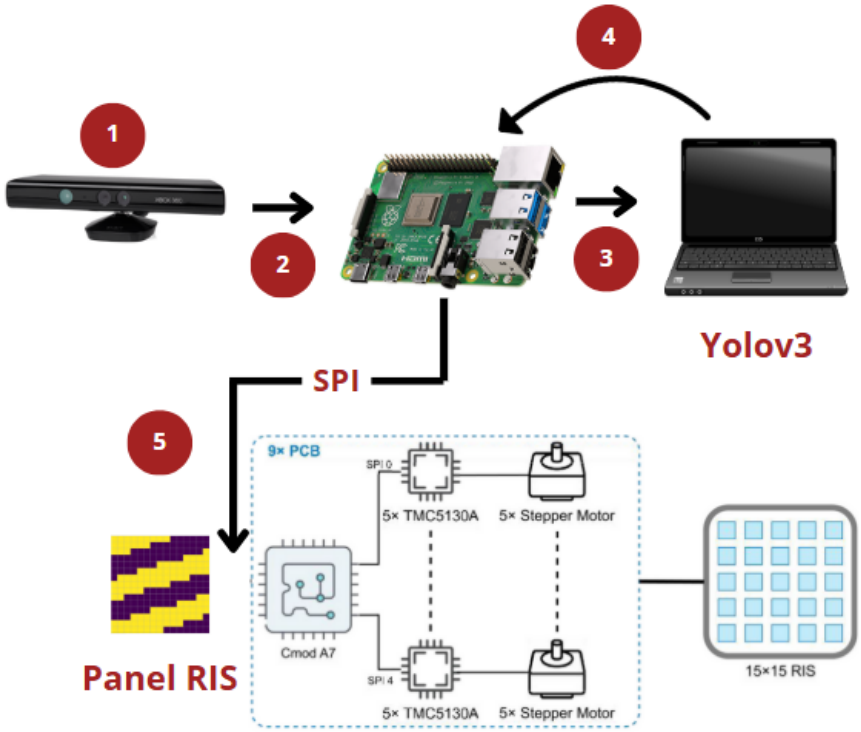

Figura 4.15: Diagrama de funcionamiento de la primera implementación

1. **Kinect V1:** El proceso comienza con la adquisición de imágenes en formato RGB, junto con información de profundidad proporcionada por el sensor.

2. **Kinect V1 a Raspberry Pi 4:** El siguiente paso es codificar la imágenes para que puedan ser

tratadas, para ello se utilizará ROS y un despliege de nodos que nos permita extraer los datos capturados por la cámara.

3. **Raspberry Pi 4 a CPU:** En este paso utilizando los tópicos de ROS, enviaremos los datos desde la Raspberry Pi hacia la CPU que los vaya a procesar. Dentro de la CPU, manteniendo el entorno de ROS, se utilizará y desarrollarán nodos que permitan procesar la imagen, realizar la estimación espacial y seleccionar el panel a configurar.

4. **CPU a Raspberry Pi 4:** Una vez seleccionado el panel a configurar,la CPU publicará en la red un tópico de ROS con los datos del panel.

5. **Raspberry Pi 4 a dispositivo RIS:** Con el panel recibido por el tópico de ROS, la Raspberry Pi transmitirá mediante el uso de sus GPIO's la información vía protocolo SPI.

4.3.1. Implementación en Raspberry Pi 4

A pesar de su tamaño compacto y ligereza, la Raspberry Pi no cuenta con la capacidad de procesamiento necesaria para ejecutar todos los procesos requeridos por los criterios de diseño. Sin embargo, su reducido tamaño la convierte en una opción adecuada para integrarse en el demostrador del sistema RIS.

En este contexto, la Raspberry Pi se utilizará principalmente para ejecutar un nodo de ROS encargado de capturar y procesar las imágenes provenientes de la cámara Kinect V1, así como para alojar una interfaz de usuario que permita controlar el sistema RIS a través de los pines GPIO. En esta sección se abordará el funcionamiento del nodo de ROS, mientras que en la sección 4.3.3 se profundizará en la interfaz de usuario.

ROS Node: *openni_ camera*

El nodo openni_camera de ROS es un controlador diseñado para cámaras compatibles con el estándar OpenNI, como la Microsoft Kinect o la ASUS Xtion Pro [30]. Este estándar abierto fue desarrollado por PrimeSense para facilitar el acceso a los datos capturados por las cámaras. Su principal ventaja es abstraer el hardware, permitiendo que los desarrolladores utilicen una única API para múltiples dispositivos.

Su función principal es capturar y publicar en ROS los distintos flujos de datos que estos sensores proporcionan, incluyendo imágenes RGB, profundidad e infrarrojo. Principalmente en el proyecto, nos centraremos en los tópicos `/camera/rgb/image_view` que contiene la imagen RGB y `/camera/depth/image_view` que contiene la información de profundidad de la escena.

4.3.2. Implementación en CPU

La CPU, al ser el dispositivo con las mejores capacidades del ecosistema, alojará los procesos dedicados a la identificación de objetos, estimación espacial y selección del panel (mediante un *dataset* predefinido).

1º ROS Node: *Darknet ROS*

Este es el primer nodo de ROS implementado dentro de la CPU, su propósito principal es permitir la detección en tiempo real de múltiples objetos dentro de una imagen o flujo de video con el modelo de detección de objetos Yolov3 [31].

Este nodo suscribe imágenes desde un tópico de ROS, como `/camera/rgb/image_view` que es el encargado de transportar las imágenes capturadas por el KinectV1, el modelo las procesa utilizando la red neuronal de Yolov3 previamente entrenada. Tras el procesamiento, el modelo produce un resultado que incluye las clases de los objetos identificados, las coordenadas de sus cajas delimitadoras (bounding boxes), y la confianza del modelo en cada detección. Estos resultados se publican en tópicos de salida como `/darknet_ros/bounding_boxes`, que contiene los datos estructurados de los objetos detectados, y `/darknet_ros/detection_image`, que muestra visualmente las cajas dibujadas sobre la imagen original.

El modelo se encuentra entrenado por un dataset que incluye 80 clases; para nuestro proyecto, únicamente será necesario identificar personas dentro de la escena. Por lo tanto, se ha realizado una alteración en el código original del nodo para que filtre sólo los resultados que hagan referencia a la clase objetivo.

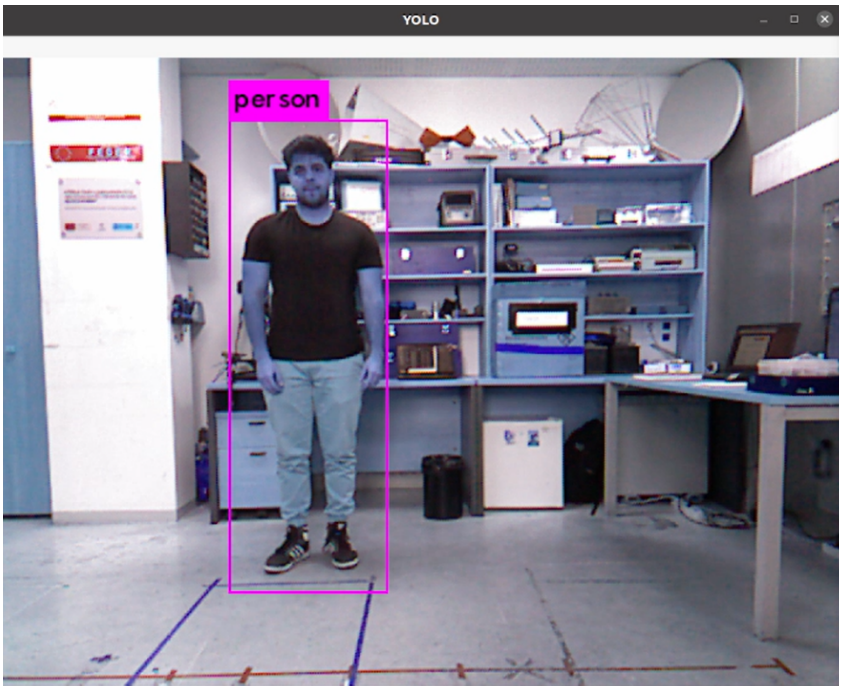

Figura 4.16: Imagen publicada por el nodo `/darknet_ros/detection_image` tras la inferencia

2º ROS Node: Position Estimator

Una vez detectado nuestro objeto objetivo, el siguiente paso es definirlo espacialmente. Por ahora sólo contamos con la siguiente información: la posición relativa en píxeles del objeto, la información de profundidad de la escena y la información óptica de la cámara previa calibración.

El primer paso de este nodo será recortar la información de profundidad de la región (*bounding box*) en la que se encuentre el objeto capturado. Al ser la región cuadrada y la forma del objetivo irregular, no nos podemos fiar de cualquier valor de profundidad que haya dentro, por ello, se realiza un histograma con los valores de profundidad de la región calculada por Yolo. La distancia del objetivo a la cámara será el valor que más frecuencia tenga en el histograma.

Una vez calculada la distancia del objetivo a la cámara, aplicaremos un **modelo de cámara pinhole**, este modelo es una representación matemática idealizada que describe cómo una cámara proyecta puntos del espacio tridimensional (3D) sobre un plano bidimensional (2D), como una imagen. Dado un punto en el espacio 3D, expresado en coordenadas de la cámara como (X, Y, Z), su proyección

en la imagen se calcula con las siguientes fórmulas:

$$u = f_x \cdot \frac{X}{Z} + c_x \tag{4.1}$$

$$v = f_y \cdot \frac{Y}{Z} + c_y \tag{4.2}$$

Aquí, (u,v) son las coordenadas del píxel en la imagen, c_x y c_y representan el centro óptico de la lente y f_x y f_y son las distancias focales entre el centro óptico de la lente y el sensor de la cámara.

Al disponer del valor de profundidad del objeto (Z) dentro de la imagen, es posible invertir el modelo para calcular la posición 3D real de un píxel de la imagen con las siguientes ecuaciones:

$$X = \frac{(u - c_x) \cdot Z}{f_x} \tag{4.3}$$

$$Y = \frac{(v - c_y) \cdot Z}{f_y} \tag{4.4}$$

$$Z = Z \tag{4.5}$$

Realizando estas transformaciones sobre el punto central del *bounding box* calculado por Yolo, somos capaces de calcular la posición 3D del objeto con una precisión de $\pm 0.2cm$. Una vez calculada su posición mediante trigonometría básica, calcularemos el ángulo de apuntamiento que debe tener el dispositivo RIS para conseguir seguir al objetivo.

3º ROS Node: Panel Selector

Este último Nodo de ROS tendrá como misión seleccionar qué panel debe ser enviado a la RIS, a partir de un dataset previamente configurado. El nodo se suscribirá al tópico que transporte los valores ángulares calculados por el nodo *Position Estimator* 4.3.2. Finalmente, publicará un tópico en la red de ROS que transporte los datos del panel en un formato comprensible por la Raspberry Pi para su posterior envío vía SPI.

4.3.3. Interfaz para usuario

La interfaz de usuario ha sido diseñada como una aplicación intuitiva que facilite el aprovechamiento completo de las capacidades del dispositivo RIS. Para su desarrollo se ha utilizado el lenguaje de programación Python. La aplicación incluye las siguientes funcionalidades:

Función 1: Load File

Esta primera función carga un panel desde un archivo .csv previamente configurado. El archivo debe ser una matriz 15x15 con números dentro del rango [0-3]. Una vez cargado el archivo, se representará el panel a configurar dentro de la interfaz, tal y como se muestra en la Figura 4.17.

Función 2: Update INDIVIDUAL Positions

Esta función sirve para reconfigurar los valores por símbolo de la RIS. Al ejecutarse, abre una ventana con una matriz 15x15 de botones que permite seleccionar individualmente el elemento a reconfigurar. Estos botones van etiquetados por dos números con la forma $[a,b]$ donde a es el número

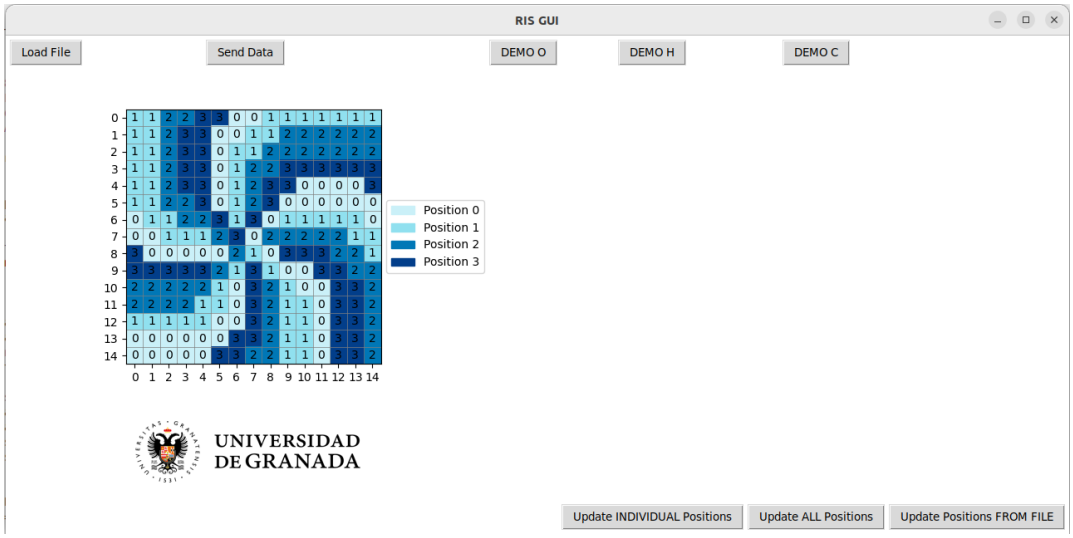

Figura 4.17: Interfaz de usuario para reprogramación de RIS

que referencia a la FPGA Cmod A7-35t a la que pertenece el elemento y b el número de elemento de [0-25]. Por otra parte, la colocación de los elementos, en la matriz de la interfaz, están fijados en la vista frontal del dispositivo RIS.

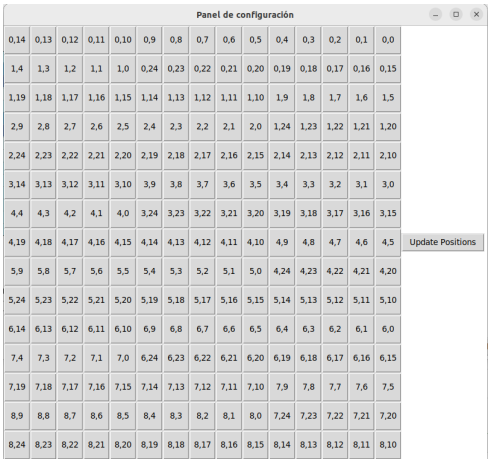

Figura 4.18: Panel 15x15 para reconfiguración individual de símbolos

Esta función busca corregir las posibles imperfecciones que pueden aparecer durante la construcción de una RIS, ofreciendo un control preciso sobre los estados de cada celda. Una vez seleccionados los elementos a configurar, escribiremos qué nuevos valores tomarán los símbolos y los enviaremos a la RIS en una sola trama de reconfiguración de símbolos 4.5.

Función 3: Update ALL Positions

Esta función configura el valor de los símbolos de todos los elementos de la RIS. Su funcionamiento es simple, al ejecutar esta función, se abre una ventana que permite escribir qué nuevos valores corresponderán a los símbolos del dispositivo. Por lo que, a pesar de que el demostrador esté desarrollado para comportarse como un sistema de 2 bits, también puede funcionar y configurarse como uno de 1 bit. Una vez seleccionados los nuevos valores, se enviará una sola trama de reconfiguración de símbolos para todos los elementos 4.5.

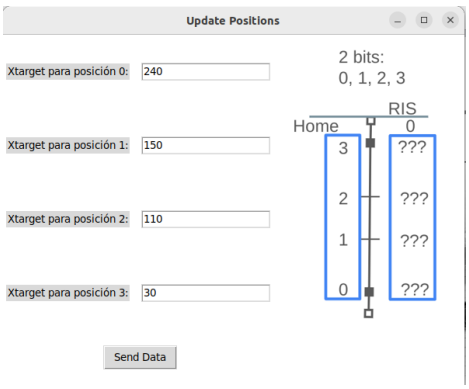

Figura 4.19: Panel para reconfiguración de todos los símbolos del dispositivo

Función 4: DEMO 'X'

Estas funciones sirven para aplicar distintas configuraciones del panel en bucle con el fin de tener una demostración técnica del funcionamiento dinámico del RIS.

Función 5: Camera Tracking

Por último, se incluye la función *Camera Tracking*. Tal y como su nombre indica, esta función habilita la reconfiguración dinámica del RIS mediante el uso de la cámara Kinect V1. La información del panel a configurar es recibida mediante un tópico de ROS que proviene de la previa selección en el nodo *Panel Selector* 4.3.2

4.4. Segunda propuesta: aceleradores hardware para IA

Para esta implementación se utilizará la tecnología de aceleradores hardware para inteligencia artificial. Para el despliegue de esta tecnología existen dos grandes proveedores, tal y como se explica en la sección 2.3. En el contexto de este proyecto se ha usado la herramienta Intel® FPGA AI Suite.

Esta herramienta tiene un flujo de trabajo concreto que involucra la necesidad de crear un sistema operativo *custom* dedicado a la actividad que queramos implementar. El flujo de trabajo para la implementación de aceleradores hardware es la siguiente:

1. **Modelo Preentrenado:** Se parte de un modelo de inteligencia artificial preentrenado en un formato compatible con frameworks cómo TensorFlow, PyTorch, Caffe, ONNX.

2. **Modelo intermedio OpenVINO:** Se transforma el modelo preentrenado en un modelo intermedio (IR Model) compatible con FPGA AI Suite.

3. **Compilación en el acelerador:** A partir de modelo IR y unos parámetros de configuración del acelerador, FPGA AI Suite selecciona que capas se implementan en la CPU del SoC FPGA y que capas del modelo se pueden implementar en la FPGA. Para conseguir una implementación óptima en un acelerador se debe poder implementar cómo mínimo el 85 % del modelo completo. La salida de este paso será un archivo `.bin` que almacenará el modelo optimizado para ejecutarse dentro del acelerador.

4. **Integración en el sistema operativo:** a partir de la herramienta *Yocto proyect* y FPGA AI Suite se creará un sistema operativo *custom* que integre el acelerador y otras librerías específicas

de OpenVINO que faciliten el acceso al acelerador en la FPGA y permitan comunicarlo con la CPU del SoC FPGA.

5. **Inferencia:** Una vez generado el sistema operativo se debe desarrollar un *middleware* específico para el modelo, que ejecute el acelerador, adapte la entrada de datos y capture la salida para el post-procesado.

6. **Post-procesado:** Tras la inferencia, se debe desarrollar un *software* que a partir de los datos recibidos tras la inferencia del modelo, los convierta en un formato legible y útil.

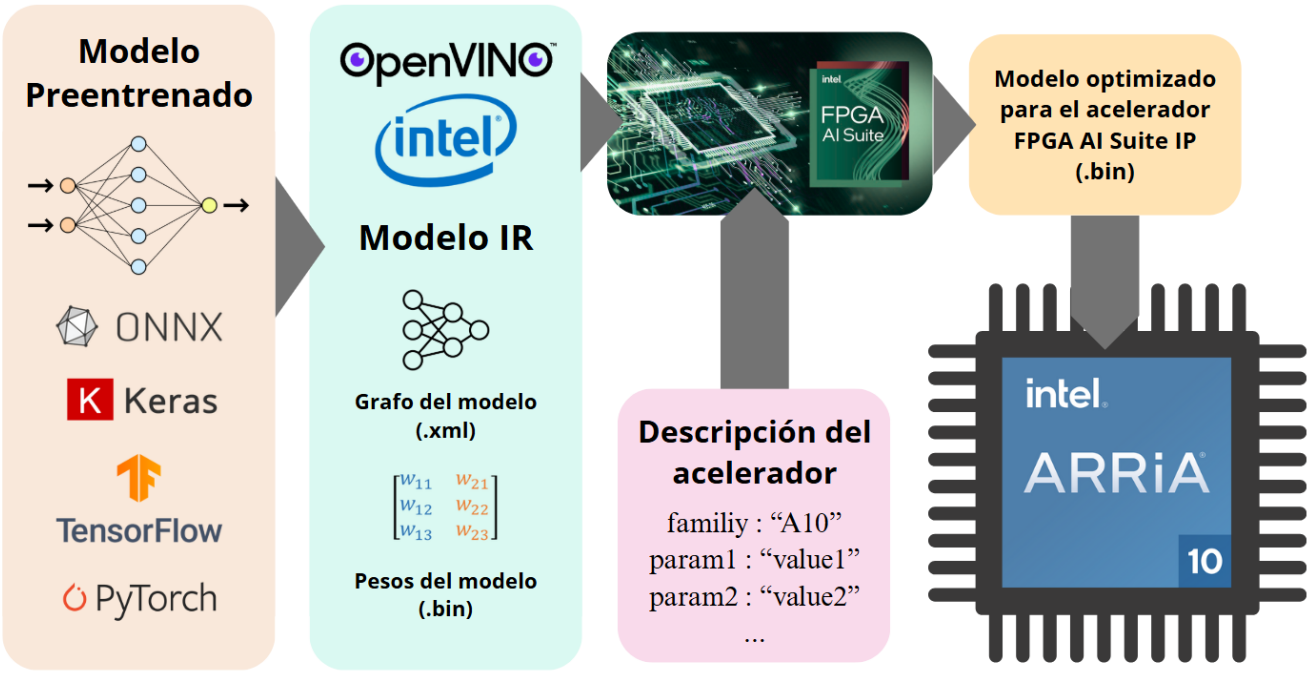

Figura 4.20: Flujo de trabajo para implementación en aceleradores hardware

4.4.1. Despliegue del sistema

Al igual que en la primera implementación (sección 4.3), el sistema debe ser compatible con el protocolo SPI y debe poder estimar la posición espacial del objetivo. Con estas premisas, se propone esta segunda implementación del sistema:

1. **Kinect V1:** Al igual que en la primera implementación, el proceso comienza con la adquisición de imágenes en formato RGB, junto con información de profundidad proporcionada por el sensor.

2. **Kinect V1 a Raspberry Pi 4:** Al igual que en la primera implementación, el siguiente paso es codificar la imágenes para que puedan ser tratadas, para ello se utilizará ROS y un despliege de nodos que nos permita extraer los datos capturados por la cámara.

3. **Raspberry Pi 4 a Arria 10:** En este paso utilizando los tópicos de ROS, enviaremos los datos desde la Raspberry Pi hacia la CPU que los vaya a procesar. Dentro de la Arria, manteniendo el entorno de ROS, se desarrollará un nodo que reciba la imagen RGB capturada por la cámara y la adapte a ala entrada del acelerador para realizar la inferencia, tras esto el nodo publicará en otro tópico las detecciones realizas por Yolo dentro del acelerador.

4. **Arria 10 a dispositivo RIS:** El siguiente paso a dar será calcular el panel a partir de la posición del objetivo en la escena, para ello se implantará un segundo acelerador con un modelo que calcule los paneles del RIS. Una vez calculado el panel se enviará mediante el protocolo SPI al dispositivo RIS.

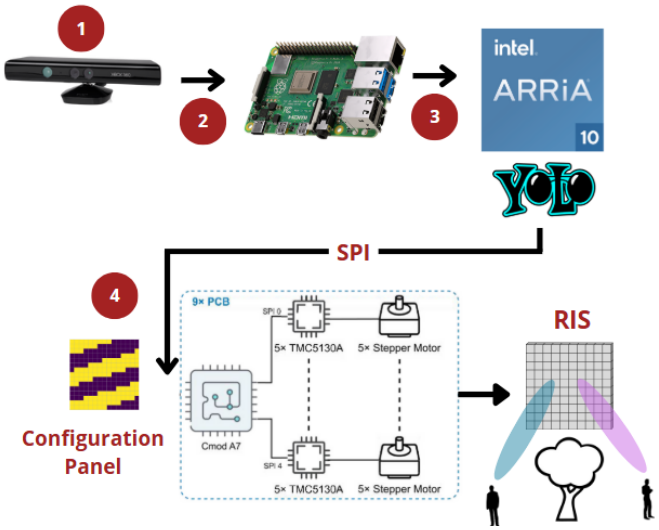

Figura 4.21: Diagrama de funcionamiento de la segunda implementación

4.4.2. Despliegue del acelerador e integración del modelo YoloV3

El primer paso en el desarrollo de esta implementación consiste en completar el flujo de integración del modelo Yolov3 dentro del acelerador hardware. Es fundamental verificar en cada etapa del proceso que las transformaciones aplicadas no alteren el comportamiento ni la precisión del modelo original. A continuación, se detallan los pasos llevados a cabo para completar correctamente dicho flujo de trabajo.

Selección del Modelo Preentrenado

OpenVINO y en consecuencia FPGA AI Suite, son herramientas sensibles a la topología de la red que van a tratar, esto implica que no todos los tipos de capas están soportados por la herramienta. Estas limitaciones dependen de la versión FPGA AI Suite que se utilice para la integración en el acelerador, en la implementación en este proyecto se ha utilizado concretamente la versión 2023.2. Las capas soportadas por esta versión están reflejadas en el manual Intel® FGPA AI Suite: IP Reference Manual [32].

Tal y como se explicó en el apartado 2.2.3, para realizar la estimación del tamaño del *bounding box*, YoloV3 requiere el uso de funciones de activación tipo *sigmoide*. Aunque el modelo original no incluye explícitamente estas capas, como se puede observar en la Tabla 3.2, algunas versiones modificadas y publicadas por la comunidad sí las integran. Esta incorporación tiene como objetivo simplificar el procesamiento de las salidas, especialmente para facilitar su uso por parte de usuarios con menor experiencia.

Esto dificulta la elección del modelo a integrar, ya que limita a ensayo y error qué modelo será compatible con Intel® FGPA AI Suite. Para solucionar este problema se optó por utilizar el modelo preentrenado de YoloV3 que proporciona el repositorio de OpenVINO: open_model_zoo [33]. Este modelo de YoloV3 es fiel a la arquitectura del original, añadiendo únicamente 3 capas extra al final de la red para dividir la salida en 3 cauces distintos que permitan diferenciar a qué formato de grid pertenece la salida (Tabla 3.3).

El modelo rescatado está preentrenado con 80 clases a partir del dataset público COCO (Common Objects in Context). Dentro de las clases que provee este dataset se encuentra la clase 'person' ideal para la detección del objetivo al que se desea aplicar el seguimiento mediante la RIS. Por lo que en este paso únicamente descargaremos el modelo preentrenado para el framework TensorFlow.

Modelo intermedio OpenVINO

Una vez decidido el modelo a transformar, el siguiente paso es convertirlo a un modelo intermedio (IR) compatible con OpenVINO. Este modelo consta de dos partes: el grafo del modelo (.xml) que contiene las distintas capas que componen el modelo y los valores asociados a los pesos de cada capa (.bin). Esta transformación se realizará mediante la función `omz_converter`:

```
omz_converter --name yolo-v3-tf \
    --download_dir $COREDLA_WORK/demo/models/ \
    --output_dir $COREDLA_WORK/demo/models/
```

Los parámetros a incluir en esta función son los siguientes:

- **–name:** es el nombre del modelo que queramos convertir, el nombre debe ser el del archivo que lo almacena.
- **–download_dir:** es el directorio donde se encuentre el modelo preentrenado que queramos convertir a modelo intermedio (IR).
- **–output_dir:** es el directorio de salida donde se guardará el nuevo modelo transformado.

Para finalizar, se debe comprobar que la salida del modelo IR concuerde con la salida del modelo preentrenado. El *script* de test para este modelo se adjunta en el Anexo A .

Este script realiza el postprocesado en los datos de salida del modelo Yolov3, tal y como se describe en la sección 3.8 devolviendo la imagen con la predicción realizada (Figura 4.22). A partir de la imagen generada por el test y las coordenadas de la predicción, decidiremos si la conversión ha sido correcta.

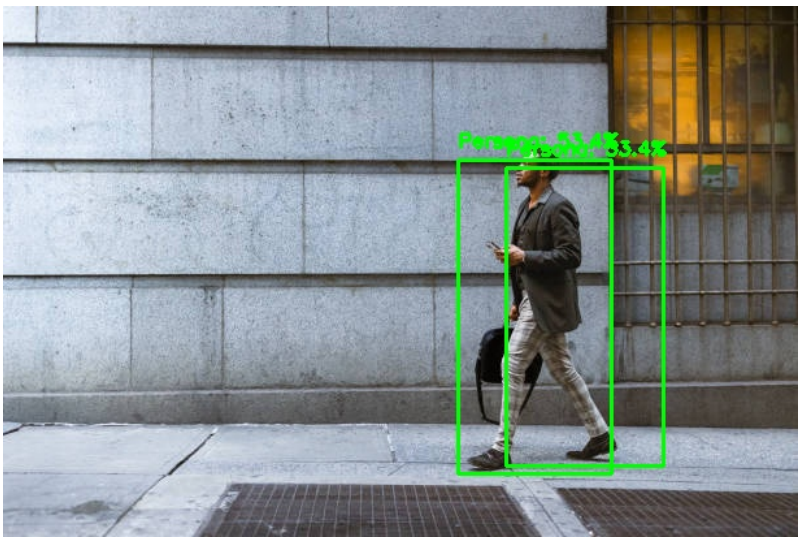

Figura 4.22: Predicción realizada con el modelo IR de Yolov3

Compilación en el acelerador

El siguiente paso consiste en compilar el modelo considerando los parámetros específicos del acelerador. La elección del acelerador es un factor determinante, ya que afectará directamente tanto al rendimiento (velocidad de operación) de la red como al área de silicio requerida para su implementación. Intel® FGPA AI Suite proporciona varios aceleradores predefinidos, en concreto el acelerador usado en este proyecto es el acelerador `A10_FP16_Performance.arch`, que utiliza representaciones en formato 16 bits 2.3.

Este acelerador representa la opción más potente después de su versión en 32 bits. La elección de este modelo se justifica en que, aunque implica una ligera pérdida de precisión en las predicciones, su uso permite minimizar significativamente el tiempo de procesamiento, lo cual es prioritario en el contexto del sistema diseñado. Al final, la selección de acelerador es un compromiso entre velocidad, área y precisión.

Con el acelerador seleccionado, el siguiente paso será compilar el modelo para adaptarlo a los parámetros del acelerador. Para ello, se utilizará la función:

```
dla_compiler --march A10_FP16_Performance.arch --network-file yolo-v3-tf.xml --foutput-format=open_vino_hetero
    --o yolov3_model.bin --batch-size=1 --fanalyze-performance
```

Los parámetros a incluir en esta función son los siguientes:

- –**march:** Especifica el archivo de arquitectura del acelerador.
- –**network-file:** Indica el archivo de descripción de red en formato OpenVINO IR (.xml), correspondiente al modelo Yolov3.
- –**foutput-format:** Define el formato de salida del modelo compilado. open_vino_hetero indica que el modelo será preparado para ejecución heterogénea combinando CPU, GPU.
- –**o:** Ruta de salida donde se almacenará el modelo compilado (.bin) listo para su ejecución en el acelerador.
- –**batch-size:** Especifica el tamaño de la entrada indicando que el procesará una sola imagen a la vez.
- –**fanalyze-performance:** Activa el análisis de rendimiento durante la compilación.

Si la compilación es exitosa y las capas del modelo son compatibles con FPGA AI Suite, la compilación devolverá el modelo compilado (.bin) y un análisis de rendimiento del modelo en el acelerador. El análisis completo se adjunta en el Anexo B, pero antes de finalizar este punto comentar dos puntos de gran importancia en este análisis:

```
The input graph is split into 4 subgraph(s), CPU:3 FPGA:1.
```

Esto implica que no todo el modelo ha sido implantado en la FPGA, sino que hay 3 sub-grafos que se alojan en el CPU. Para el modelo de Yolov3 los tres sub-grafos corresponden a las salidas que diferencian los formatos del grid usados para la detección. En concreto, las capas alojadas en la CPU son las siguientes:

Por último, el parámetro:

```
FINAL THROUGHPUT = 13.4897 fps
```

Layer	Tamaño del grid (Shape)
conv2d_58/BiasAdd	13 × 13
conv2d_66/BiasAdd/YoloRegion	26 × 26
conv2d_74/BiasAdd/YoloRegion	52 × 52

Tabla 4.1: Capas alojadas en la CPU tras compilación

Indica cuál es la estimación para la velocidad de operación del modelo en el acelerador. Estos datos no son definitivos ya que pueden variar entre un 30-50 % pero sí nos dan una idea de la capacidad que va a tener el sistema.

Integración en el sistema operativo

En la sección 3.4 se mencionó que el dispositivo Arria 10 es un SoC FPGA, lo que significa que integra, dentro de un mismo encapsulado, una FPGA junto con un microprocesador. Este microprocesador posee la capacidad suficiente para alojar un sistema operativo, lo que permite ejecutar software de alto nivel directamente en el propio dispositivo.

Para la construcción del sistema operativo se ha utilizado *Yocto proyect* que genera imágenes de Linux personalizadas con el fin de optimizar las capacidades del microprocesador. En concreto, para el proyecto se utilizó la versión Yocto Project 5.1, también conocida como Styhead.

Esta versión es compatible con ROS Noetic esencial en el ecosistema del proyecto para la comunicación con Raspberry Pi 4. Yocto funciona a partir de recetas predefinidas que facilitan la integración del *kernel* y las aplicaciones dentro del sistema operativo.

A modo de resumen, la construcción del sistema operativo mediante Yocto depende de los archivos `local.conf` y `bblayers.conf`. El archivo `local.conf` recoge los paquetes que se van a instalar dentro de la imagen, el *kernel* y *bootloader* a implementar. Por otra parte, `bblayers.conf` guarda las rutas de las recetas de todo el software y middleware a usar.

Como el archivo `bblayers.conf` no es de interés para mostrar la configuración de la imagen, se muestra únicamente la configuración realizada en el archivo `local.conf`:

```
CONF_VERSION = "2"
MACHINE = "arria10"
```

- **CONF_VERSION:** Es la versión del formato del archivo de configuración.
- **MACHINE:** Define la plataforma hardware de destino, lo que indica que se compilarán paquetes y una imagen compatible con la FPGA Arria 10.

El conjunto de herramientas introducidas en el sistema operativo:

```
IMAGE_INSTALL:append = " nano"
IMAGE_INSTALL:append = " python3"
IMAGE_INSTALL:append = " libftdi"
IMAGE_INSTALL:append = " libusb1"
IMAGE_INSTALL:append = " libxi"
IMAGE_INSTALL:append = " libxmu"
IMAGE_INSTALL:append = " freeglut"
IMAGE_INSTALL:append = " coreutils"
IMAGE_INSTALL:append = " python3-posix-ipc"
IMAGE_INSTALL:append = " python3-numpy"
IMAGE_INSTALL:append = " opencv"
```

- **nano:** Editor de texto
- **coreutils:** Comandos básicos para el GNU.
- **python3, python3-numpy, python3-posix-ipc:** Soporte completo de Python y bibliotecas científicas y de IPC.
- **libftdi, libusb1:** Para comunicación USB.
- **libxi, libxmu, freeglut:** Utilizadas en aplicaciones gráficas con OpenGL.
- **opencv:** Biblioteca para visión por computador.

Por último, la integración de ROS:

```
ROS_DISTRO = "noetic"
EXTRA_IMAGE_FEATURES ?= "ros-implicit-workspace"
IMAGE_INSTALL:append = " ros-core "
```

- **ROS_DISTRO:** Especifica la distribución de ROS que se usará, en este caso ROS Noetic.
- **EXTRA_IMAGE_FEATURES:** Incluye características adicionales en la imagen, como soporte para entornos ROS.
- **ros-core:** Añade los componentes esenciales de ROS.

Middleware y software

Por último, se debe desarrollar un código compatible con las herramientas de FPGA AI Suite que ejecute el acelerador en el SoC FPGA y permita extraer la salida del modelo para el post-procesado.

El *middleware* desarrollado para la ejecución del acelerador se basa en una versión modificada del script de ejemplo proporcionado por FPGA AI Suite, disponible en el repositorio [34]. Este script permite ejecutar el acelerador y mantenerlo activo indefinidamente dentro de la FPGA.

Para gestionar la salida del acelerador, el programa accede a un archivo denominado `shared.mem`, el cual representa un área de memoria compartida entre el procesador y la FPGA. Este archivo mapea una región específica de la memoria del SoC FPGA con un tamaño equivalente al de la salida del acelerador, permitiendo así un acceso rápido y simplificado por parte del microcontrolador.

Por otra parte, el *software* se encargará de inyectar imágenes al acelerador, extraer los resultados de la predicción del archivo `shared.mem` y realizar el post-procesado al igual que se hace en el script de testeo del Anexo A.

Capítulo 5

Evaluación y resultados

Tras el diseño y desarrollo de los sistemas expuestos en el Capítulo 4, en este capítulo se presentan los resultados obtenidos a partir de la implementación y evaluación de cada uno de los sistemas que conforman el proyecto. Se incluyen tanto métricas cuantitativas, que permiten valorar el rendimiento y la eficiencia de los módulos desarrollados, como un análisis cualitativo que refleja el grado de integración, usabilidad y robustez del sistema en condiciones reales de operación.

5.1. Evaluación del módulo de comunicación SPI

Para la evaluación del módulo de comunicación SPI se ha empleado la técnica de *testbench*, ampliamente utilizada en el ámbito del diseño digital con lenguajes de descripción hardware como VHDL. Esta metodología consiste en generar un entorno de simulación que permite verificar el comportamiento funcional del componente bajo prueba sin necesidad de hardware físico. En particular, se inyectan señales de entrada a través de los terminales del módulo, emulando las condiciones de operación reales a las que se enfrentará durante su integración en el sistema completo.

El testbench permite observar cómo responde el módulo ante distintos patrones de estímulo, detectar posibles errores lógicos y validar la sincronización de las señales. Gracias a esta técnica, es posible depurar y ajustar el diseño de forma iterativa, reduciendo así el riesgo de fallos en fases posteriores de implementación física o despliegue sobre FPGA.

Para nuestro caso, la simulación se realizará sobre el componente *top*, el que unifica y conecta todos los componentes descritos en la sección 4.2. El código completo *testbench* supera las 2000 líneas, lo que imposibilita que se adjunte completamente, por ello se compartirán trozos de este que aporten a la explicación del entorno de simulación.

Por otra parte, se evaluará tanto la latencia introducida por la lógica implementada como el coste en área ocupado dentro de la FPGA. Con este propósito, en la Tabla 3.1 se presentan los recursos disponibles del dispositivo Cmod A7-35t, lo cual permite contextualizar la eficiencia de la implementación desarrollada.

5.1.1. Declaración de señales y terminales de conexión

Esta compone la primera parte del *testbench* en ella se declaran los terminales del componente y los buses internos (*signals* en VHDL) que transportarán la salida simulada y los estímulos de entrada (Anexo C). Para diferenciar los *signals* de los terminales del componente, se alterará el nombre de los terminales eliminando el prefijo `o_` e `i_` que los diferenciaba como salida (out) y entrada (in)

respectivamente, heredando el nombre de los terminales que se les asigne.

```
i_MOSI => MOSI, -- i_MOSI (terminal real), MOSI (signal simulada)
```

5.1.2. Simulación de la señal de reloj

En esta sección del *testbench* se lleva a cabo la generación de las señales de reloj necesarias para la simulación (Anexo D). En particular, se crean dos señales diferenciadas: una correspondiente al reloj interno del componente bajo prueba y otra asociada al protocolo de comunicación SPI.

La señal de reloj interna tiene un período de 10 ns (100 MHz), mientras que la señal de reloj usada para la comunicación SPI tiene un período de 100 ns (10 MHz). La diferencia de frecuencia entre las dos señales se debe a que el hardware implementado requiere una frecuencia mayor a la de operación del protocolo SPI para gestionar internamente las operaciones lógicas y de control que aseguren la comunicación. En la simulación, esta diferencia es 10:1 que es una proporción límite, ya que en contextos reales, la diferencia es de mayor orden.

5.1.3. Simulación de envío de trama

Para simular el envío de una trama mediante el protocolo SPI, se inyectará un array de valores binarios a través del terminal MOSI del dispositivo (Anexo E). La transmisión comienza cuando el testbench pone en estado lógico bajo la señal CS (Chip Select), activando así el esclavo SPI. A partir de ese momento, y sincronizado con los flancos del reloj SPI, se introducirá un bit por ciclo a través del terminal MOSI hasta completar la trama completa de 54 bits. En total se envían 2 tramas:

- **Trama 1:** `0xC0 3E3F FFFF FFFF`, esta trama pertenece al tipo reconfiguración de símbolo. En concreto, esta instrucción configura la memoria de los motores menos significativos para todos los símbolos con el valor `0x01 FF`, (equivalente a '0' en pasos).
- **Trama 2:** `0xA4 0FAA AAAA AAAA`, esta trama pertenece al tipo reconfiguración de panel. En ella se incluye la configuración de un paquete de 25 motores divididos de 5 e 5. La trama envía una configuración distinta para el paquete de los 5 motores más significativos (`0b0000001111`) y un paquete común para los 20 menos significativos (`0b1010101010`).

  Esta prueba se realiza para comprobar que la memoria ha sido grabada correctamente, ya que el valor en pasos de los motores menos significativos de cada paquete de 5 tendrá un valor igual para distintos símbolos. Además, se puede comprobar que el sistema transmite los pasos a configurar.

5.1.4. Resultados

En consonancia con lo presentado en las secciones anteriores, la idea de este test es comprobar cómo el módulo de comunicación SPI responde a distintos estímulos usando el conjunto de tramas para la configuración del dispositivo RIS.

Primer resultado, respuesta al estímulo

Este primer resultado tiene como objetivo mostrar de forma cualitativa que el sistema responde correctamente al estímulo generado por una trama de configuración enviada al panel. En la Figura 5.1 se observa cómo el *testbench* inyecta una secuencia binaria, que simula el envío de una trama de

reconfiguración del panel, a través del terminal de entrada `MOSI`, lo que da inicio a una operación de comunicación mediante el protocolo SPI.

Una vez activada la señal de selección del esclavo (`cs_xx`), individual para cada esclavo, y sincronizada la transmisión con la señal de reloj SPI, la trama comienza a propagarse hacia los drivers TMC5130.

Como registro cuantitativo, se observa que el módulo ha requerido únicamente 16 ciclos de reloj para procesar la trama de configuración e iniciar la comunicación SPI. Este resultado demuestra un procesamiento prácticamente instantáneo desde el momento en que la trama es recibida por el sistema hasta que se genera la señal de activación del bus SPI. Esta baja latencia pone de manifiesto la eficiencia de la lógica implementada y valida el uso de este hardware para tareas de control en tiempo real.

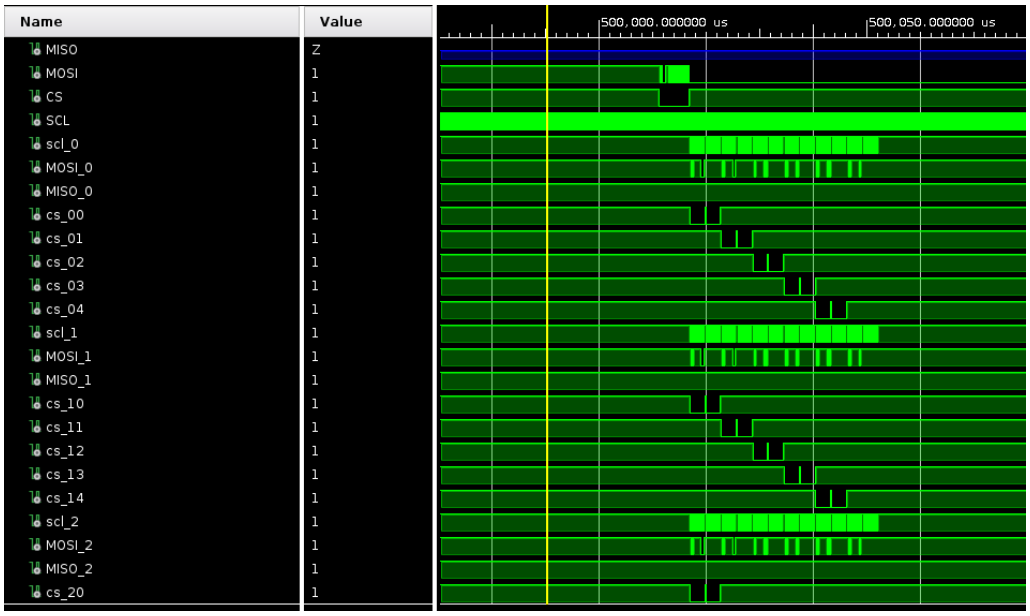

Figura 5.1: Estimulo del sistema ante la entrada de una trama de configuración del panel

Segundo resultado, lectura de la memoria tras la escritura

Con este resultado se busca validar que los valores de los símbolos han cambiado para los esclavos a los que se les ha reescrito la memoria. En este caso, los esclavos afectados son los que se sincronizan con el terminal `cs_x[0 ó 1]`. Se observa que a pesar de recibir distintos símbolos de configuración, al tener escrito en su memoria el mismo valor de pasos para todos los símbolos, la secuencia binaria es la misma (Figura 5.2).

Resumen y coste de recursos

Finalizando ese primer paquete de resultados dedicado al rendimiento del módulo de comunicación SPI obtenemos la siguiente tabla de resultados:

Parámetro	**Valor**
LUTs	1.628
Flip-Flops	1.475
Latencia recepción-envío	16 ciclos de reloj

Tabla 5.1: Recursos utilizados y latencias registradas para el módulo Cmod A7-35t

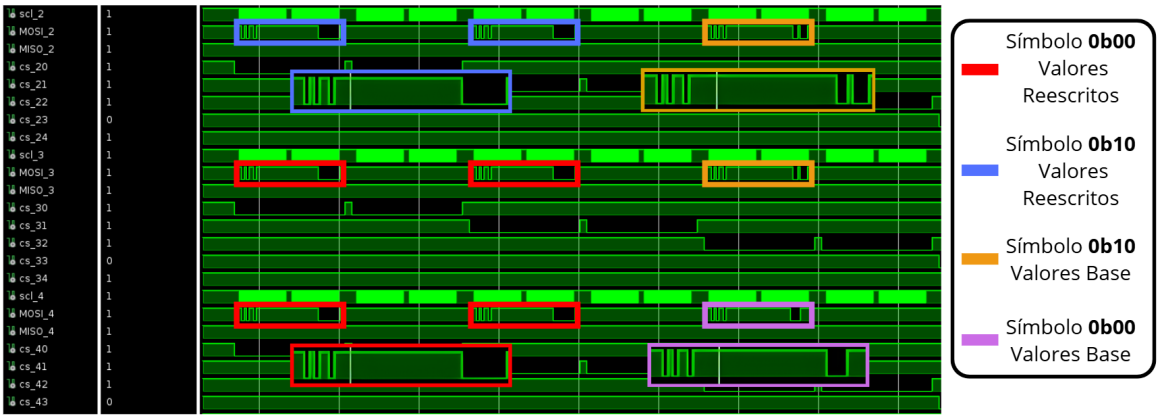

Figura 5.2: Salida del sistema para estados con el símbolo pero distinto valor almacenado en la memoria

En la Tabla 5.1 se observa que el módulo ha utilizado un 7.82 % de las LUTs disponibles y un 3.4 % de flip-flops. Además, presenta una latencia de recepción y envío en la comunicación SPI de 16 ciclos de reloj a una frecuencia de 12 MHz, lo que equivale a una latencia total aproximada de 1.3 μs. Estos resultados demuestran la efectividad del módulo diseñado, consiguiendo respuestas casi instantáneas en la reconfiguración de dispositivos RIS. Además, el bajo consumo de recursos lógicos disponibles optimiza el diseño y deja margen suficiente para futuras ampliaciones o integraciones adicionales en la FPGA.

5.2. Comparativa de latencia en la implementación de Yolov3

En esta sección se presenta un análisis comparativo de la latencia obtenida al ejecutar el modelo Yolov3 en una CPU y dentro de un acelerador hardware.

Para ello, se comparará el tiempo que la red tarda en procesar un único fotograma en cada plataforma evaluada. Además, al igual que se hizo en la sección 5.1.4, se analizará el uso de recursos lógicos empleados durante el despliegue del acelerador, con el fin de valorar su eficiencia en términos de área ocupada y latencia obtenida. En las tablas 5.2 y 5.3 se especifican los recursos de los sistemas donde se implementa Yolov3.

Parámetro	**Valor**
ALMs (Adaptive Logic Modules)	251.680
Flip-Flops	1.006.720
M20K Block	42.620

Tabla 5.2: Recursos disponibles en FPGA Intel Arria 10 SX 660 [24]

Recurso	**Especificación**
Modelo de CPU	Intel i7-11370H
Núcleos	4
Hilos por núcleo	1
Memoria RAM	8 GB
Frecuencia base	3,3 GHz
Sistema operativo	Ubuntu 20.04 LTS

Tabla 5.3: Recursos disponibles en la CPU

5.2.1. Resultados implementación en CPU

Para calcular la frecuencia de procesamiento de la red Yolov3 en la CPU, se ha utilizado la herramienta interna de ROS `rostopic hz`. Esta herramienta calcula la frecuencia media de publicación (en Hz) de un tópico en ROS, en otras palabras, calcula cuántos fotogramas por segundo (FPS) está procesando la red de Yolov3.

La función se ejecuta mediante el siguiente comando:

```
rostopic hz /nombre_del_topic
```

Donde `rostopic hz` es la función a utilizar y `/nombre_del_topic` es el nombre del tópico que queramos estudiar. En nuestro caso, el tópico a medir es `/darknet_ros/detection_image` que publica los resultados de la inferencia del modelo.

Los resultados se reflejan en la Tabla 5.4:

Recurso	**Valor medido**	**% de Uso**
Tasa media	1,545 Hz	
Tasa mínima	1,982 Hz	
Tasa máxima	1,357 Hz	
Núcleos en uso	3	75 %

Tabla 5.4: Rendimiento en la CPU

5.2.2. Resultados implementación en aceleradores hardware

Para calcular la frecuencia de procesamiento de la red Yolov3 en el acelerador hardware, se ha implementado una función dentro del software de post-procesado que contabiliza cuántas inferencias ha logrado completar el acelerador en el intervalo de un segundo. Una vez medido, guarda el dato en un vector y calcula dinámicamente cada segundo la tasa media, máxima y mínima. Los resultados se reflejan en la tabla 5.5:

Recurso	**Valor medido**	**% de Uso**
Tasa media	5,65 Hz	
Frecuencia Máxima	5,94 Hz	
Frecuencia Mínima	5,48 Hz	
ALMs usados	85.995	34,16 %
Flip-Flops usados	229.398	22.78 %
M20K Block usados	1.461	3,42 %

Tabla 5.5: Rendimiento y coste en Arria 10

5.2.3. Resumen y valoración de resultados

Tras lo descrito anteriormente, podemos concluir que el sistema que mejor optimiza el modelo de Yolov3 es el que usa aceleradores hardware mejorando la velocidad de inferencia un 366 % al modelo implementado en una CPU.

Por otra parte, comparando los recursos necesarios para ejecutar el modelo en los sistemas, la CPU utiliza el 75 % de todos sus recursos, imposibilitando incluir otro modelo de Yolov3 en el mismo sistema.

En el caso del sistema basado en el acelerador hardware, se ha utilizado menos del 34 % de todos los recursos lógicos disponibles, lo que permitiría, si la aplicación lo requiere, ejecutar hasta 3 modelos

de Yolov3 independientes para 3 puntos de visión distintos.

5.3. Integración en demostrador

En esta sección se presentan los resultados obtenidos tras la integración del módulo de comunicación SPI con el módulo de seguimiento. Para validar esta integración, se ha empleado el demostrador RIS de 2 bits con reconfiguración mecánica, desarrollado conjuntamente por el grupo de investigación SWAT-UGR y el grupo DiTEC-UGR.

Cabe señalar que la integración se ha realizado exclusivamente con la primera versión del módulo de seguimiento, ya que la segunda propuesta se encuentra actualmente en fase de validación y despliegue.

5.3.1. Ensamblaje del sistema

El demostrador RIS de 2 bits con reconfiguración mecánica consta de una superficie reconfigurable compuesta por una matriz de 15×15 elementos, los cuales utilizan motores para llevar a cabo su reconfiguración. Además, el sistema está conformado por 9 placas independientes, cada una de las cuales integra un módulo Cmod A7-35t (Breadboardable Artix-7 FPGA) y 25 drivers TMC5130, encargados de controlar bloques de 25 motores por placa (Figura 5.3).

La comunicación externa de cada placa se lleva a cabo mediante el protocolo SPI, utilizando un bus impreso que interconecta los módulos Cmod A7-35t a través de pines de 2.54 mm. En estos pines se conecta el sistema configurador, el cual se encuentra aislado eléctricamente mediante un level shifter, encargado de garantizar un nivel de voltaje constante de 3.3V en los pines del módulo Cmod A7-35t, independientemente del nivel lógico del sistema configurador.

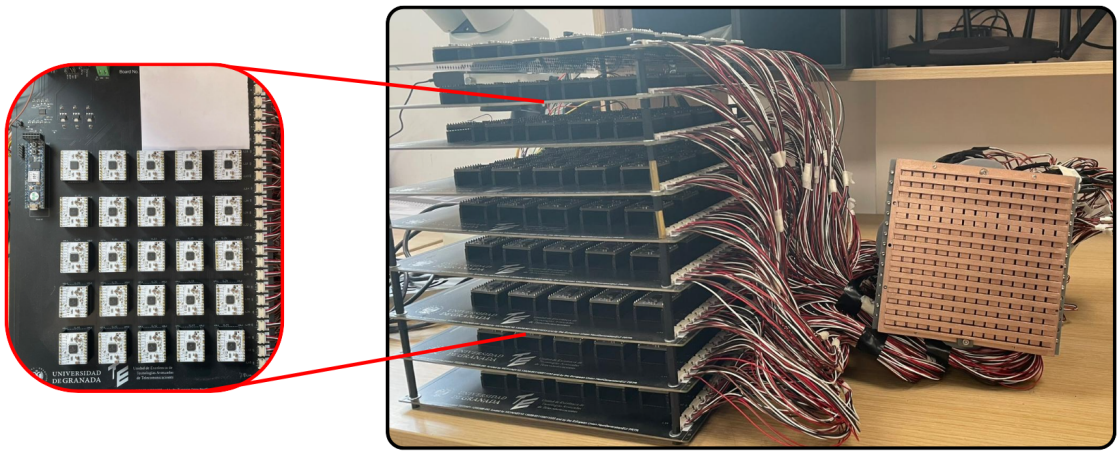

Figura 5.3: Demostrador RIS de 2 bits con reconfiguración mecánica

Finalmente, se conectará una Raspberry Pi 4 a los pines de 2.54 mm destinados a la comunicación SPI. Además, se asignará un pin específico para el canal Chip Select (CS), siguiendo la asignación de la Figura 5.4.

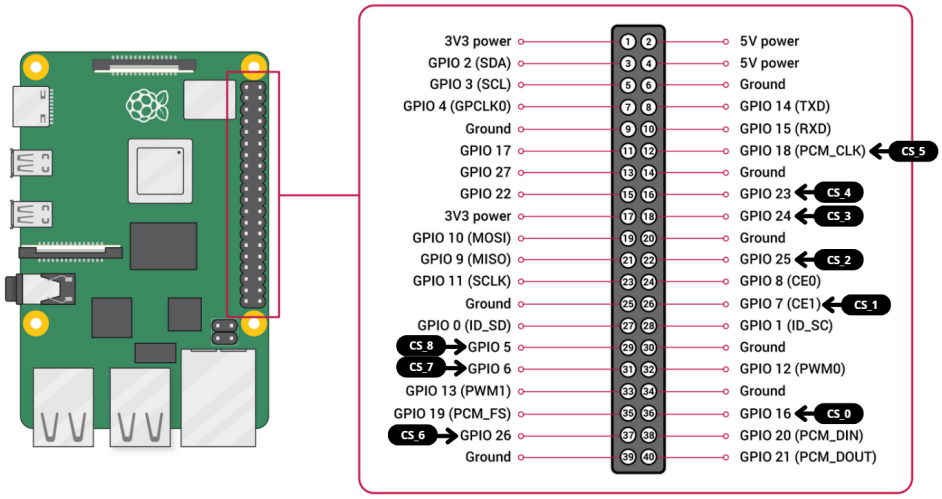

Figura 5.4: Asignación del canal Chip Select en la Raspberry Pi 4

5.3.2. Configuración del módulo de seguimiento

Como primer paso a realizar, configuraremos la red de ROS en los dispositivos que conformen la red. Para una mejor optimización de la red, se recomienda que el MASTER de la red sea el dispositivo que capture los datos (en nuestro caso, la Raspberry Pi 4). El archivo para realizar la configuración será: `/home/{user}/.bashrc`

```
export ROS_MASTER_URI = http://<IP_DEL_MASTER >:11311
export ROS_HOSTNAME = <IP_LOCAL>
```

Una vez establecida la jerarquía de la red de ROS, se procede a iniciar el nodo encargado de gestionar la obtención de las imágenes desde el Kinect V1.

```
roslaunch openni_launch openni.launch
```

Para comprobar si el nodo se ha iniciado correctamente y está capturando imagen, desde el segundo dispositivo ejecutaremos el siguiente comando que permitirá ver el flujo de datos que viaja por el tópico:

```
rostopic echo /camera/rgb/image_raw
```

Una vez comprobado el funcionamiento de la red y del primer nodo, el siguiente paso es iniciar el nodo que ejecuta Yolo3 en la CPU:

```
roslaunch darknet_ros darknet_ros.launch
```

Este nodo, además de ejecutar la red Yolov3, abrirá una ventana en la que se podrán visualizar en tiempo real los resultados del procesamiento realizado por la red (tal y como muestra la Figura 5.5). El siguiente nodo a iniciar es el nodo para el cálculo de la posición espacial del objetivo. Para ello, se ejecutará el comando:

```
rosrun position_estimator position_estimator.py
```

Cómo punto final se iniciará la interfaz de usuario de Python en la Raspberry Pi 4 y se procederá a iniciar la función de reconfiguración del panel a partir del seguimiento del objetivo.

5.3.3. Presentación del funcionamiento del demostrador

Una vez montado e integrado el sistema, se puede verificar que los distintos módulos que conforman el proyecto han sido integrados exitosamente, permitiendo la reconfiguración dinámica del demostrador RIS de 2 bits con reconfiguración mecánica. A continuación, se presentan una serie de figuras que evidencian el cumplimiento del objetivo propuesto en este trabajo fin de máster.

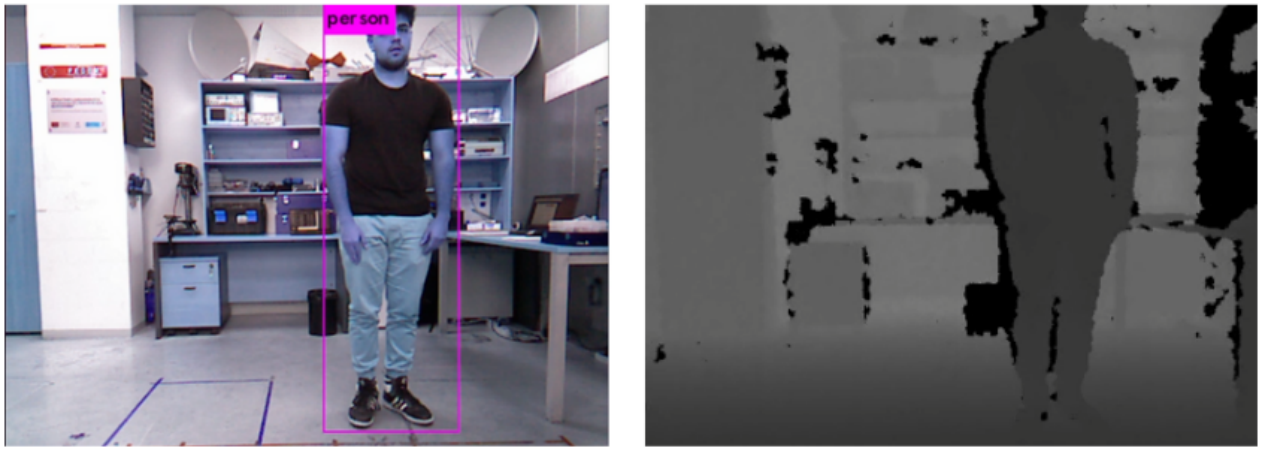

Figura 5.5: a) Salida del nodo que integra Yolov3 b) Imagen infrarroja capturada por el Kinect V1

En la Figura 5.5 se presenta la salida del modelo de Yolov3 con la imagen infrarroja capturada por el Kinect V1. Mostrando el correcto funcionamiento del módulo de seguimiento.

Finalmente se presenta cómo varía el panel configurado en la RIS para distintos ángulos de apuntamiento, que evidencia el funcionamiento conjunto del módulo de seguimiento y del módulo de comunicación SPI desarrollado en este trabajo fin de máster (Figura 5.6).

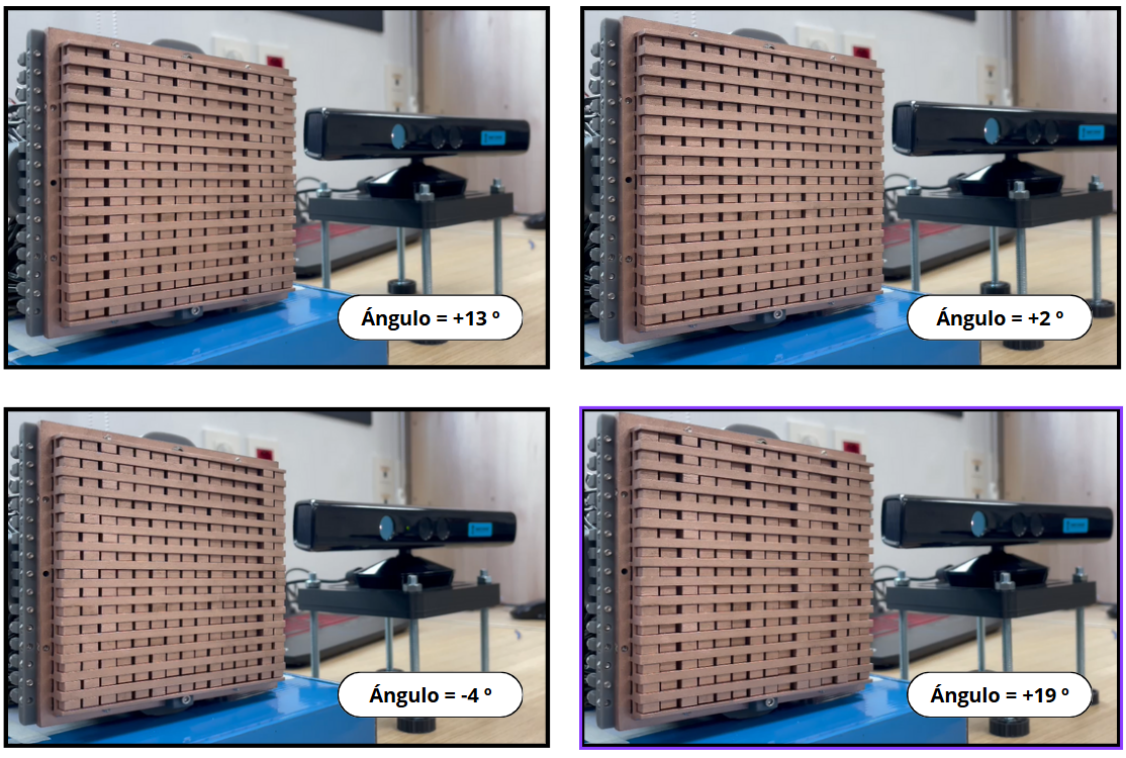

Figura 5.6: Configuración del panel para distintos ángulos de apuntamiento

Capítulo 6

Conclusiones

En este último capítulo se presentará una reflexión sobre los resultados obtenidos en el proyecto, se evaluará el grado de cumplimiento de los objetivos planteados y, finalmente, se propondrán posibles líneas de investigación futuras relacionadas con esta iniciativa.

6.1. Conclusión

Este Trabajo Fin de Máster presenta la primera integración de hardware para superficies inteligentes mecánicas reconfigurables. A pesar de que el hardware integrado está adaptado para la configuración de una RIS 15x15 de 225 elementos, el despliegue realizado en la sección 4.2 es altamente modular y puede ser adaptado para controlar más elementos e incluso aumentar la codificación de estados para conseguir un dispositivo RIS de 3 bits.

Por otro lado, el módulo de seguimiento se basa en el trabajo desarrollado por la Universidad de Xi'an (sección 2.1.1); sin embargo, a diferencia de dicho enfoque, este proyecto introduce por primera vez el uso de aceleradores hardware para las tareas de seguimiento y reconfiguración en este tipo de dispositivos RIS. En este sentido, al haberse desarrollado dos implementaciones del módulo de seguimiento, ha sido posible demostrar las ventajas que ofrecen los aceleradores hardware frente a las soluciones tradicionales basadas en CPU, mejorando significativamente la velocidad de inferencia en modelos complejos, como los utilizados en la detección de objetos a partir de imágenes.

Esta mejora en el rendimiento no solo permite una reconfiguración más rápida de las superficies RIS en función del entorno, sino que también habilita nuevos casos de uso en los que la latencia y la adaptabilidad son factores críticos, como la comunicación en contextos de transporte inteligente o en redes densas de dispositivos IoT. Además, el uso de aceleradores hardware sienta las bases para la integración de modelos de aprendizaje profundo más complejos directamente en el sistema embebido, lo que representa un paso importante hacia la autonomía e inteligencia de futuras superficies reconfigurables.

En definitiva, este Trabajo Fin de Máster no solo demuestra la viabilidad técnica de controlar superficies inteligentes reconfigurables mediante hardware dedicado, sino que también establece una base sólida para futuras investigaciones orientadas a la escalabilidad y eficiencia de modelos de inteligencia artificial que mejoren las capacidades de los dispositivos RIS. La arquitectura propuesta y los resultados obtenidos confirman el potencial de esta línea de desarrollo para convertirse en un componente clave de las redes de comunicación del futuro, alineándose con los requisitos emergentes de la próxima generación de sistemas 6G.

6.2. Cumplimento de objetivos

A continuación, se detalla el cumplimiento de los objetivos propuestos en este trabajo fin de máster. Para ello, se analizan de forma individual las tareas que estructuran el desarrollo del proyecto, verificando su implementación, validación y alineación con los resultados obtenidos.

TA1 **Diseño de hardware:** Este objetivo se ha cumplido satisfactoriamente mediante el desarrollo de un hardware que permite la reconfiguración de estados de una superficie inteligente mecánica reconfigurable compuesta por 225 elementos, utilizando un protocolo estándar de comunicación hardware como es el SPI.

TA2 **Diseño de un protocolo:** Este objetivo se ha cumplido satisfactoriamente desarrollando un protocolo de tramas eficiente que permite configurar de forma eficiente bloques de 25 elementos en paquetes de 7 bytes. Por otra parte, el protocolo añade tramas específicas dedicadas a la configuración de estados ofreciendo la capacidad de cambiar tanto la frecuencia de reflexión de la RIS mecánica cómo de aumenta la codificación de estados.

TA3 **Diseño hardware de una memoria volátil:** Que almacena los valores valores equivalentes en pasos de los estados y modular para el aumento de número de elementos y estados a representar.

TA4 **Simulación y evaluación** La evaluación del hardware se ha llevado a cabo en entornos simulados, así como mediante su implementación en un dispositivo RIS real.

TA5 **Implementación de un sistema de seguimiento activo en CPU:** Se ha desarrollado un sistema inspirado en el implementado por la Universidad de Xi'an (sección 2.1.1), para seguimiento activo con RIS.

TA6 **Implementación de un sistema de seguimiento activo en aceleradores hardware:** Se ha realizado un sistema de seguimiento activo basado en la tecnología de aceleradores hardware que mejora las latencias de sistema presentado por la Universidad de Xi'an.

TA7-8 **Integración y Evaluación:** Se ha integrado el primer sistema de seguimiento con el hardware desarrollado en las tareas TA1–TA3 en una superficie inteligente mecánica reconfigurable real, demostrando su funcionamiento correcto en un entorno físico y validando la viabilidad de la solución propuesta para aplicaciones en tiempo real.

6.3. Líneas de investigación

El presente proyecto abre diversas posibilidades de mejora y ampliación, tanto en el campo de visión por computador como en el ámbito de redes convolucionales. A continuación, se presentan algunas líneas de investigación que podrían abordarse en futuros trabajos:

- **Estimación espacial a partir de varios sensores** Durante la realización de este proyecto sólo hemos realizado la estimación de posición a partir de un único sensor, esto limita al dispositivo RIS a un apuntamiento restringido a la región que registra el sensor. Por lo cuál incluir más dispositivos que capturen imagen aumentará la región de apuntamiento útil.
- **Uso de RF-Imaging:** Esta línea de investigación explora el uso de imágenes generadas a partir de señales de radiofrecuencia (RF-Imaging) para reconstruir en tiempo real el entorno electromagnético que rodea a una superficie inteligente reconfigurable (RIS). El objetivo es utilizar esta información como entrada para algoritmos de reconfiguración adaptativa, permitiendo que la RIS ajuste su comportamiento de forma autónoma ante cambios dinámicos en el entorno, como movimiento de objetos, interferencias o presencia de obstáculos.

- **Integración hardware de modelos CNN:** Esta línea de investigación busca cambiar la tecnología de aceleradores hardware por redes optimizadas que implementen capas convolucionales directamente en hardware. El enfoque busca maximizar la eficiencia en términos de latencia, consumo energético y área ocupada, facilitando así la integración de modelos de visión artificial en sistemas embebidos con recursos limitados.

Anexo A

Test para el modelo IR de Yolov3

```python
from openvino.inference_engine import IECore
import cv2
import numpy as np
import time


def intersection_over_union(box_1, box_2):


    width_of_overlap_area = min(box_1['xmax'], box_2['xmax']) - max(box_1['xmin'], box_2['xmin'])
    height_of_overlap_area = min(box_1['ymax'], box_2['ymax']) - max(box_1['ymin'], box_2['ymin'])

    if width_of_overlap_area < 0 or height_of_overlap_area < 0:
        area_of_overlap = 0
    else:
        area_of_overlap = width_of_overlap_area * height_of_overlap_area

    box_1_area = (box_1['ymax'] - box_1['ymin']) * (box_1['xmax'] - box_1['xmin'])
    box_2_area = (box_2['ymax'] - box_2['ymin']) * (box_2['xmax'] - box_2['xmin'])
    area_of_union = box_1_area + box_2_area - area_of_overlap

    return area_of_overlap / area_of_union if area_of_union != 0 else 0

def apply_nms(detections, iou_threshold=0.5, prob_threshold=0.5):
    """Aplica (NMS)"""
    # Filtra por confianza minima
    detections = [obj for obj in detections if obj['confidence'] >= prob_threshold]

    # Ordena por confianza (de mayor a menor)
    detections = sorted(detections, key=lambda x: x['confidence'], reverse=True)

    # Aplica NMS
    for i in range(len(detections)):
        if detections[i]['confidence'] == 0:
            continue
        for j in range(i + 1, len(detections)):
            if detections[i]['class_id'] != detections[j]['class_id']:
                continue
            if intersection_over_union(detections[i], detections[j]) > iou_threshold:
                detections[j]['confidence'] = 0  # Suprime la caja menos confiable

    # Filtra las cajas suprimidas
    return [obj for obj in detections if obj['confidence'] > 0]



# Inicializa el motor de inferencia
ie = IECore()

# Carga la red
net = ie.read_network(model="yolo-v3-tf.xml", weights="yolo-v3-tf.bin")

# Carga la red a la CPU
exec_net = ie.load_network(network=net, device_name="CPU")
```

```
55
56
57  # Obtener el nombre del blob de entrada
58  input_blob = next(iter(net.input_info))
59
60  # Leer y preprocesar la imagen
61  image = cv2.imread('test_9.jpg')
62  original_h, original_w = image.shape[:2]
63  image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
64  image_resized = cv2.resize(image_rgb, (416, 416))
65
66
67  # Input con forma (1, 416, 416, 3)
68  input_data_nhwc = np.expand_dims(image_resized, axis=0)
69
70  # Convertir a NCHW para pasar al modelo
71  input_data_nchw = input_data_nhwc.transpose((0, 3, 1, 2))
72
73  # Ejecutar inferencia
74  output = exec_net.infer(inputs={input_blob: input_data_nchw})
75
76  anchors_map = {
77      13:  np.array([[116, 90],
78                     [156, 198],
79                     [373, 326]]),
80
81      26: np.array([[30, 61],
82                    [62, 45],
83                    [59, 119]]),
84
85       52: np.array([[10, 13],
86                     [16, 30],
87                    [33, 23]])
88  }
89
90
91  def sigmoid(x):
92      return 1 / (1 + np.exp(-x))
93
94
95  detections = []
96  for output_name, output_data in output.items():
97      _, _, h, w = output_data.shape
98      anchors = anchors_map.get(h)
99      if anchors is None:
100         continue
101
102     output_reshaped = output_data.reshape(1, 3, 85, h, w).transpose(0, 1, 3, 4, 2)
103
104     for i in range(h):
105         for j in range(w):
106             for k in range(3):
107                 pred = output_reshaped[0, k, i, j]
108                 tx, ty, tw, th = pred[0:4]
109                 obj_conf = sigmoid(pred[4])
110                 class_probs = sigmoid(pred[5:])
111                 person_conf = class_probs[0]
112                 final_conf = obj_conf * person_conf
113
114                 if final_conf > 0.45:
115                     bx = (sigmoid(tx) + j) * (416 / w)
116                     by = (sigmoid(ty) + i) * (416 / h)
117                     bw = anchors[k][0] * np.exp(tw)
118                     bh = anchors[k][1] * np.exp(th)
119
120                     x_min = int((bx - bw / 2) * original_w / 416)
121                     y_min = int((by - bh / 2) * original_h / 416)
122                     x_max = int((bx + bw / 2) * original_w / 416)
123                     y_max = int((by + bh / 2) * original_h / 416)
124
125                     detections.append({
126                         'xmin': x_min,
127                         'ymin': y_min,
128                         'xmax': x_max,
129                         'ymax': y_max,
130                         'confidence': final_conf,
```

```
                        'class_id': 0  # Asume que solo detectamos personas (clase 0)
                    })

# Aplica NMS
final_detections = apply_nms(detections, iou_threshold=0.5, prob_threshold=0.5)

# Dibuja las cajas finales
for idx, obj in enumerate(final_detections, 1):
    x_min, y_min, x_max, y_max = obj['xmin'], obj['ymin'], obj['xmax'], obj['ymax']
    confidence = obj['confidence']
    print (f"El objeto {idx} con una confianza del {confidence:.3f}% tiene las siguientes coordenadas:")
    print(f"\n Coordenadas: x_min={x_min}, y_min={y_min}, x_max={x_max}, y_max={y_max}\n")
    color = (0, 255, 0)  # Verde
    thickness = 2
    cv2.rectangle(image, (x_min, y_min), (x_max, y_max), color, thickness)
    label = f"Persona: {confidence * 100:.1f}%"
    cv2.putText(image, label, (x_min, y_min - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, thickness)

# Guarda la imagen
cv2.imwrite("out_test.jpg", image)
print(f"Guardado en out_test.jpg")

```

Anexo B

Análisis de rendimiento del acelerador A10_FP16_Performance.arch

```
###############################################################################
# CONVERTING MODEL TO FPGA AI SUITE COMPILED GRAPH
###############################################################################


network-weightings not specified. Auto assigning network weights to 1.0
Model path set to /home/angel/Documentos/prueba_openvino/yolov3_openvino_model/yolo-v3-tf.xml
read_graph for yolo-v3-tf complete.
[ INFO ] The input graph is split into 4 subgraph(s), CPU:3 FPGA:1.
[ WARNING ] Input graph is split into too many subgraphs, this can be caused by unsupported architecture layers
      - check yolo-v3-tf_dla_messages.txt. Splitting the graph into many partitions can lead to running out of
     FPGA DDR memory.
Starting compilation
Finished compilation in 5999 ms
Exporting input transform to file
Exporting output transform to file
Executing performance estimate
Estimating performance
----------------------------------------------------------------
yolo-v3-tf_0 reported throughput: 13.489707
TOTAL DDR SPACE REQUIRED = 138.704590 MB
      DDR CONFIG BUFFER SIZE = 0.213867 MB
      DDR FILTER BUFFER SIZE = 118.273438 MB
      DDR INTERMEDIATE BUFFER SIZE = 7.921875 MB
TOTAL DDR TRANSFERS REQUIRED = 227.178040 MB
      DDR FILTER READS REQUIRED   = 147.017090 MB
      DDR FEATURE READS REQUIRED  = 49.497375 MB
      DDR FEATURE WRITES REQUIRED = 30.449707 MB
NUMBER OF DDR FEATURE READS = 88.000000
MINIMUM AVERAGE DDR BANDWIDTH REQUIRED = 3064.565081 MB/s

----------------------------------------------------------------
Performance Estimator Throughput Breakdown
Arch: kvec64xcvec32_i4x1_fp16_sb31744_xbark32_actk32_poolk4
Number of DLA instances                          = 1
Number of DDR Banks per DLA instance             = 1
CoreDLA Target Fmax                              = 265 MHz
PE Target Fmax                                   = 265 MHz
Batch Size                                       = 1
PE-only Conv Throughput No DDR                   = 14 fps
PE-only Conv Throughput                          = 14 fps
Overall Throughput Inf PE Buf Depth (zero MPBW)  = 14 fps
Overall Throughput Zero PE Buf Depth (zero MPBW) = 14 fps
Overall Throughput Inf PE Buf Depth              = 14 fps
Overall Throughput Zero PE Buf Depth             = 13 fps
----------------------------------------------------------------
FINAL THROUGHPUT = 13.4897 fps
FINAL THROUGHPUT PER FMAX (CoreDLA) = 0.0509046 fps/MHz
FINAL THROUGHPUT PER FMAX (PE)      = 0.0509046 fps/MHz
Performance Estimator Execution Time: 4 ms
```

```
Executing area estimate
Estimated area:
ALMs: 61588
ALUTs: 75762
Registers: 220506
DSPs: 1114
M20Ks: 1461
Memory ALMs: 2314
```

Anexo C

Declaración de señales y terminales de conexión

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   -- Uncomment the following library declaration if using
6   -- arithmetic functions with Signed or Unsigned values
7   --use IEEE.NUMERIC_STD.ALL;
8
9   -- Uncomment the following library declaration if instantiating
10  -- any Xilinx leaf cells in this code.
11  --library UNISIM;
12  --use UNISIM.VComponents.all;
13
14  entity fpga_spi_control_top_tb is
15  --  Port ( );
16  end fpga_spi_control_top_tb;
17
18  architecture Behavioral of fpga_spi_control_top_tb is
19
20  component fpga_spi_control_top is
21    Port ( i_en_top            : in  STD_LOGIC;
22           i_rstn          : in  STD_LOGIC;
23           i_clk           : in  STD_LOGIC;
24
25           i_CS            : in  STD_LOGIC;
26           i_SCL           : in  STD_LOGIC;
27           i_MOSI          : in  STD_LOGIC;
28           o_MISO          : out STD_LOGIC;
29
30           o_scl_0         : out STD_LOGIC;
31           o_MOSI_0        : out STD_LOGIC;
32           i_MISO_0        : in  STD_LOGIC;
33           o_cs_00         : out STD_LOGIC;
34           o_cs_01         : out STD_LOGIC;
35           o_cs_02         : out STD_LOGIC;
36           o_cs_03         : out STD_LOGIC;
37           o_cs_04         : out STD_LOGIC;
38
39           o_scl_1         : out STD_LOGIC;
40           o_MOSI_1        : out STD_LOGIC;
41           i_MISO_1        : in  STD_LOGIC;
42           o_cs_10         : out STD_LOGIC;
43           o_cs_11         : out STD_LOGIC;
44           o_cs_12         : out STD_LOGIC;
45           o_cs_13         : out STD_LOGIC;
46           o_cs_14         : out STD_LOGIC;
47
48           o_scl_2         : out STD_LOGIC;
49           o_MOSI_2        : out STD_LOGIC;
50           i_MISO_2        : in  STD_LOGIC;
51           o_cs_20         : out STD_LOGIC;
52           o_cs_21         : out STD_LOGIC;
```

```
53          o_cs_22         : out STD_LOGIC;
54          o_cs_23         : out STD_LOGIC;
55          o_cs_24         : out STD_LOGIC;
56
57          o_scl_3         : out STD_LOGIC;
58          o_MOSI_3        : out STD_LOGIC;
59          i_MISO_3        : in  STD_LOGIC;
60          o_cs_30         : out STD_LOGIC;
61          o_cs_31         : out STD_LOGIC;
62          o_cs_32         : out STD_LOGIC;
63          o_cs_33         : out STD_LOGIC;
64          o_cs_34         : out STD_LOGIC;
65
66          --out_flag_top : out std_logic;
67          --o_frame_complete :  out std_logic_vector (9 downto 0);
68
69          o_scl_4         : out STD_LOGIC;
70          o_MOSI_4        : out STD_LOGIC;
71          i_MISO_4        : in  STD_LOGIC;
72          o_cs_40         : out STD_LOGIC;
73          o_cs_41         : out STD_LOGIC;
74          o_cs_42         : out STD_LOGIC;
75          o_cs_43         : out STD_LOGIC;
76          o_cs_44         : out STD_LOGIC;
77
78          o_wip           : out STD_LOGIC
79
80          );
81    end component fpga_spi_control_top;
82
83
84  -- Inputs
85  signal en     : STD_LOGIC := '0';
86  signal rstn   : STD_LOGIC := '1';
87  signal clk    : STD_LOGIC := '0';
88  -- Input SPI
89  signal MISO   : STD_LOGIC := '1';
90  signal MOSI   : STD_LOGIC := '1';
91  signal CS     : STD_LOGIC := '1';
92  signal SCL    : STD_LOGIC := '1';
93  -- Outputs 0
94  signal scl_0  : STD_LOGIC := '1';
95  signal MOSI_0 : STD_LOGIC := '1';
96  signal MISO_0 : STD_LOGIC := '1';
97  signal cs_00  : STD_LOGIC := '1';
98  signal cs_01  : STD_LOGIC := '1';
99  signal cs_02  : STD_LOGIC := '1';
100 signal cs_03  : STD_LOGIC := '1';
101 signal cs_04  : STD_LOGIC := '1';
102
103 -- Outputs 1
104 signal scl_1  : STD_LOGIC := '1';
105 signal MOSI_1 : STD_LOGIC := '1';
106 signal MISO_1 : STD_LOGIC := '1';
107 signal cs_10  : STD_LOGIC := '1';
108 signal cs_11  : STD_LOGIC := '1';
109 signal cs_12  : STD_LOGIC := '1';
110 signal cs_13  : STD_LOGIC := '1';
111 signal cs_14  : STD_LOGIC := '1';
112
113 -- Outputs 2
114 signal scl_2  : STD_LOGIC := '1';
115 signal MOSI_2 : STD_LOGIC := '1';
116 signal MISO_2 : STD_LOGIC := '1';
117 signal cs_20  : STD_LOGIC := '1';
118 signal cs_21  : STD_LOGIC := '1';
119 signal cs_22  : STD_LOGIC := '1';
120 signal cs_23  : STD_LOGIC := '1';
121 signal cs_24  : STD_LOGIC := '1';
122
123 -- Outputs 3
124 signal scl_3  : STD_LOGIC := '1';
125 signal MOSI_3 : STD_LOGIC := '1';
126 signal MISO_3 : STD_LOGIC := '1';
127 signal cs_30  : STD_LOGIC := '1';
128 signal cs_31  : STD_LOGIC := '1';
```

```
129 signal cs_32  : STD_LOGIC := '1';
130 signal cs_33  : STD_LOGIC := '1';
131 signal cs_34  : STD_LOGIC := '1';
132
133 -- Outputs 4
134 signal scl_4  : STD_LOGIC := '1';
135 signal MOSI_4 : STD_LOGIC := '1';
136 signal MISO_4 : STD_LOGIC := '1';
137 signal cs_40  : STD_LOGIC := '1';
138 signal cs_41  : STD_LOGIC := '1';
139 signal cs_42  : STD_LOGIC := '1';
140 signal cs_43  : STD_LOGIC := '1';
141 signal cs_44  : STD_LOGIC := '1';
142 signal out_flag_tb  : STD_LOGIC := '1';
143 signal o_frame_complete_tb : std_logic_vector (9 downto 0);
144
145
146 begin
147
148 -- Instantiate the Unit Under Test (UUT)
149 top : fpga_spi_control_top
150   Port map ( i_en_top            => en,
151              i_rstn          => rstn,
152              i_clk           => clk,
153
154              i_CS            => CS,
155              i_SCL           => SCL,
156              i_MOSI          => MOSI,
157              o_MISO          => MISO,
158
159              o_scl_0         => scl_0,
160              o_MOSI_0        => MOSI_0,
161              i_MISO_0        => MISO_0,
162              o_cs_00         => cs_00,
163              o_cs_01         => cs_01,
164              o_cs_02         => cs_02,
165              o_cs_03         => cs_03,
166              o_cs_04         => cs_04,
167
168              o_scl_1         => scl_1,
169              o_MOSI_1        => MOSI_1,
170              i_MISO_1        => MISO_1,
171              o_cs_10         => cs_10,
172              o_cs_11         => cs_11,
173              o_cs_12         => cs_12,
174              o_cs_13         => cs_13,
175              o_cs_14         => cs_14,
176
177              o_scl_2         => scl_2,
178              o_MOSI_2        => MOSI_2,
179              i_MISO_2        => MISO_2,
180              o_cs_20         => cs_20,
181              o_cs_21         => cs_21,
182              o_cs_22         => cs_22,
183              o_cs_23         => cs_23,
184              o_cs_24         => cs_24,
185
186              o_scl_3         => scl_3,
187              o_MOSI_3        => MOSI_3,
188              i_MISO_3        => MISO_3,
189              o_cs_30         => cs_30,
190              o_cs_31         => cs_31,
191              o_cs_32         => cs_32,
192              o_cs_33         => cs_33,
193              o_cs_34         => cs_34,
194
195              o_scl_4         => scl_4,
196              o_MOSI_4        => MOSI_4,
197              i_MISO_4        => MISO_4,
198              o_cs_40         => cs_40,
199              o_cs_41         => cs_41,
200              o_cs_42         => cs_42,
201              o_cs_43         => cs_43,
202              o_cs_44         => cs_44,
203
204              --o_frame_complete => o_frame_complete_tb,
```

```
205             --out_flag_top => out_flag_tb,
206
207             o_wip          => wip
208            );
209
```

Anexo D

Simulación de la señal de reloj

```
-- Output
signal wip    : STD_LOGIC := '0';

-- Clock period definitions
constant clk_period : time := 10 ns;

constant spi_clk_period : time := 100 ns;
constant internal_clk_period : time := 10 ns;


-- Clock process definitions
process
  begin
      clk <= '1';
      wait for internal_clk_period / 2;
      clk <= '0';
      wait for internal_clk_period / 2;
  end process;

  -- Clock generation for SPI clock
  process
  begin
      SCL <= '1';
      wait for spi_clk_period / 2;
      SCL <= '0';
      wait for spi_clk_period / 2;
  end process;

-- Stimulus process
stim_proc: process
begin
wait for 20 ns;
rstn <= '0';
wait for 1000000 ns;

wait;
end process;
```

Anexo E

Ejemplo de envío de trama simulada

```
------------------------------------------- Header
   MOSI <= '1';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '1';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '1';
   wait for spi_clk_period;

   ------------------------------------ 50 - 54

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '0';
   wait for spi_clk_period;

   MOSI <= '1';
   wait for spi_clk_period;

   MOSI <= '1';
   wait for spi_clk_period;

   MOSI <= '1';
   wait for spi_clk_period;

   MOSI <= '1';
   wait for spi_clk_period;

   ---------------------------------------- 40 - 50

   MOSI <= '1';
```

```
55      wait for spi_clk_period;
56
57      MOSI <= '0';
58      wait for spi_clk_period;
59
60      MOSI <= '1';
61      wait for spi_clk_period;
62
63      MOSI <= '0';
64      wait for spi_clk_period;
65
66      MOSI <= '1';
67      wait for spi_clk_period;
68
69      MOSI <= '0';
70      wait for spi_clk_period;
71
72      MOSI <= '1';
73      wait for spi_clk_period;
74
75      MOSI <= '0';
76      wait for spi_clk_period;
77
78      MOSI <= '1';
79      wait for spi_clk_period;
80
81      MOSI <= '0';
82      wait for spi_clk_period;
83      ------------------------------------------------- 30 - 40
84
85      MOSI <= '1';
86      wait for spi_clk_period;
87
88      MOSI <= '0';
89      wait for spi_clk_period;
90
91      MOSI <= '1';
92      wait for spi_clk_period;
93
94      MOSI <= '0';
95      wait for spi_clk_period;
96
97      MOSI <= '1';
98      wait for spi_clk_period;
99
100     MOSI <= '0';
101     wait for spi_clk_period;
102
103     MOSI <= '1';
104     wait for spi_clk_period;
105
106     MOSI <= '0';
107     wait for spi_clk_period;
108
109     MOSI <= '1';
110     wait for spi_clk_period;
111
112     MOSI <= '0';
113     wait for spi_clk_period;
114
115     ------------------------------------------- 20 - 30
116
117     MOSI <= '1';
118     wait for spi_clk_period;
119
120     MOSI <= '0';
121     wait for spi_clk_period;
122
123     MOSI <= '1';
124     wait for spi_clk_period;
125
126     MOSI <= '0';
127     wait for spi_clk_period;
128
129     MOSI <= '1';
130     wait for spi_clk_period;
```

```
131
132     MOSI <= '0';
133     wait for spi_clk_period;
134
135     MOSI <= '1';
136     wait for spi_clk_period;
137
138     MOSI <= '0';
139     wait for spi_clk_period;
140
141     MOSI <= '1';
142     wait for spi_clk_period;
143
144     MOSI <= '0';
145     wait for spi_clk_period;
146 ------------------------------------------------ 10 - 20
147     MOSI <= '1';
148     wait for spi_clk_period;
149
150     MOSI <= '0';
151     wait for spi_clk_period;
152
153     MOSI <= '1';
154     wait for spi_clk_period;
155
156     MOSI <= '0';
157     wait for spi_clk_period;
158
159     MOSI <= '1';
160     wait for spi_clk_period;
161
162     MOSI <= '0';
163     wait for spi_clk_period;
164
165     MOSI <= '1';
166     wait for spi_clk_period;
167
168     MOSI <= '0';
169     wait for spi_clk_period;
170
171     MOSI <= '1';
172     wait for spi_clk_period;
173
174     MOSI <= '0';
175     wait for spi_clk_period;
176
177     ---------------------------------- 0 - 10
178     -- Deactivate chip select after frame transmission
179     CS <= '1';
180
181     wait for 50 ms;
```

Bibliografía

[1] CNMC, El tráfico de datos móviles creció más de un 40 % el año pasado, Disponible en: https://blog.cnmc.es/2022/08/30/el-trafico-de-datos-moviles-crecio-mas-de-un-40-el-ano-pasado/. –Consultado el 11 julio de 2025.

[2] H. Berry, R. G. Malech, and W. A. Kennedy, *The Reflectarray Antenna*, IEEE Transactions on Antennas and Propagation, vol. 11, no. 6, pp. 645–651, Nov. 1963.

[3] M. Baena-Molina, Á. Palomares-Caballero, G. Martínez-García, J. E. Galeote-Cazorla, A. Ramírez-Arroyo, and J. F. Valenzuela-Valdés, *1-bit Mechanically Reconfigurable Metasurface as a Beam Splitter for Indoor Environments at 28 GHz*, IEEE Antennas and Wireless Propagation Letters.

[4] M. Ouyang, Y. Wang, F. Gao, S. Zhang, P. Li, and J. Ren, *Computer Vision-Aided Reconfigurable Intelligent Surface-Based Beam Tracking: Prototyping and Experimental Results*, IEEE Antennas and Wireless Propagation Letters.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 1999, pp. 1150–1157.

[6] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, vol. 1, pp. 886–893.

[7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *arXiv preprint arXiv:1312.6229*, 2013.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. ISBN: 9780262035613.

[9] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[10] J. R. Quinlan,*Induction of decision trees, Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[11] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, vol. 65, no. 6, pp. 386–408, 1958.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition, Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, M. P. Lungren, and A. Y. Ng, *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*, arXiv:1711.05225, 2017.

[14] F. Schroff, D. Kalenichenko, and J. Philbin, *FaceNet: A Unified Embedding for Face Recognition and Clustering*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, arXiv:1503.03832, 2015.

[15] A. Menon, B. Omman, and A. S., *Pedestrian Counting Using YOLOv3*, in *2021 International Conference on Innovative Trends in Information Technology (ICITIIT)*, pp. 1–9, 2021.

[16] C. Liu, S.-C. Hu, C. Wang, K. Lafata, and F.-F. Yin, *Automatic detection of pulmonary nodules on CT images with YOLOv3: Development and evaluation using simulated and patient data*, *Physics in Medicine and Biology*, vol. 65, no. 23, 235018, 2020. PMID: 33014725. PMCID: PMC7495314.

[17] IEEE Standards Association, *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019.

[18] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, *High-performance FPGA-based CNN accelerator with block-floating-point arithmetic*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[19] Zhichen Wang, Hengyi Li, Xuebin Yue, and Lin Meng, "Briefly Analysis about CNN Accelerator based on FPGA," *Procedia Computer Science*, vol. 202, pp. 277–282, 2022.

[20] Alberto Martín-Martín, Rubén Padial-Allué, Encarnación Castillo, Luis Parrilla, Ignacio Parellada-Serrano, Alejandro Morán, and Antonio García,
"Hardware implementations of a deep learning approach to optimal configuration of reconfigurable intelligence surfaces,"
Sensors, vol. 24, no. 3, p. 899, Jan. 2024.

[21] ROS, ROS - Robot Operating System, Disponible en: `https://www.ros.org/`. –Consultado el 11 julio de 2025.

[22] Wikipedia, Kinect, Disponible en: `https://es.wikipedia.org/wiki/Kinect`. –Consultado el 11 julio de 2025.

[23] M. R. Naeemabadi, B. Dinesen, O. K. Andersen, and J. Hansen, *Investigating the impact of a motion capture system on Microsoft Kinect v2 recordings: A caution for using the technologies together*. Disponible en: `https://doi.org/10.1371/journal.pone.0204052`, Sep. 2018.

[24] Intel Arria 10 Device Overview. -Intel®- Recuperado de: `https://www.intel.com/content/www/us/en/docs/programmable/683332/current/device-overview.html`, Consultado el 11 julio de 2025.

[25] TMC5130A-TA DATASHEET. -Analog Devices- Recuperado de: `https://www.analog.com/media/en/technical-documentation/data-sheets/TMC5130A_datasheet_rev1.20.pdf`, Consultado el 11 julio de 2025.

[26] TMC5130A-TA BOB Description. -Analog Devices- Recuperado de: `https://www.analog.com/media/en/technical-documentation/data-sheets/TMC5130A-BOB_datasheet_rev1.00.pdf`, Consultado el 11 julio de 2025.

[27] DIGILENT , Cmod7 Reference Manual, Disponible en: `https://digilent.com/reference/_media/reference/programmable-logic/cmod-a7/cmod_a7_rm.pdf`. –Consultado el 11 julio de 2025.

[28] Intel Corporation, FPGA AI Suite – Intel®, Disponible en: `https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/fpga-ai-suite.html`. –Consultado el 11 julio de 2025.

[29] J. Redmon and A. Farhadi, *YOLOv3: An Incremental Improvement*. Disponible: `https://arxiv.org/abs/1804.02767`, 2018.

[30] ROS Wiki Contributors, openni_camera – ROS Wiki, Disponible en: `http://wiki.ros.org/openni_camera` –Consultado el 11 julio de 2025.

[31] Marko Bjelonic, *YOLO ROS: Real-Time Object Detection for ROS*, https://github.com/leggedrobotics/darknet_ros, –Consultado el 11 julio de 2025.

[32] Intel® FGPA AI Suite: IP Reference Manual, Layer / Primitive Ranges, Disponible en: https://www.intel.com/content/www/us/en/docs/programmable/768974/2023-2/layer-primitive-ranges.html. –Consultado el 11 julio de 2025.

[33] OpenVINO, open_model_zoo, Disponible en: https://github.com/openvinotoolkit/open_model_zoo. –Consultado el 11 julio de 2025.

[34] AEW2015, altera_examples, Disponible en: https://github.com/AEW2015/altera_examples/tree/main/py_AI. –Consultado el 11 julio de 2025.