UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

" IMPLEMENTACIÓN DE LOS MODELOS DE REDES NEURONALES CONVOLUCIONALES YOLO EN UN DISPOSITIVO AVNET ULTRA 96 V2 MEDIANTE VITIS-AI "

TRABAJO FIN DE GRADO

Febrero -2025

AUTOR: Joaquín Perticari Ventura

DIRECTOR/ES: Roberto Gutiérrez Mazón

Resumen del proyecto

El presente proyecto tiene como objetivo el desarrollo de una plataforma hardware heterogénea y reconfigurable para la aplicación de inteligencia artificial en la asistencia al diagnóstico por imagen mediante microondas. Se busca diseñar un sistema con bajo consumo energético y alta velocidad de respuesta, optimizado para su implementación en entornos de apoyo clínico.

Durante un período de nueve meses, se trabajó en la creación de una guía detallada que abarcara la mayor cantidad posible de etapas del desarrollo, empleando hardware reconfigurable de la distribuidora Avnet para adaptar la solución a los requerimientos específicos del proyecto. Sin embargo, debido a limitaciones en los recursos de hardware disponibles, el desarrollo se detuvo en la fase de integración de un modelo previamente entrenado con un conjunto de datos de prueba. A pesar de esta restricción, los resultados obtenidos fueron satisfactorios, evidenciando el potencial de la plataforma.

Finalmente, el trabajo analiza las causas que impidieron la finalización completa del proyecto y plantea una perspectiva para su continuidad, identificando posibles mejoras y líneas de desarrollo futuras.

ÍNDICE DE CONTENIDO

GLOSARIO 7
CAPÍTULO 1: INTRODUCCIÓN 9
1.1. MOTIVACIÓN 9
1.2. OBJETIVOS 9
1.3. DESCRIPCIÓN 10
CAPÍTULO 2: DESCRIPCIÓN DE LA PROPUESTA 12
2.1. CONCEPTOS CLAVE 12
2.2. ANTECEDENTES AL PROYECTO REALIZADO 13
2.3. CONTEXTO/ ESTADO DEL ARTE 14
2.4. PROPUESTA 22
CAPÍTULO 3: MATERIALES Y MÉTODOS 23
3.1. MATERIALES: HARDWARE 23
3.2. MATERIALES: SOFTWARE 25
3.3. MÉTODOS: REPRESENTACIÓN DE LOS DATOS 28
CAPÍTULO 4: FLUJO DE TRABAJO 29
4.1. DESARROLLO DE LA PLATAFORMA HARDWARE 29
4.2. OBTENCIÓN DE LAS REDES NEURONALES CONVOLUCIONALES YOLO 62
CAPÍTULO 5: PUESTA EN MARCHA 86
5.1. PREPARACIÓN DE LA TARJETA 86
5.2. CONEXIÓN A LA TARJETA 86
5.3. PRUEBA DE LA PLATAFORMA HARDWARE 87
5.4. PRUEBA DE LOS MODELOS DEL MODEL ZOO DE VITIS AI 88
5.5. IMPLEMENTACIÓN DE LOS MODELOS YOLO EN FORMATO XMODEL 90
5.6. IMPLEMENTACIÓN DE LOS MODELOS YOLO EN FORMATO ONNX 110
CAPÍTULO 6: ENTRENAMIENTO PARA LA DETECCIÓN DE ARTEFACTOS MÉDICOS 112
6.1. PREPARACIÓN DE LA BASE DE DATOS 112
6.2. PREPARACIÓN, ENTRENAMIENTO Y PRUEBA DEL MODELO 114
CAPÍTULO 7: RESULTADOS 115
7.1. MODELOS 115
7.2. APLICACIÓN 117
7.3. DETECCIONES 118
CAPÍTULO 8: CONCLUSIONES 121
8.1. TRABAJOS FUTUROS 123
CAPÍTULO 9: BIBLIOGRAFÍA 124
CAPÍTULO 10: ANEXOS 125

ÍNDICE DE FIGURAS

Figura 1. Flujo de la memoria. 11
Figura 2. Entorno de desarrollo de Vitis™ AI. 20
Figura 3. Tarjeta Avnet Ultra 96 v2. 23
Figura 4. Adaptador UART/COM Avnet AES-ACC-U96-JTAG. 24
Figura 5. Conexión física del adaptador a la tarjeta. 24
Figura 6. Flujo del desarrollo de una plataforma hardware con DPU. 29
Figura 7. Herramienta de WINE a instalar. 32
Figura 8. Archivos de MobaXTerm Portable. 33
Figura 9. Modelo generado por el script de Avnet. 35
Figura 10. Ubicación del archivo .bd de definición. 36
Figura 11. Configuración de la conexión Maestro-Esclavo. 36
Figura 12. Conexión realizada de la salida LPD del MPSoC. 37
Figura 13. Configuración de los puertos AXI. 37
Figura 14. Ubicación de Generate Output Products. 38
Figura 15. Configuración de Generate Output Products. 38
Figura 16. Configuración del HDL Wrapper. 39
Figura 17. Ubicación Generate Bitstream. 39
Figura 18. Configuración de Generate Bitstream. 40
Figura 19. Ubicación de Export Platform. 40
Figura 20. Configuración de Export Platform. 41
Figura 21. Archivo XSA exportado. 41
Figura 22. Panel de petalinux-config. 43
Figura 23. Sección de Yocto Settings. 43
Figura 24. Configuración de la capa de usuario en Yocto Settings→user layers. 44
Figura 25. Configuración de Image Packaging Configuration. 44
Figura 26. Acceso a Serial Settings en Subsystem AUTO Hardware Settings. 45
Figura 27. Configuración de Serial Settings. 45
Figura 28. Panel de petalinux-config -c kernel. 46
Figura 29. Ubicación de Misc devices. 47
Figura 30. Configuración del controlador de la DPU. 47
Figura 31. Configuración de la memoria contigua DMA en Library Routines. 48
Figura 32. Configuración de CPU idle PM support. 48
Figura 33. Configuración de CPU Frequency Scaling. 49
Figura 34. Panel petalinux-config -c rootfs. 50
Figura 35. Paquete para aceleradores de hardware de vitis. 50
Figura 36. Panel de BalenaEtcher. 51
Figura 37. Configuración de la plataforma en Vitis Classic 2023.2. 53

Figura 38. Panel de Windows→Preferences con la librería de la DPU. 55
Figura 39. Crear el proyecto de aplicación..... 55
Figura 40. Ubicación de dpu_conf.vh en el Explorer..... 56
Figura 41. Configuración de DPU_TRD_system_hw_link.prj..... 56
Figura 42. Acceso a la configuración del compilador C++/C..... 57
Figura 43. Panel de configuración del compilador C++/C. 57
Figura 44. Enrutado de los archivos del compilador de la DPU. 58
Figura 45. Cambiando la prioridad a las librerías de la DPU..... 58
Figura 46. Construcción preliminar del proyecto de aplicación. 58
Figura 47. Ubicación de la opción de reparación del gestor de archivos..... 60
Figura 48. Ubicación del creador de la imagen de disco..... 61
Figura 49. Configuración de la exportación de la imagen..... 61
Figura 50. Infografía de la estructura del apartado 4.2..... 62
Figura 51. Infografía del proceso de cuantización usando Vitis AI quantizer. 64
Figura 52. Comprobación del acceso a la GPU desde Docker..... 67
Figura 53. Acceso al Docker GPU de Vitis AI y su acceso a la GPU. 68
Figura 54. Infografía de las etapas para la obtención de YOLOv4. 69
Figura 55. Infografía de las etapas para la obtención de YOLOv6m. 71
Figura 56. Infografía de las etapas para la obtención de YOLOv7. 75
Figura 57. Diferencias entre SiLU (rojo) y LeakyReLU con pendiente negativa (azul)..... 78
Figura 58. Modelos en formato XModel compilados..... 82
Figura 59. Resultados del script compilador de los modelos del model_zoo. 83
Figura 60. Modelos ONNX cuantizados. 85
Figura 61. Resultado si la DPU está correctamente integrada. 88
Figura 62. Resultados si a la DPU le faltan los controladores. 88
Figura 63. Resultados de ResNet50_PT para la figura 64. 90
Figura 64. Imagen de prueba para la detección de objetos..... 90
Figura 65. Vitis-ai-library actualizada. 111
Figura 66. Tennist por YOLOv6s..... 118
Figura 67. Tennist por YOLOv6m. 118
Figura 68. Tennis por YOLOv4. 118
Figura 69. Tennist por YOLOv7. 118
Figura 70. Kitchen por YOLOv6s..... 119
Figura 71. Kitchen por YOLOv6m..... 119
Figura 72. Kitchen por YOLOv4..... 119
Figura 73. Kitchen por YOLOv7..... 119
Figura 74. BI-RADS 4c por QUANT. 120
Figura 75. BI-RADS 4c por QAT. 120

Figura 76. BI-RADS 1 y 3 por QUANT. 120
Figura 77. BI-RADS 1 y 3 por QAT. 120

GLOSARIO

AMD: Empresa de semiconductores que fabrica CPUs, GPUs y FPGAs, incluyendo productos de la línea Xilinx.

Xilinx: Compañía especializada en FPGAs, SoCs y aceleradores de hardware, ahora parte de AMD.

FPGA (Field-Programmable Gate Array): Circuito integrado reconfigurable que permite implementar hardware personalizado mediante programación.

ZYNQ: Familia de SoCs de Xilinx que combina CPU ARM con FPGA en un solo chip.

MPSoC (Multiprocessor System-on-Chip): Multiprocesador hetereogéneo en un solo chip, integrando CPU, GPU y otros aceleradores.

UltraScale+: Familia de FPGAs y MPSoCs de alto rendimiento de AMD Xilinx.

CNN (Convolutional Neural Network): Red neuronal convolucional usada en visión artificial para el procesamiento de imágenes.

YOLO (You Only Look Once): Modelo de detección de objetos en tiempo real basado en redes neuronales convolucionales.

CPU (Central Processing Unit): Unidad central de procesamiento, encargada de tareas generales en un sistema.

GPU (Graphics Processing Unit): Unidad de procesamiento gráfico optimizada para cómputo paralelo.

DPU (Deep Learning Processing Unit): Unidad de procesamiento profundo para acelerar redes neuronales en dispositivos embebidos.

COCO (Common Objects in Context): Conjunto de datos de imágenes con anotaciones para entrenamiento de modelos de visión artificial.

BI-RADS (Breast Imaging Reporting and Data System): Sistema de categorización de hallazgos en mamografías para detección de cáncer de mama.

SiLU (Sigmoid Linear Unit): Función de activación sigmoidal linealmente ponderada, usada en redes neuronales.

ReLU (Rectified Linear Unit): Función de activación que devuelve valores positivos y cero para negativos.

Leaky ReLU (Leaky Rectified Linear Unit): Variante de ReLU que permite multiplicar valores negativos por un hiperparámetro y devolver también una salida.

CSP (Cross Stage Partial Network): Arquitectura de redes neuronales que mejora eficiencia dividiendo y fusionando características.

SPP (Spatial Pyramid Pooling): Módulo de redes neuronales que extrae características multiescala mediante pooling espacial.

SPPF (Spatial Pyramid Pooling Fast): Versión optimizada de SPP con menor costo computacional y mejor eficiencia.

AI (Artificial Intelligence): Inteligencia artificial, disciplina que desarrolla algoritmos para resolver problemas automáticamente.

PTQ (Post-Training Quantization): Cuantización postentrenamiento que reduce la precisión de un modelo al convertir las operaciones con números flotantes en números enteros.

QAT (Quantization-Aware Training): Cuantización durante el entrenamiento que reduce la precisión de un modelo al convertir las operaciones con números flotantes en números enteros.

BUSI (Breast Ultrasound Image dataset): Conjunto de datos de imágenes de ultrasonido mamario con anotaciones para detección de cáncer.

CUDA (Compute Unified Device Architecture): Plataforma de computación paralela de NVIDIA para acelerar procesamiento en GPUs.

UART (Universal Asynchronous Receiver-Transmitter): Protocolo de comunicación serie para transmisión de datos entre dispositivos.

SSH (Secure Shell): Protocolo seguro para acceso remoto a sistemas mediante redes cifradas.

XMODEL: Formato optimizado de modelos de IA para ejecución en DPUs de Xilinx.

ONNX (Open Neural Network Exchange): Formato abierto de intercambio de modelos de IA entre diferentes frameworks y hardware.

mAP (Mean Average Precision): Métrica que evalúa precisión en detección de objetos.

VART (Vitis AI Runtime): Entorno de ejecución para modelos de IA en DPUs de Xilinx.

XIR (Xilinx Intermediate Representation): Interfaz de representación intermedia de modelos optimizados para ejecución en hardware Xilinx.

CAPÍTULO 1: INTRODUCCIÓN

1.1. MOTIVACIÓN

La motivación principal detrás de este trabajo radica en la necesidad de explorar y desarrollar una plataforma hardware que permita el procesamiento eficiente de redes neuronales convolucionales, específicamente para la integración y ejecución de modelos YOLO en una FPGA. El creciente uso de arquitecturas DNN y CNN en aplicaciones de inteligencia artificial, sumado al uso de hardware específico como las FPGA, permite un procesamiento más eficiente y personalizado para cada tarea, reduciendo el consumo energético y mejorando el rendimiento en comparación con soluciones basadas únicamente en CPU o GPU. Por otra parte, el avance en las técnicas de diagnóstico médicas haciendo uso de las nuevas tecnologías requiere del aprendizaje de habilidades específicas para el equipo profesional médico, y este proceso de aprendizaje puede ser facilitado por el uso de inteligencias artificiales entrenadas de forma capaz para asistir a los profesionales en su curva de aprendizaje. Así pues, este proyecto busca poner a prueba las habilidades del estudiante para el desarrollo de una plataforma hardware completa y del entrenamiento y desarrollo de una inteligencia artificial con el algoritmo YOLO orientada al diagnóstico por imagen en una nueva tecnología de diagnóstico.

1.2. OBJETIVOS

Desarrollar una plataforma hardware customizada sobre el chip UltraScale+ 96v2 en la que hubiese una unidad Deep Learning Unit (DPU) y entrenar un modelo de inteligencia artificial para ayudar en el diagnóstico por imagen realizado con microondas.

Este trabajo tiene como objetivo desarrollar una plataforma hardware reconfigurable para la tarjeta de tecnología FPGA Avnet UltraScale+ 96v2, capaz de ejecutar modelos de redes neuronales convolucionales [Convolutional Neural Network (CNN)] en una unidad específica para redes profundas [Deep Learning Unit (DPU)], y entrenar un modelo YOLO (You Only Look Once) optimizado para la inferencia en tiempo real. Los objetivos específicos incluyen:

- Configurar una plataforma hardware que soporte redes neuronales convolucionales (CNN).
- Integrar y probar modelos YOLO en la plataforma, evaluando su rendimiento.
- Realizar una implementación eficiente que permita el uso de los recursos limitados de la FPGA.
- Explorar el proceso de cuantización y optimización de modelos, con un enfoque en YOLOv6 y YOLOv7.
- Entrenar un modelo de forma personalizada para su aplicación en el ámbito médico.

1.3. DESCRIPCIÓN

Este trabajo presenta el desarrollo completo de una plataforma basada en el dispositivo UltraScale 96v2, haciendo uso de herramientas como Vivado, Vitis y Petalinux para la configuración del entorno hardware y software. Se detalla todo el proceso, desde la instalación de sistemas operativos en el entorno de desarrollo, la configuración de los componentes hardware, la compilación de modelos de redes neuronales y la prueba de su funcionamiento. Se deja un registro claro de los pasos seguidos y las decisiones tomadas con tal de que el presente trabajo sirva a su vez como guía para aquellos que quieran desarrollar una plataforma hardware con fines similares.

El presente proyecto tiene como objetivo el desarrollo de una plataforma hardware reconfigurable para la asistencia en el diagnóstico por imagen mediante microondas, utilizando inteligencia artificial. Esta técnica experimental, aplicada al diagnóstico del cáncer de mama, busca ser una alternativa más precisa y menos invasiva que los métodos tradicionales con rayos X. Para ello, se utilizó una tarjeta Avnet Ultra 96v2, que permite un procesamiento paralelo y eficiente en comparación con CPUs y GPUs convencionales, logrando una mayor velocidad de respuesta y un menor consumo energético, gracias a la tecnología reconfigurable de las FPGA.

El desarrollo se dividió en fases críticas, comenzando con la generación de la plataforma hardware de la tarjeta mediante Vivado y la creación de un sistema operativo adaptado usando Petalinux. A partir de estos elementos, se desarrolló una aplicación capaz de reservar segmentos específicos de la FPGA para su reconfiguración sin apagar el sistema. Finalmente, se integraron redes neuronales convolucionales (CNN) preentrenadas y se validó su rendimiento utilizando el dataset COCO128. Uno de los principales desafíos fue la compatibilidad con las nuevas versiones de herramientas, que requería ajustes constantes y una curva de aprendizaje empinada.

Se probaron modelos cuantizados y sin cuantizar en imágenes de alto contraste, evaluando la precisión y confianza de las detecciones. Aunque no se lograron mediciones del consumo energético, la FPGA mostró un rendimiento similar en velocidad de respuesta a una CPU y GPU estándar, con una disminución notable en la calidad de detección debido al proceso de cuantización. Un éxito parcial se logró al obtener detecciones satisfactorias con el dataset de prueba, aunque las limitaciones técnicas impidieron completar la integración total con imágenes de microondas, se consiguió entrenar el modelo con un dataset de ecografías de seno con criterios de detección y diagnóstico BI-RADS.

El proyecto concluye con una discusión sobre los desafíos encontrados, como la pérdida de calidad de detección tras la cuantización y la falta de compatibilidad con las herramientas más recientes. Se proponen mejoras para futuras fases, incluyendo el uso de hardware más potente para aumentar la precisión del entrenamiento y reducir los errores de cuantización. Además, se recomienda adaptar la plataforma para permitir la integración en tiempo real con instrumentos médicos, mejorando la aplicabilidad práctica de la tecnología en un entorno clínico.

La siguiente infografía discurre las distintas etapas del trabajo desarrollado a continuación.

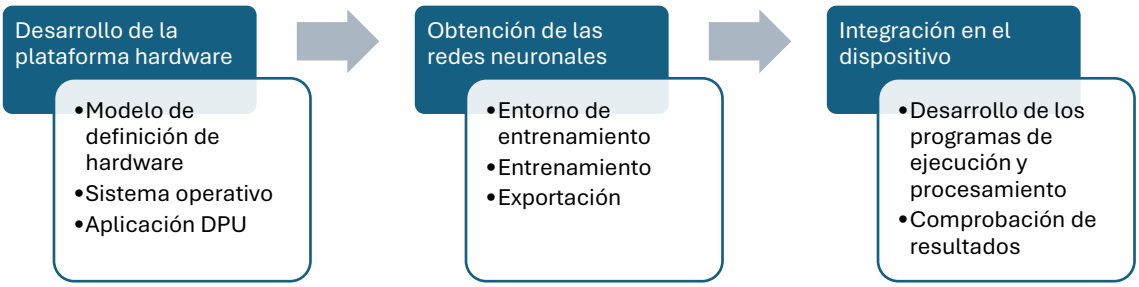

Figura 1. Flujo de la memoria.

CAPÍTULO 2: DESCRIPCIÓN DE LA PROPUESTA

2.1. CONCEPTOS CLAVE

En esta sección se introducen los conceptos fundamentales relacionados con el núcleo conceptual de esta tesis. Estos términos constituyen la base teórica y tecnológica sobre la cual se sustenta el desarrollo y análisis realizado en los apartados posteriores.

- **YOLO (You Only Look Once)**

  YOLO es un modelo de inteligencia artificial basado en redes neuronales convolucionales (CNN, por sus siglas en inglés) diseñado específicamente para la detección de objetos en imágenes y vídeos. A diferencia de otros métodos que realizan múltiples pasadas sobre una imagen para localizar y clasificar objetos, YOLO procesa la imagen completa en una sola evaluación, lo que lo hace extremadamente rápido y eficiente. Este enfoque divide la imagen en una cuadrícula, asignando a cada celda la tarea de detectar objetos en su región correspondiente. Su capacidad para realizar detección en tiempo real ha hecho que sea ampliamente utilizado en aplicaciones que van desde la conducción autónoma hasta la vigilancia y el análisis de imágenes médicas.

- **Imágenes diagnósticas por tecnología de microondas**

  Las imágenes diagnósticas basadas en tecnología de microondas representan un avance emergente en el campo de la medicina, caracterizado por su capacidad para obtener imágenes de alto contraste entre tejidos de forma completamente inocua para el paciente. Este método aprovecha las diferencias en las propiedades electromagnéticas de los tejidos, como la permitividad dieléctrica y la conductividad, para identificar áreas de interés. En particular, los tejidos sanos y los tejidos enfermos exhiben propiedades diamagnéticas distintas, lo que facilita su diferenciación. Este enfoque tiene un gran potencial en aplicaciones como la detección temprana de tumores, ya que no implica el uso de radiación ionizante y es altamente seguro para el paciente.

- **DPU (Deep Learning Processor Unit)**

  La DPU es una unidad de procesamiento diseñada específicamente para plataformas hardware reconfigurables, como las basadas en FPGA (Field-Programmable Gate Arrays) o SoCs (System on Chips). Estas unidades están optimizadas para ejecutar redes neuronales profundas (DNN, por sus siglas en inglés) de forma eficiente, aprovechando su arquitectura paralela y la posibilidad de adaptarse a requisitos específicos de cada aplicación. Al implementar físicamente los modelos de deep learning en el hardware, la DPU permite un funcionamiento significativamente más rápido y eficiente en comparación con implementaciones puramente software, reduciendo la latencia y el consumo energético. Estas características hacen que las DPU sean especialmente relevantes en escenarios de procesamiento en tiempo real, como la detección de objetos en imágenes o la inferencia en dispositivos con restricciones de recursos.

En conjunto, estos conceptos proporcionan el marco teórico y tecnológico necesario para abordar los retos y objetivos planteados en esta tesis, así como para contextualizar las soluciones implementadas en el desarrollo del trabajo.

2.2. ANTECEDENTES AL PROYECTO REALIZADO

En los últimos años, gracias al desarrollo de las diversas herramientas para las plataformas hardware han aparecido numerosos proyectos de la integración de los modelos de redes neuronales YOLO. Estas aplicaciones suelen tener como fin hacer uso de los avances en YOLO para adaptarlos a distintos dispositivos que carecen de la capacidad computacional para hacerlo, desde proyectos que permiten el análisis de las diferentes posiciones de los objetos que recorren una cinta transportadora para su paletizado hasta asistentes de conducción o desarrollo de vehículos con conducción autónoma. Gracias a la creciente comunidad de desarrolladores e interesados han aparecido diferentes tutoriales para facilitar el desarrollo de estas plataformas y la integración de la DPU, particularmente para tarjetas no distribuidas por AMD diseñadas exclusivamente para esos fines, añadiendo una nueva capa de diversidad a estos proyectos y facilitando el trabajo a los nuevos desarrolladores. Entre estos tutoriales encontraremos el desarrollado por un antiguo alumno de la UMH, Jose María Murcia en conjunción con el profesor Roberto Gutiérrez Mazón (Bailén, 2023), también uno desarrollado por Mario Bergeron, miembro veterano de la web para desarrolladores de Avnet, hackster.io, con una gran variedad de proyectos sobre la integración de Vitis-AI para diferentes plataformas (Bergeron, 2023). A su vez, cabe desatacar los tutoriales desarrollados por la empresa poseedora de la patente de la DPU, AMD y Xilinx (Xilinx, 2023), así como empresas afiliadas a ella como lo es LogicTronix (LigicTronix, 2023).

Actualmente, las problemáticas que abordan los investigadores que intentan introducir la tecnología de microondas para el diagnóstico por imagen son diversas, entre las que se haya la curva de aprendizaje de los profesionales médicos y técnicos para el uso y la interpretación de los resultados (El-Abed, 2022). Y aquí es donde es quizás más interesante la integración de la inteligencia artificial para la detección de objetos en imágenes, ya que ofrece una respuesta rápida, y si el modelo ha sido entrenado con rigurosidad, una respuesta válida, lo que puede favorecer mucho el aprendizaje gracias a los refuerzos y respuestas del modelo. Esto reduciría mucho los costes de formación de los profesionales médicos y facilitaría así su integración y aproximando sus beneficios a los pacientes.

Por otra parte, la utilización de las plataformas hardware para el procesamiento de redes neuronales tiene otra problemática, y es que durante el diseño de estas plataformas no es posible el uso de la coma flotante y se debe recurrir a números enteros. Al cambio de números de coma flotante, que cumplen la función de representar a los números reales, a números enteros, se le conoce como cuantización y es un proceso en el que siempre se perderá parte de la información. Es en este cambio donde radica la mayor deficiencia de usar las redes neuronales en la DPU frente a usarlas en la CPU o GPU, ya que aquí es donde

se pierde la mayor parte de la precisión en el modelo. Hoy en día, se han explorado diferentes maneras de llevar a cabo esta cuantización para perder la menor precisión posible, entre los métodos desarrollados destacan la cuantización después del entrenamiento (Post-Training Quantization [PTQ]) y el entrenamiento consciente de cuantización (Quantization Aware Training [QAT]), siendo esta la óptima para perder la menor precisión posible. Durante el transcurso del proyecto haremos uso de estos métodos para cuantizar lo modelos entrenados para su uso en la DPU y eventualmente podremos comprobar cuál de estos procesos nos es más beneficioso, pues esto también puede variar según la naturaleza de nuestra red neuronal.

Finalmente, debido a los avances en el diseño de la red neuronal YOLO y su extensa modularización, el proceso de cuantización, en el que es revisado por las herramientas de Vitis-AI para asegurarse de usar los operadores permitidos para la cuantización, y el compilador, que es responsable de convertir el modelo cuantizado en una serie de instrucciones para la DPU específica que se va a emplear y optimizando el uso de los nodos y operadores, serán el mayor desafío para su integración en el proyecto para nuestro objetivo específico. Este desafío requerirá de un amplio conocimiento del funcionamiento interno de una red neuronal, así como saber cómo ajustarla y modificarla sin que esta pierda las propiedades que la definen. Aunque este fue un desafío que intentamos abordar, las capacidades requeridas fueron demasiado exigentes y no se pudo completar esta tarea.

2.3. CONTEXTO/ ESTADO DEL ARTE

YOLO ha tenido una amplia trayectoria desde la fecha en la que fue lanzado en 2015, ha tenido varias revisiones y actualizaciones, así como distintas ramas y modularizaciones para cada modelo. Desde YOLOv1 donde se rompió con los modelos R-CNN y estableció un algoritmo que revisaba la imagen de una sola pasada, hasta la más reciente versión, YOLOv11 donde el modelo ha sido optimizado para el entrenamiento con pocos recursos sin perder precisión en las detecciones finales, ha habido muchos cambios que revisaremos en detalle. El análisis de las revisiones del proyecto de YOLO se ha basado principalmente en los estudios (Hussain M. , 2024) y (Hussain R. K., 2024) que han llevado a cabo una investigación sobre los avances en cada etapa.

Por otra parte, las imágenes diagnósticas por tecnología de microondas se empezaron a desarrollar desde el año 1918, con las aplicaciones de los rayos X, pero no fue hasta hace unas pocas décadas que se exploró con frecuencias diferentes menos nocivas y se descubrió la diferencia en las propiedades diamagnéticas de los tejidos. Desde entonces se han llevado a cabo diferentes integraciones y estudios para desarrollar esta tecnología y adaptarla al campo médico.

Finalmente, las unidades de procesamiento de redes profundas fueron desarrolladas por Xilinx y posteriormente por AMD dentro del programa de desarrollo de las aplicaciones

integrables en las plataformas hardware reconfigurables con la intención de fomentar el desarrollo de este campo. Desde las primeras versiones desarrolladas en 2019 hasta las más recientes versiones que han sido compatibilizadas con los formatos más usados en la industria de redes neuronales ha habido diversos cambios y actualizaciones, estas las revisaremos a continuación.

2.3.1. Evolución de YOLO (You Only Look Once)

La evolución del algoritmo YOLO ha transformado el campo de la detección de objetos en tiempo real, gracias a la introducción de mejoras continuas en su arquitectura. En este apartado, se analiza cada versión publicada hasta la fecha, con énfasis en las innovaciones fundamentales y los cambios introducidos.

1. **YOLOv1 (2015)**

La primera versión de YOLO revolucionó la detección de objetos al plantear el problema como una regresión unificada en lugar de dividirlo en etapas separadas (como se hacía en detectores basados en regiones como R-CNN). Este enfoque permitió realizar predicciones simultáneas de clases y ubicaciones de objetos en una única pasada de la red neuronal, lo que marcó un avance significativo en la velocidad de inferencia.

- **Arquitectura**: Utilizó una red convolucional de 24 capas, seguida por 2 capas completamente conectadas.
- **Ventajas**: Capacidad de procesar imágenes completas y detectar objetos en tiempo real.
- **Limitaciones**: Baja precisión para objetos pequeños y problemas con la detección de objetos cercanos debido al uso de una única rejilla.

2. **YOLOv2 (YOLO9000, 2016)**

YOLOv2 introdujo varias optimizaciones que mejoraron tanto la velocidad como la precisión:

- **Técnicas clave**:
    - Uso de anclas generadas mediante clustering k-means para mejorar la asignación de cajas delimitadoras.
    - Normalización por lotes en todas las capas, lo que aceleró el entrenamiento y mejoró la estabilidad del modelo.
    - Ajuste a imágenes con resoluciones más altas para aumentar la precisión en objetos pequeños.
- **YOLO9000**: Capacidad de detectar más de 9000 clases utilizando datos combinados de detección y clasificación, gracias a la introducción de una técnica llamada "WordTree".

3. **YOLOv3 (2018)**

Esta versión representó un rediseño significativo en la arquitectura del modelo:

- **Innovaciones principales**:
    - Uso de una red residual con bloques Darknet-53 como base, que mejoró la propagación de gradientes y la capacidad de aprendizaje.
    - Predicciones en múltiples escalas para manejar objetos de diferentes tamaños.
    - Introducción de la función de activación Leaky ReLU.
- **Impacto**: Aumentó la precisión en objetos pequeños y mejoró el rendimiento en escenarios complejos sin comprometer la velocidad.

4. **YOLOv4 (2020)**

Con YOLOv4, el enfoque se centró en la optimización para hardware de bajo coste, manteniendo la precisión y velocidad:

- **Mejoras tecnológicas**:
    - Uso de conexiones CSP (Cross-Stage Partial) para reducir la redundancia de cómputo y mejorar la eficiencia.
    - Incorporación del Spatial Pyramid Pooling (SPP) para aumentar el campo de visión de la red sin penalizar la velocidad.
    - Algoritmos avanzados como Mosaic Data Augmentation y CIOU (Complete Intersection over Union) para mejorar la generalización y la calidad de las cajas.

5. **YOLOv5 (2020)**

YOLOv5, desarrollado por Ultralytics, no es una continuación oficial, pero rápidamente ganó popularidad debido a su enfoque en la facilidad de uso y la optimización:

- **Novedades**:
    - Herramientas simplificadas para el entrenamiento y despliegue.
    - Tamaño de modelo más ligero y compatible con dispositivos móviles.
    - Introducción de configuraciones “Small”, “Medium” y “Large” para distintas necesidades.

6. **YOLOv6 (2022)**

Enfocado en la optimización de eficiencia en dispositivos con recursos limitados:

- **Mejoras clave**:
    - Arquitectura simplificada para acelerar la inferencia.
    - Mejor compatibilidad con aceleradores de hardware.

7. **YOLOv7 (2022)**

YOLOv7 incluyó mejoras específicas para lograr un rendimiento superior en aplicaciones en tiempo real:

- **Innovaciones**:
    - Óptima configuración de capa eficiente.
    - Introducción de Etapas Reductoras Expandibles para aprovechar mejor la arquitectura paralela.

8. **YOLOv8 (2023)**

De los mismos desarrolladores de YOLOv5, Ultralytics presenta un modelo que ha mejorado todos sus elementos.

- **Mejoras clave**:
    - Arquitectura optimizada para mayor precisión y velocidad.
    - Mejoras en la detección de objetos pequeños y en condiciones de iluminación variables.
    - Integración de técnicas avanzadas de aumento de datos para mejorar la robustez del modelo.

9. **YOLOvX y YOLO-World (2023)**

YOLOvX y YOLO-World representan una ramificación en la evolución de YOLO hacia la especialización modular y la integración industrial.

1. **YOLOvX**:
    - Arquitectura optimizada para personalizaciones específicas de tareas.
    - Soporte ampliado para módulos adicionales como NAS (Neural Architecture Search) y POSE.
2. **YOLO-World**:
    - Diseñado para entornos industriales, destacando por su robustez y adaptabilidad a aplicaciones de gran escala.

10. **YOLOv9, YOLOv10 y YOLOv11 (2024)**

Las versiones más recientes han consolidado a YOLO como el estándar en detección de objetos:

- **YOLOv9**:
    - Introducción de técnicas de aprendizaje semi-supervisado para mejorar la precisión con menos datos etiquetados.
    - Optimización de la arquitectura para reducir el tamaño del modelo sin sacrificar rendimiento.

- **YOLOv10**:
    - Incorporación de nuevas técnicas de aprendizaje semi-supervisado.
    - Arquitectura más ligera gracias a la integración de módulos SPPF y mejoras en la regularización del modelo.
- **YOLOv11**:
    - Optimización total para compatibilidad con hardware acelerado y entornos Edge.
        - Uso de una arquitectura híbrida CNN-transformer para mejorar la detección en condiciones adversas.
        - Reducción del número de parámetros sin afectar la precisión mediante técnicas de pruning y quantization avanzadas.
    - Introducción de algoritmos avanzados de inferencia distribuida, lo que permite divisiones de carga en sistemas multi-dispositivo.
    - Mejor integración con hardware especializado, como TPUs y FPGAs, permitiendo inferencia aún más rápida.
    - Nueva estrategia de entrenamiento con datos sintéticos y aprendizaje auto-supervisado para mejorar la robustez.

2.3.2. Desarrollo de las Imágenes Diagnósticas por tecnología de microondas

El desarrollo de imágenes diagnósticas mediante tecnología de microondas ha experimentado un auge en las últimas décadas debido a su potencial en aplicaciones médicas, como la detección de cáncer, y en otros campos industriales y de seguridad. Este apartado se enfoca en investigaciones actuales relevantes en este ámbito.

Aparte de los proyectos europeos, las investigaciones recientes han ampliado los límites de lo que es posible con la tecnología de microondas. Algunas líneas de trabajo actuales incluyen:

1. **Reconstrucción de Imágenes Híbridas**:
    - **Enfoque**: Combinar datos de microondas con otras modalidades de imagen, como ultrasonido y resonancia magnética, para aumentar la precisión diagnóstica.
    - **Aplicaciones**:
        - Mejora en la detección de cáncer de mama mediante un análisis multimodal que reduce falsos positivos y negativos.
        - Uso en evaluación de tejidos blandos, especialmente en cartílagos y lesiones musculares.
2. **Sensores Portátiles y Vestibles**:
    - **Objetivo**: Miniaturizar los sensores de microondas para aplicaciones portátiles, como dispositivos vestibles que permitan monitorear la salud en tiempo real.

- **Proyectos destacados**:
  - Dispositivos para la detección temprana de cáncer de mama en entornos clínicos y domésticos.
  - Sensores para la monitorización de cicatrización de heridas en pacientes diabéticos.
  - Equipos compactos de bajo costo para la identificación temprana de infecciones pulmonares y afecciones respiratorias.

3. **Optimización de Algoritmos de Procesamiento**:
   - **Innovación**: Implementación de aprendizaje profundo y algoritmos avanzados de reconstrucción de imágenes para mejorar la calidad de las imágenes generadas por microondas.
   - **Impacto**:
     - Reducción del ruido en las imágenes.
     - Incremento de la velocidad de reconstrucción, permitiendo su uso en tiempo real.
4. **Aplicaciones Industriales**:
   - Aunque el uso médico sigue siendo el foco principal, investigaciones recientes han explorado aplicaciones industriales como:
     - Inspección de materiales compuestos en la industria aeronáutica.
     - Detección de corrosión en estructuras metálicas.

El desarrollo de imágenes diagnósticas por microondas sigue siendo un campo prometedor que combina avances tecnológicos con una creciente cantidad de aplicaciones prácticas. Los proyectos europeos han desempeñado un papel crucial en el impulso de esta tecnología, mientras que las investigaciones actuales apuntan hacia sistemas más eficientes, versátiles y accesibles. A pesar de los desafíos en la diferenciación de tejidos y la reducción de interferencias, los avances en sensores, algoritmos de procesamiento y enfoques híbridos continúan consolidando esta tecnología como una alternativa prometedora en la detección temprana del cáncer y otras enfermedades.

La revisión del estado del arte se ha fundamentado principalmente en los trabajos de (El-Abed, 2022), (Wang, 2023), (Nazish Khalid, 2024) y (Salcedo, 2012), quienes han abordado diversos aspectos del tema.

2.3.3. Evolución de las herramientas AMD Xilinx IA

AMD, a través de su adquisición de Xilinx, ha desempeñado un papel fundamental en el desarrollo de Unidades de Procesamiento de Redes Profundas (DPU) para acelerar aplicaciones de inteligencia artificial en hardware reconfigurable. Las DPUs son componentes especializados diseñados para ejecutar operaciones de redes neuronales de manera más eficiente que las unidades de procesamiento tradicionales (CPU/GPU), permitiendo que dispositivos como SoCs (System on Chips) y FPGAs (Field-Programmable Gate Arrays) ofrezcan un rendimiento superior en tareas de inferencia de modelos de IA.

La plataforma Vitis™ AI de AMD proporciona un entorno unificado para implementar modelos de aprendizaje profundo en dispositivos adaptables, aprovechando tanto el

hardware como las herramientas de software optimizadas para cada caso de uso. Esta plataforma ha evolucionado significativamente desde su lanzamiento, introduciendo mejoras que no solo han ampliado la gama de modelos soportados, sino que también han optimizado la integración con el hardware y las herramientas de desarrollo de AMD, ofreciendo a los desarrolladores una forma eficiente de desplegar aplicaciones de IA en el borde.

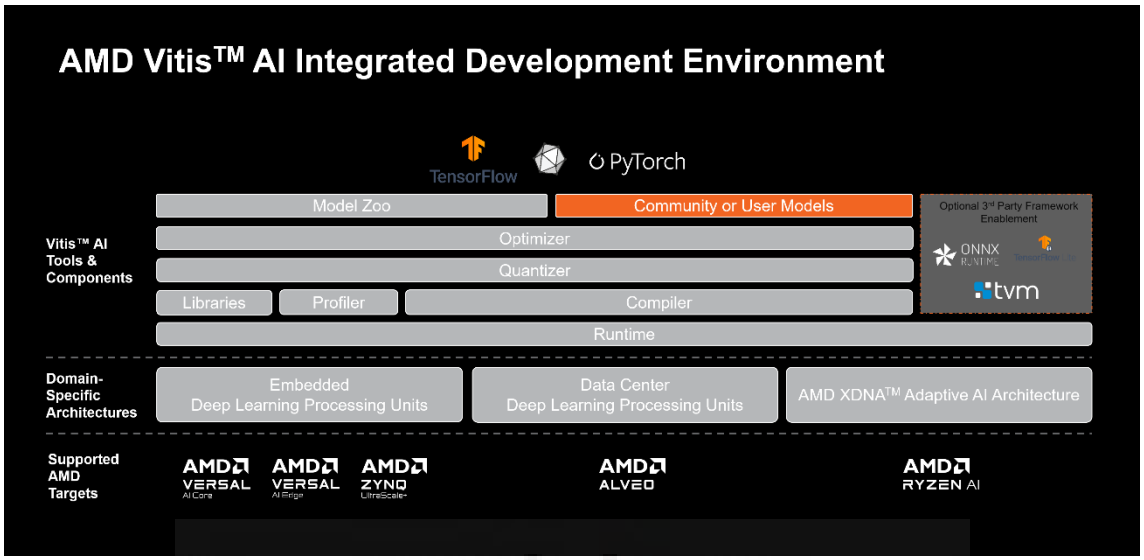

Figura 2. Entorno de desarrollo de Vitis™ AI

1. **Vitis™ AI 1.0 (2019): Un Primer Paso Hacia la Inteligencia Artificial en Hardware Adaptable**

La versión 1.0 de Vitis™ AI marcó el comienzo de la integración de redes neuronales profundas (DNNs) en dispositivos reconfigurables. Esta primera versión de la plataforma se centró en ofrecer soporte para una amplia variedad de modelos de aprendizaje profundo, adaptándose a las necesidades de las aplicaciones iniciales en inteligencia artificial. Además, permitió a los desarrolladores utilizar el ecosistema Vitis™ de AMD para trabajar con FPGAs y SoCs en un entorno común. La versión 1.0 sentó las bases para las futuras actualizaciones, implementando herramientas de desarrollo y bibliotecas que facilitaron la adopción de la tecnología en aplicaciones reales.

Características principales de Vitis™ AI 1.0:

- Soporte inicial para una variedad de modelos de IA populares.
- Implementación optimizada en hardware adaptable (FPGAs y SoCs).
- Herramientas básicas de desarrollo y análisis de rendimiento.

2. **Vitis™ AI 2.0 (2020): Mejoras en Eficiencia y Compatibilidad**

La versión 2.0 de Vitis™ AI, lanzada en 2020, introdujo una serie de mejoras clave que aumentaron la eficiencia del sistema y la compatibilidad con arquitecturas de DPU más avanzadas. Esta actualización permitió una integración más estrecha con las herramientas de desarrollo de AMD, lo que facilitó la adopción de Vitis™ AI en un mayor número de proyectos. Además, se mejoró el soporte para nuevas arquitecturas de DPU, lo que resultó en un aumento considerable en el rendimiento de la inferencia de modelos.

Mejoras en Vitis™ AI 2.0:

- Optimización de las arquitecturas DPU para mejorar el rendimiento de la inferencia.
- Incremento en la compatibilidad con más frameworks de aprendizaje profundo, como TensorFlow, PyTorch y Caffe.
- Mejora de la integración con el ecosistema de herramientas de desarrollo Vitis™ de AMD.

3. **Vitis™ AI 3.0 (2021): Expansión en la Compatibilidad y Mejora de las Herramientas de Desarrollo**

Con la llegada de Vitis™ AI 3.0 en 2021, la plataforma amplió su compatibilidad con una gama aún mayor de modelos y frameworks de aprendizaje profundo. Esta actualización también se centró en mejorar las herramientas de depuración y análisis de rendimiento, permitiendo a los desarrolladores obtener información detallada sobre el comportamiento de los modelos en hardware. Estas capacidades fueron esenciales para optimizar el uso de la DPU y mejorar la eficiencia en la ejecución de modelos complejos de inteligencia artificial.

Novedades clave en Vitis™ AI 3.0:

- Expansión del soporte para más modelos y frameworks de IA, incluidos frameworks populares como TensorFlow Lite.
- Mejora de las herramientas de análisis de rendimiento y depuración.
- Optimización de la gestión de recursos en FPGAs y SoCs para obtener el máximo rendimiento.

4. **Vitis™ AI 3.5 (2023): Optimización para Aplicaciones en el Borde**

Vitis™ AI 3.5, lanzado en 2023, ha sido una de las actualizaciones más significativas, especialmente para las aplicaciones de inteligencia artificial en el borde (edge computing). Esta versión incorporó soporte para los dispositivos Versal™ AI Edge, los cuales están diseñados para satisfacer las demandas de procesamiento de IA en tiempo real en entornos distribuidos. Las mejoras en la eficiencia energética y el rendimiento fueron clave en esta actualización, lo que permitió ejecutar modelos de IA en dispositivos con recursos limitados, sin sacrificar la precisión ni la velocidad de ejecución.

Características clave de Vitis™ AI 3.5:

- Soporte ampliado para dispositivos Versal™ AI Edge, optimizados para aplicaciones en el borde.
- Mejoras en la eficiencia energética para ejecutar inferencias de IA en dispositivos con recursos limitados.
- Integración mejorada con sistemas de baja latencia para aplicaciones de procesamiento en tiempo real.

La evolución de las Unidades de Procesamiento de Redes Profundas (DPU) y la plataforma Vitis™ AI ha sido fundamental para consolidar el uso de hardware adaptable en el ámbito de la inteligencia artificial. Desde su lanzamiento, cada iteración de Vitis™ AI ha ofrecido herramientas y optimizaciones que han ampliado el alcance y la eficiencia de las

aplicaciones de aprendizaje profundo, desde implementaciones iniciales en FPGAs y SoCs hasta soluciones avanzadas para entornos distribuidos y aplicaciones en el borde.

Estas innovaciones no solo han permitido abordar problemas complejos en tiempo real, sino que también han democratizado el uso de inteligencia artificial en áreas donde antes era impráctico. En este contexto, la capacidad de las DPUs y de Vitis™ AI para adaptarse a diversos escenarios ha sentado las bases para explorar soluciones específicas en áreas clave, como la detección de objetos mediante redes neuronales convolucionales.

La revisión sobre los avances de este entorno de desarrollo ha sido posible gracias a las guías de usuario tanto de Vitis AI UG1414 para cada versión como de sus librerías UG1354 y documentación online agrupada (AMD, 2022) y (AMD, 2023).

2.4. PROPUESTA

Nuestra propuesta se enmarca en el amplio ámbito del desarrollo de soluciones tecnológicas orientadas a la salud, planteando la creación de una plataforma hardware de bajo coste que integre un modelo de inteligencia artificial entrenado específicamente para la detección de tumores en ecografías. Este proyecto tiene como objetivo no solo diseñar y construir una plataforma funcional, sino también proporcionar una guía integral que abarque todas las etapas necesarias para su desarrollo, desde la conceptualización inicial hasta la implementación final.

Además de su enfoque en la detección de tumores, la propuesta busca contribuir al avance en la utilización de imágenes por microondas como herramienta complementaria en el diagnóstico médico. En particular, el proyecto pretende ofrecer un dispositivo accesible y eficiente que pueda facilitar tanto el aprendizaje como la formación práctica de los profesionales de la salud en el uso e interpretación de estas tecnologías. De este modo, se aspira a reforzar las capacidades formativas en entornos académicos y clínicos, favoreciendo la adopción de metodologías innovadoras en la práctica médica.

En resumen, esta iniciativa no solo responde a la necesidad de soluciones tecnológicas accesibles en el ámbito de la salud, sino que también establece una base sólida para futuras investigaciones y desarrollos en el campo de la detección médica asistida por inteligencia artificial en dispositivos reconfigurables.

CAPÍTULO 3: MATERIALES Y MÉTODOS

3.1. MATERIALES: HARDWARE

El desarrollo de este proyecto requirió el uso de los siguientes equipos:

3.1.1. Tarjeta Avnet Ultra 96V2 con FPGA AMD Xilinx ZYNQ UltraScale+ MPSoC

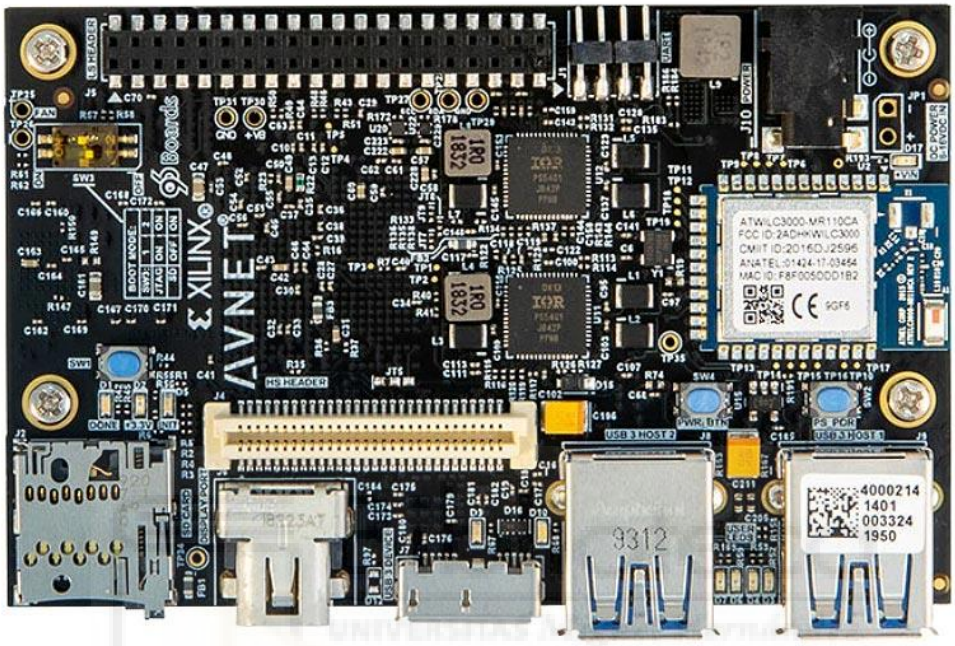

Figura 3. Tarjeta Avnet Ultra 96 v2.

La tarjeta **Ultra 96V2** de Avnet es una solución basada en la tecnología FPGA de AMD Xilinx diseñada para aplicaciones en la periferia (Edge) y proyectos de inteligencia artificial (IA). Sus características principales incluyen:

- **Procesador principal**: Incluye un sistema heterogéneo ZYNQ UltraScale+ MPSoC ZU3EG, que combina:
    - Un procesador **ARM Cortex-A53** de cuatro núcleos para tareas de aplicación.
    - Un procesador **ARM Cortex-R5** de dos núcleos, orientado a tareas en tiempo real.
    - Una **unidad de procesamiento gráfico (GPU)** Mali-400 MP2 para procesamiento gráfico básico.
- **Capacidades de hardware programable**: Un FPGA de gama alta UltraScale+ que permite aceleración personalizada y tareas de procesamiento paralelizado.

- **Memoria**: Incluye 2 GB de memoria LPDDR4 para el sistema, junto con 512 MB de memoria QSPI para almacenamiento de configuración y 8 GB de eMMC para almacenamiento adicional.
- **Conectividad**: Ofrece interfaces clave, como:
    - Ethernet Gigabit para comunicación en redes de alta velocidad.
    - USB 3.0 para periféricos y almacenamiento externo.
    - Interfaces MIPI CSI y DSI para conectividad con cámaras y pantallas.
- **Distintivos frente a otros dispositivos Edge**:
    - **Arquitectura MPSoC híbrida**, que combina procesamiento general, en tiempo real y aceleración por hardware, destacándose frente a otras soluciones Edge más limitadas como NVIDIA Jetson Nano o Raspberry Pi.
    - **Optimización para IA y procesamiento de datos en tiempo real**, superando a otras FPGA de la misma familia con capacidades más básicas, como el ZYNQ UltraScale+ ZU2.
    - **Compatibilidad con Vitis AI**, lo que permite ejecutar modelos de aprendizaje profundo optimizados para hardware con soporte para herramientas específicas como DNNDK y AI Compiler.

Además, fue necesario el uso de un módulo adaptador para su conexión al ordenador, un Avnet UART/COM, el AES-ACC-U96-JTAG.

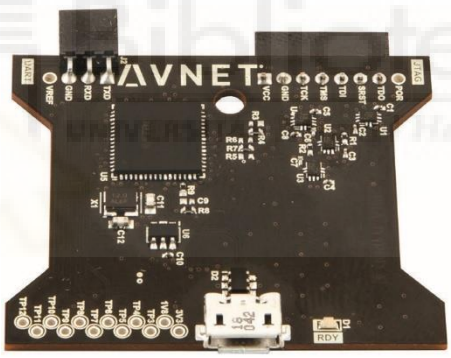

Figura 4. Adaptador UART/COM Avnet AES-ACC-U96-JTAG.

Este accesorio lo conectaremos en la tarjeta de la siguiente manera:

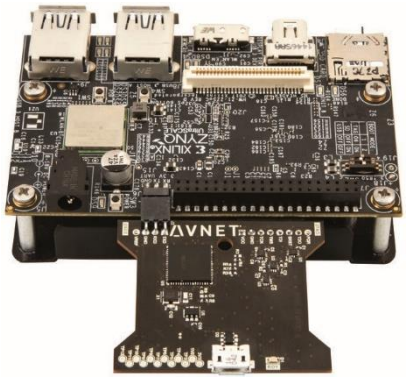

Figura 5. Conexión física del adaptador a la tarjeta.

3.1.2. Ordenador portátil personal ASUS TUF DASH F15 FX517ZM-HN002

El ordenador portátil utilizado en el proyecto cuenta con las siguientes especificaciones:

- **Procesador**: Intel® Core™ i7-12650H de 12ª generación (10 núcleos: 6 núcleos P y 4 núcleos E), con frecuencias base de 2.3 GHz y un boost de hasta 4.7 GHz.
- **GPU**: NVIDIA® GeForce RTX™ 3060 Laptop GPU con GDDR6 de 6 GB, frecuencias de 1452 MHz (base) y 1402 MHz (boost + OC).
- **Memoria RAM**: 16 GB DDR5-4800 (8 GB SO-DIMM x 2).

A pesar de estas especificaciones, la GPU integrada en este portátil resultó insuficiente para los requisitos de entrenamiento de modelos de aprendizaje profundo. Se consideró la posibilidad de utilizar una eGPU, pero esta solución excedía el presupuesto disponible. También se valoró emplear un portátil auxiliar como fuente de potencia gráfica, pero las restricciones en la fuente de alimentación hicieron inviable esta opción.

3.2. MATERIALES: SOFTWARE

El sistema operativo seleccionado para el proyecto fue **Ubuntu 22.04 LTS**, debido a su soporte completo para todas las herramientas requeridas.

Las herramientas utilizadas durante el desarrollo del proyecto se pueden clasificar en cuatro grupos principales:

3.2.1. Herramientas

1. **Herramientas de Adaptive Computing Support de AMD:**
    - **Vivado**: Plataforma de diseño para la síntesis y análisis de diseños HDL (Lenguaje de Descripción de Hardware) específicamente orientados a FPGAs de Xilinx. Fue esencial para la creación del diseño lógico del hardware y la configuración de pines.
    - **Vitis**: Herramienta de desarrollo que permite integrar aceleradores de hardware con software de aplicación. En el proyecto, se utilizó tanto para compilar los kernels de aceleración como para implementar las plataformas personalizadas.
    - **Petalinux**: Utilizada para la generación de sistemas operativos embebidos personalizados, adaptados al hardware del UltraScale+ 96V2 gracias a las librerías de Avnet publicadas en GitHub, nos permitieron hacer las configuraciones del kernel y drivers específicos.
2. **Herramientas para el uso de Vitis AI:**
    - **Docker**: Plataforma de contenedores utilizada para ejecutar entornos preconfigurados de Vitis AI y herramientas relacionadas, garantizando compatibilidad y aislamiento.
    - **XRT (Xilinx Runtime):** Framework que proporciona un entorno de ejecución para aplicaciones de hardware acelerado, facilitando la comunicación entre software y FPGA.
    - **PyTorch y ONNX**: Frameworks utilizados en las etapas de entrenamiento, optimización y exportación de modelos de aprendizaje profundo. PyTorch

permitió el desarrollo y entrenamiento de modelos, mientras que ONNX se empleó para comprobar el funcionamiento de los modelos en el PC, y como método de exportación universal de los distintos modelos en línea.

3. **Herramientas para compatibilizar hardware y software:**

    - **CUDA**: Framework de NVIDIA utilizado para tareas de aprendizaje profundo, específicamente en etapas iniciales del proyecto donde la GPU del portátil personal se usó para pruebas de entrenamiento y validación.
    - **Wine**: Software que permitió ejecutar aplicaciones diseñadas para sistemas Windows en el entorno Linux, necesario para algunas herramientas secundarias como MobaXTerm.
    - **CMake**: Herramienta de gestión de compilaciones empleada para la configuración y construcción de proyectos en C++.
    - **Paquetes de la BIOS de ASUS**: Controladores y herramientas específicas requeridas para garantizar la compatibilidad del sistema operativo con el hardware del portátil ASUS TUF DASH F15.

4. **Herramientas para la puesta en marcha y visualización de resultados:**

    - **MobaXterm**: Cliente de terminal y servidor SSH utilizado para establecer conexiones remotas con la tarjeta UltraScale+ 96V2. Fue clave para la ejecución, monitorización y transferencia de archivos en el dispositivo, así como para visualizar los resultados del procesamiento gracias a la interfaz SSH -X.

    - **BalenaEtcher**: Herramientas para el quemado de imágenes del Sistema Operativo en dispositivo de almacenamiento. Necesario para la preparación de la tarjeta SD que almacenará el S.O. del dispositivo.

3.2.2. Datasets

Dentro de los materiales de software también están incluidos los conjuntos de datos que se han utilizado para los entrenamientos de los distintos modelos Las herramientas utilizadas durante el desarrollo del proyecto se pueden clasificar en dos grupos principales:

1. **COCO**: El dataset COCO (Common Objects in Context) es un conjunto de datos ampliamente utilizado en visión por computador, diseñado para facilitar el entrenamiento y evaluación de modelos de detección de objetos, segmentación y captioning de imágenes. Contiene más de 200,000 imágenes etiquetadas, con aproximadamente 80 clases de objetos distribuidos en escenarios reales, capturando contextos variados y complejos. Este dataset es especialmente valioso para la investigación y desarrollo de algoritmos de inteligencia artificial debido a la calidad y diversidad de sus datos, que reflejan desafíos del mundo real.
    - **COCO128**: Este es un subconjunto de las imágenes y etiquetas ofrecidas por COCO, aunque estas imágenes son extraídas de las versiones de años anteriores para evitar la redundancia si se ha entrenado con COCO. Este conjunto se puede obtener con el script disponible en el repositorio de Ultralytics.

2. **Breast UltraSound Tumor Detection BUSI**: Este conjunto de datos fue desarrollado para su integración en modelos de detección de objetos. Los resultados del estudio publicado se recogen en el siguiente enlace https://www.cancerimagingarchive.net/collection/breast-lesions-usg/.

La comunicación con el chip de Avnet Ultra 96V2 se llevó a cabo a través de sus puertos **UART**. Para establecer esta conexión, se utilizó un adaptador proporcionado por Avnet que permitía convertir la señal UART/COM a micro USB, el adaptador Avnet JTAG-UART, lo cual facilitó la conexión con el ordenador personal. Este adaptador fue esencial para garantizar una transmisión de datos estable y confiable.

La configuración del puerto UART se realizó con una velocidad de transmisión de **115200 bits por segundo**, que se considera estándar para este tipo de dispositivos embebidos. A través de esta conexión, se obtuvo acceso textual directo al dispositivo, lo que permitió la configuración inicial del sistema operativo y su red.

Uno de los pasos fundamentales consistió en la modificación del archivo **wpa_supplicant.conf**, que es responsable de gestionar las conexiones inalámbricas en sistemas Linux. Esta modificación permitió conectar la tarjeta a una red Wi-Fi específica, habilitando así el acceso remoto al dispositivo mediante el protocolo **SSH** (Secure Shell).

Para la gestión y visualización de esta conexión SSH, se empleó la herramienta **MobaXterm**, que ofreció una interfaz robusta y multifuncional. Esta herramienta no solo facilitó el acceso remoto al dispositivo, sino que también incluyó características adicionales clave para el desarrollo del proyecto:

- **Visualización remota de imágenes:** MobaXterm permitió visualizar las imágenes generadas como salida por **OpenCV**, directamente a través de la conexión SSH. Esto fue particularmente útil para verificar y ajustar en tiempo real los resultados del procesamiento de imágenes en el dispositivo.
- **Interfaz de exploración de archivos:** La herramienta incluyó un explorador gráfico de archivos que simplificó la transferencia de datos entre el ordenador personal y la tarjeta, permitiendo una gestión más eficiente de los recursos del proyecto.

Este método de adquisición y gestión de datos resultó esencial para el desarrollo del proyecto, ya que estableció una comunicación efectiva con el hardware y proporcionó un entorno eficiente para la supervisión y manipulación de los resultados generados.

3.3. MÉTODOS: REPRESENTACIÓN DE LOS DATOS

La representación de los datos de salida de los modelos se llevó a cabo mediante la utilización de la biblioteca **OpenCV**. Esta herramienta permitió superponer sobre la imagen de entrada las cajas delimitadoras correspondientes a las detecciones realizadas por los modelos. Posteriormente, el resultado procesado se mostró en una ventana de visualización, siempre que el sistema contara con la capacidad de desplegar imágenes.

En los casos donde no fue posible generar una ventana de display, el programa gestionó la salida de manera alternativa, mostrando los datos en un formato de texto estructurado. Esta representación textual incluyó información relevante sobre las detecciones, como las clases identificadas, las coordenadas de las cajas delimitadoras, y las áreas correspondientes.

Este enfoque aseguró que los resultados fueran accesibles y comprensibles para el usuario final, independientemente de las limitaciones de hardware o software del sistema anfitrión.

CAPÍTULO 4: FLUJO DE TRABAJO

En el siguiente capítulo recorreremos los dos flujos de trabajo que hemos seguido para obtener los componentes necesarios para poner a prueba nuestro dispositivo en el procesamiento de redes neuronales entrenadas por nosotros.

4.1. DESARROLLO DE LA PLATAFORMA HARDWARE

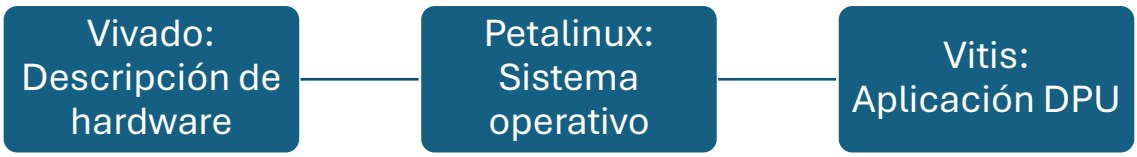

Figura 6. Flujo del desarrollo de una plataforma hardware con DPU.

4.1.1. Instalación de Ubuntu y los drivers de compatibilización

Muy específico de mi caso para explicar la generalidad

Preparación del entorno, carpeta /opt/xilinx y /opt/WorkSpace

4.1.2. Instalación de las herramientas Vivado y Vitis y adición de las tarjetas de Avnet al repositorio

La instalación de las herramientas necesarias para el desarrollo de la plataforma hardware se realizó descargando **Vivado** y **Vitis** desde la página oficial de AMD (https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2023-2.html). Para ello, fue necesario registrarse en el sitio web de AMD e indicar el propósito previsto para el uso de las herramientas.

4.1.2.1. Proceso de Instalación

1. **Descarga del Instalador**

   Se seleccionó el instalador específico para sistemas operativos basados en Linux, asegurando compatibilidad con el entorno de trabajo.

2. **Preparación del Sistema**

   Antes de ejecutar el instalador, se modificaron los permisos del archivo ejecutable y de la carpeta de destino para permitir su instalación. Esta carpeta de destino tendrá que estar ubicada de forma accesible desde el directorio raíz, por la guía ofrecida por Jose María Murcia, se recomendará la siguiente carpeta '`/opt/xilinx/`'. En la mayoría de los casos, se requiere ejecutar el instalador con privilegios de superusuario utilizando el comando `sudo`. Para este proyecto, fue suficiente con ejecutar el instalador con el comando `sudo -i` y dentro del entorno de superusuario ejecutar `./FPGAs_AdaptiveSoCs_Unified_2023.2_1013_2256_Lin64.bin`.

3. **Configuración de la Instalación**

   Durante el proceso de instalación, se configuraron las opciones para optimizar el uso del espacio en disco sin comprometer la funcionalidad necesaria. Las preferencias seleccionadas fueron:

   - **Vitis Unified Software Platform**: Incluye Vitis, Vivado y Vitis HLS.
   - **DocNav**: Herramienta de navegación documental.
   - **Devices**:
     - "Install Devices for Kria SOMs and Starter Kits".
     - "Devices for Custom Platforms", seleccionando los SoCs Zynq-7000 y ZYNQ UltraScale+ MPSoC.

4. **Confirmación y Finalización**

   Tras aceptar los términos y condiciones, se configuró la ruta de instalación en `/opt/xilinx`. Aunque se recomienda crear asociaciones de herramientas y enlaces de escritorio, estos no se configuraron debido a problemas técnicos específicos. Finalmente, se inició la instalación, que tuvo una duración aproximada de 30 a 45 minutos, y concluyó de manera exitosa.

4.1.2.2. Adición de la Tarjeta Avnet al Repositorio

Para que Vivado reconociera la tarjeta UltraScale+ 96v2 de Avnet, fue necesario agregar sus archivos de definición al repositorio de tarjetas. El procedimiento fue el siguiente:

1. **Descarga de los Archivos de Definición (BDF)**

   Se descargaron los archivos específicos proporcionados por Avnet para la tarjeta UltraScale+ 96v2.

2. **Ubicación de los Archivos**

   Los archivos se copiaron en la carpeta correspondiente de Vivado:

```
/opt/xilinx/Vivado/2023.2/data/xhub/boards/
```

Esta es la ubicación donde Vivado almacena la definición de las tarjetas compatibles, incluyendo aquellas de terceros.

3. **Instalación de Librerías Adicionales**

   En el directorio de instalación de Vitis (`/opt/xilinx/Vitis/2023.2/`), se ejecutó el script `installLibs.sh` utilizando el comando:

```
bash -x installLibs.sh
```

Este script asegura que las bibliotecas necesarias para las herramientas estén correctamente configuradas en el sistema.

Este flujo de trabajo garantizó que las herramientas Vivado y Vitis estuvieran completamente operativas y que la tarjeta Avnet UltraScale+ 96v2 pudiera integrarse correctamente al entorno de desarrollo.

4.1.3. Instalación de Petalinux y XRT

Descargamos de aquí https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2023-2.html y preparamos las cosas antes de instalar. Para saber que instalar he seguido la siguiente guía de instalación de Petalinux de Trenz (Hartfiel, 2023), donde nos indica los problemas de instalación de esta versión y como sortearlos. Lo primero es instalar las arquitecturas necesarias, luego los paquetes necesarios y finalmente configurar el dpkg.

La instalación de **PetaLinux** y **XRT** es fundamental para garantizar la integración del software y el hardware en el dispositivo UltraScale+ 96v2. A continuación, se detalla el proceso seguido:

4.1.4. Descarga de los Recursos

1. **PetaLinux**

   La herramienta fue descargada desde la página oficial de AMD (https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2023-2.html).

2. **Guía de Instalación**

   Se empleó la guía proporcionada en el wiki de Trenz Electronix (Hartfiel, 2023), la cual detalla los pasos necesarios para sortear posibles problemas específicos de la versión de PetaLinux utilizada.

4.1.4.1. Preparación del Entorno

Antes de proceder con la instalación, se llevaron a cabo las siguientes tareas:

- **Arquitecturas Requeridas**: Se añadieron las arquitecturas necesarias para la compatibilidad con el entorno de desarrollo.

```
sudo dpkg –add-architecture i386
```

- **Instalación de Paquetes**: Se instalaron todos los paquetes adicionales que la guía recomendaba como imprescindibles para garantizar una instalación exitosa.

```
sudo apt-get install iproute2 gawk python3 build-essential gcc git make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1 gnupg wget git diffstat chrpath socat xterm autoconf libtool tar unzip texinfo zlib1g-dev gcc-multilib automake zlib1g:i386 screen pax gzip cpio python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint bc libtinfo5 subversion u-boot-tools -y
```

- **Configuración de** `dpkg`: Se ajustaron las configuraciones de `dpkg` para evitar conflictos durante la instalación.

```
sudo dpkg-reconfigure dash
```

En la ventana que se nos abra debemos seleccionar no, el cambió debería realizarse sin problemas, pero se puede comprobar ejecutando `ls -l /bin/sh`

4.1.4.2. Instalación de XRT

El primer paso fue instalar el **Xilinx Runtime (XRT)**, el cual proporciona las bibliotecas y controladores necesarios para interactuar con la FPGA. Este componente se instaló automáticamente en la carpeta predeterminada `/opt/xilinx/`, sin necesidad de indicar manualmente la ruta de instalación. El proceso consiste en descargar el archivo para tu arquitectura específica (https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/alveo.html), cambiarle los permisos para darle permisos de ejecución e instalarlo por medio de APT usando `sudo apt install ./xrt_202320.2.16.204_22.04-amd64-xrt.deb`.

4.1.5. Instalación de PetaLinux

Una vez configurado el entorno, se procedió con la instalación de PetaLinux:

1. **Ejecución del Instalador**: El instalador se ejecutó siguiendo los pasos recomendados en la guía de JMM (Bailén, 2023).

   ./petalinux-v2023.2-10121855-installer.run –dir *opt/*xilinx/Petalinux/2023.2/

2. **Aceptación de los Términos y Condiciones (TyC)**: Durante el proceso, se solicitó leer y aceptar los términos y condiciones. Es importante mencionar que, para aceptar los TyC, se debe presionar la tecla `q` para salir de la visualización del texto antes de confirmarlos.

Este procedimiento aseguró la instalación correcta de PetaLinux, completando así la preparación del entorno de desarrollo necesario para el proyecto.

4.1.6. Instalación de BalenaEtcher y MobaXterm

Para descargar BalenaEtcher iremos a la siguiente página (https://etcher.balena.io/#download-etcher) y descargaremos la versión AppImage, que es un archivo ejecutable que no requiere de instalación. Con el archivo descargado le daremos los permisos de ejecución y ya estaría disponible para ejecutarse desde la terminal.

Por otra parte, para MobaXterm, ya que haremos uso de WINE para su ejecución, lo primero sería instalar WINE. Para ello, iremos al instalador de herramientas oficial de Ubuntu, que está instalado por defecto, y buscaremos WINE en el buscador. Seleccionaremos wine y nuestra versión concreta de Ubuntu e instalaremos.

Figura 7. Herramienta de WINE a instalar.

Finalmente, descargaremos el archivo `.exe` de MobaXterm y lo descomprimiremos en una carpeta que nos sea cómoda para ejecutarlo. Yendo a la siguiente página (https://mobaxterm.mobatek.net/download-home-edition.html) descargaremos la versión portable. Yo la descomprimí en la carpeta de `/home/Downloads/` y resultó en una carpeta donde contenía todos los archivos necesarios para la ejecución.

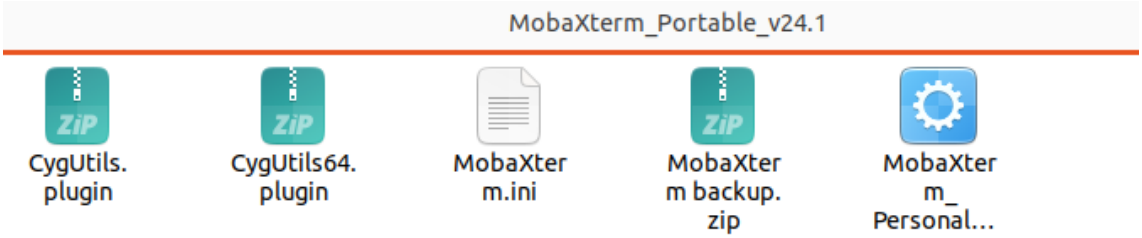

Figura 8. Archivos de MobaXTerm Portable.

4.1.7. Uso de los scripts facilitados por Avnet: modificaciones para su parada en un segmento específico.

El proceso de configuración se basó en la guía de Mario Bergeron (Bergeron, 2023), en la cual se detallan los repositorios necesarios para la configuración del entorno. A continuación, se describe el procedimiento llevado a cabo, incluyendo las modificaciones realizadas para solventar problemas de compatibilidad y permitir la detención del script en un punto específico.

4.1.7.1. Descarga de Repositorios

Para organizar el entorno, se creó una carpeta denominada **Avnet_2023_2**, donde se descargaron las ramas correspondientes de los repositorios facilitados por Avnet. No obstante, se decidió descargar únicamente cuatro de los repositorios indicados en la guía:

```
git clone https://github.com/Avnet/bdf
git clone -b 2023.2 https://github.com/Avnet/hdl
git clone -b 2023.2 https://github.com/Avnet/petalinux
git clone -b 2023.2 https://github.com/Avnet/meta-avnet
```

4.1.7.2. Modificaciones Realizadas

1. **Detención del Script**
   Para evitar que el script continúe de forma automática, se modificó el archivo `petalinux/scripts/common.sh`. En este caso, tras la línea 169, se agregó un bloque de código que solicita al usuario confirmar si desea continuar:

```
# Pausar antes de la operación importante
read -p "¿Desea continuar? (y/N): " -n 1 -r respuesta

if [[ $respuesta == "y" || $respuesta == "Y" ]]; then
  # Realizar la operación importante
  echo "Realizando operación importante..."
```

```
else
  echo "Saliendo..."
  exit 1
fi
```

2. **Ajuste del Número de Núcleos Utilizados por Vivado**
   En el archivo `hdl/scripts/project_scripts/u96v2_sbc_base.tcl`, se modificó la línea 173 para ajustar la carga de trabajo al hardware disponible. Por defecto, el script emplea el total de núcleos disponibles en el sistema para acelerar los cálculos, pero esto puede ser problemático en ordenadores con limitaciones de RAM o sistemas de enfriamiento, como los portátiles.

```
##launch_runs impl_1 -to_step write_bitstream -jobs $numberOfCores
launch_runs impl_1 -to_step write_bitstream -jobs 4
```

En este caso, se redujo el número de núcleos utilizados de 16 (valor predeterminado) a 4, evitando que el proceso se detenga debido a problemas de recursos.

4.1.7.3. Ejecución del Script de PetaLinux

Una vez realizadas las modificaciones, se ejecutó el script principal desde la carpeta `petalinux/` siguiendo estos pasos:

1. **Configuración del Entorno**

   Antes de ejecutar el script, fue necesario cargar las herramientas correspondientes con los siguientes comandos:

```
source /opt/xilinx/Vitis/2023.2/settings64.sh
source /opt/xilinx/Vivado/2023.2/settings64.sh
source /opt/xilinx/xrt/setup.sh
source /opt/xilinx/Petalinux/2023.2/settings.sh
```

2. **Ejecución del Script**

   Finalmente, se ejecutó el script de PetaLinux con el siguiente comando:

```
bash scripts/make_u96v2_sbc_base.sh
```

Durante la ejecución, el sistema mostró mensajes informativos sobre las operaciones realizadas y se detuvo en el punto especificado gracias a las modificaciones previas.

Este flujo de trabajo permitió una configuración personalizada y adaptada a las limitaciones del equipo, garantizando una ejecución eficiente y controlada.

4.1.8. Modificación del modelo de Vivado generado.

Este apartado describe los pasos para realizar las modificaciones necesarias en el proyecto generado por Vivado, utilizando su interfaz gráfica. A continuación, se detallan los procedimientos para corregir la plataforma hardware, con la inclusión de imágenes explicativas que reforzarán cada uno de los pasos.

1. **Acceso al Proyecto Generado**
Después de ejecutar el script, podemos desplazarnos a la carpeta `hdl/` desde la misma ventana de comandos. Dentro de esta carpeta, se habrá generado una subcarpeta llamada `hdl/projects/`, donde encontraremos el archivo `.xsa`, que corresponde a la exportación del proyecto completo. Para continuar, abrimos Vivado desde la terminal con el siguiente comando:

```
vivado &
```

2. **Abrir el Proyecto en Vivado**
Una vez dentro de Vivado, en el apartado **Quick Start**, seleccionamos **'Open Project >'** y, en el explorador de archivos, navegamos hasta la carpeta `Avnet_2023_2/hdl/projects/u96v2_sbc_base_2023_2/` y abrimos el archivo `u96v2_sbc_base.xpr`. Esto abrirá el proyecto en Vivado.

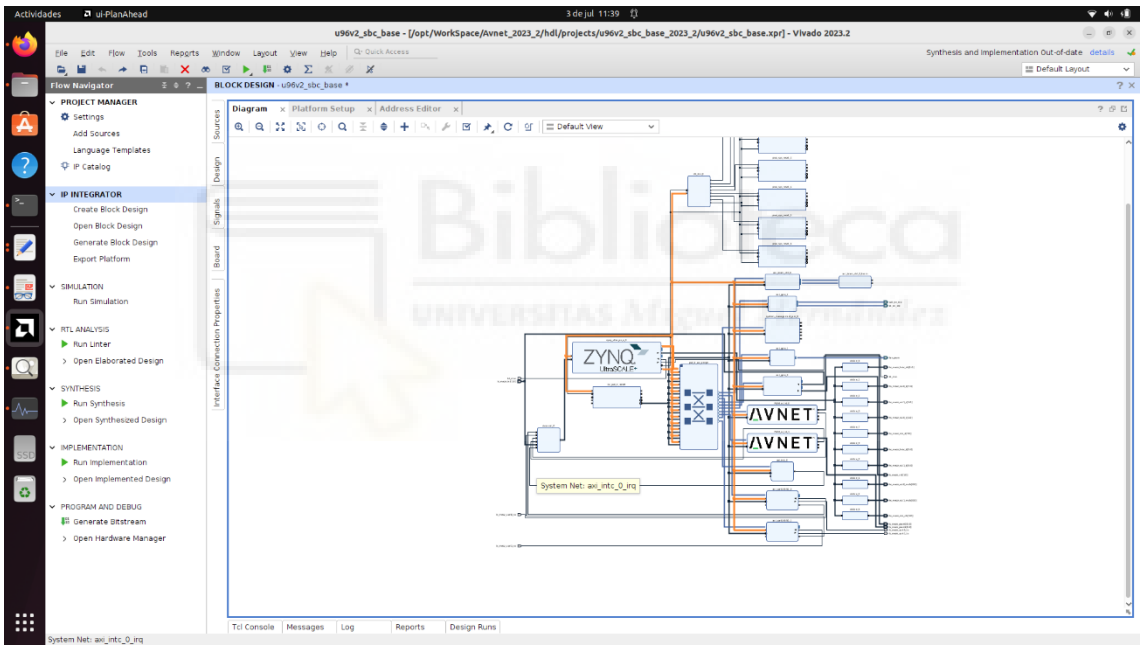

Figura 9. Modelo generado por el script de Avnet.

3. **Exploración de la Interfaz de Vivado**
La interfaz de Vivado se divide en cuatro partes principales:

- **Flow Navigator**
- **Project Manager**
- **TCL Console**
- **Barra de herramientas superior**

Dentro del **Project Manager**, en la sección **Sources**, desplegamos las opciones hasta encontrar el archivo `.bd`. Al hacer doble clic sobre este archivo, se abrirá el diseño gráfico en el área de trabajo a la derecha.

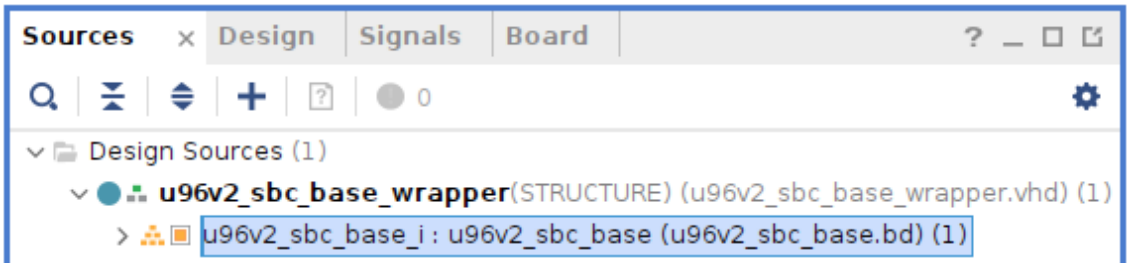

Figura 10. Ubicación del archivo .bd de definición.

4. **Modificación del Diseño en Vivado**

Dentro de la pestaña **Diagram**, hacemos doble clic sobre **ZYNQ**. En las pestañas de la izquierda, seleccionamos **PS-PL Configuration**. Aquí desplegamos **PS-PL Interface > Master Interface** y desactivamos **AXI HPM0 FPD**, mientras activamos **AXI HPM0 LPD**.

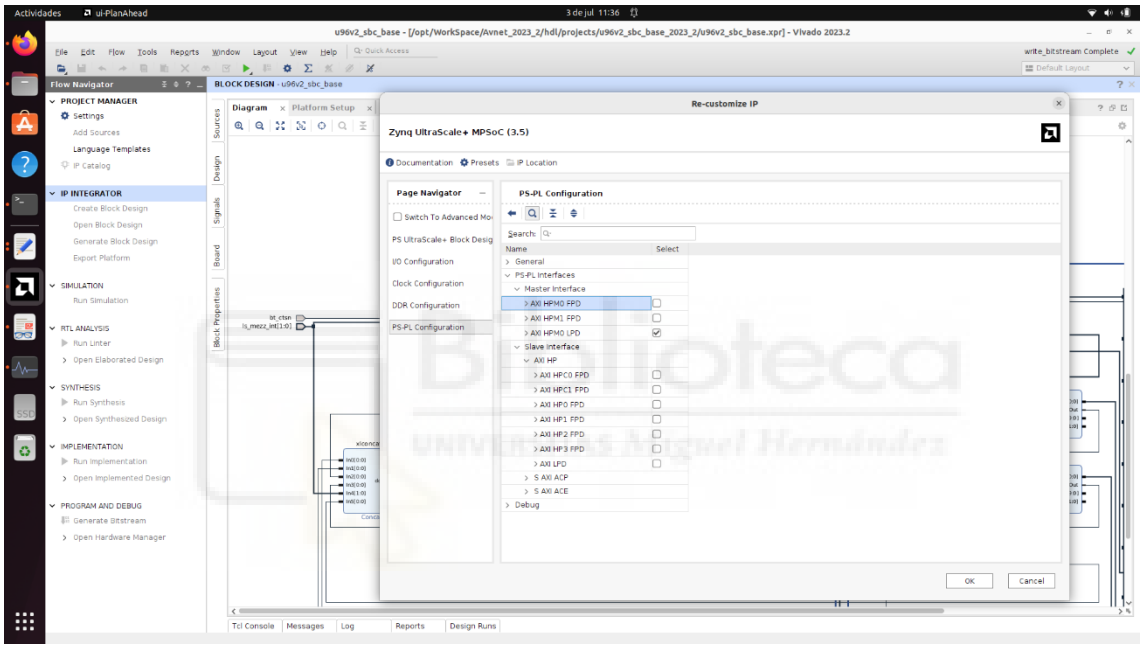

Figura 11. Configuración de la conexión Maestro-Esclavo.

A continuación, guardamos y salimos de esta sección para volver al **Diagram**. En este punto, se habrá abierto una conexión que podremos corregir utilizando la opción **Run Connection Automation**.

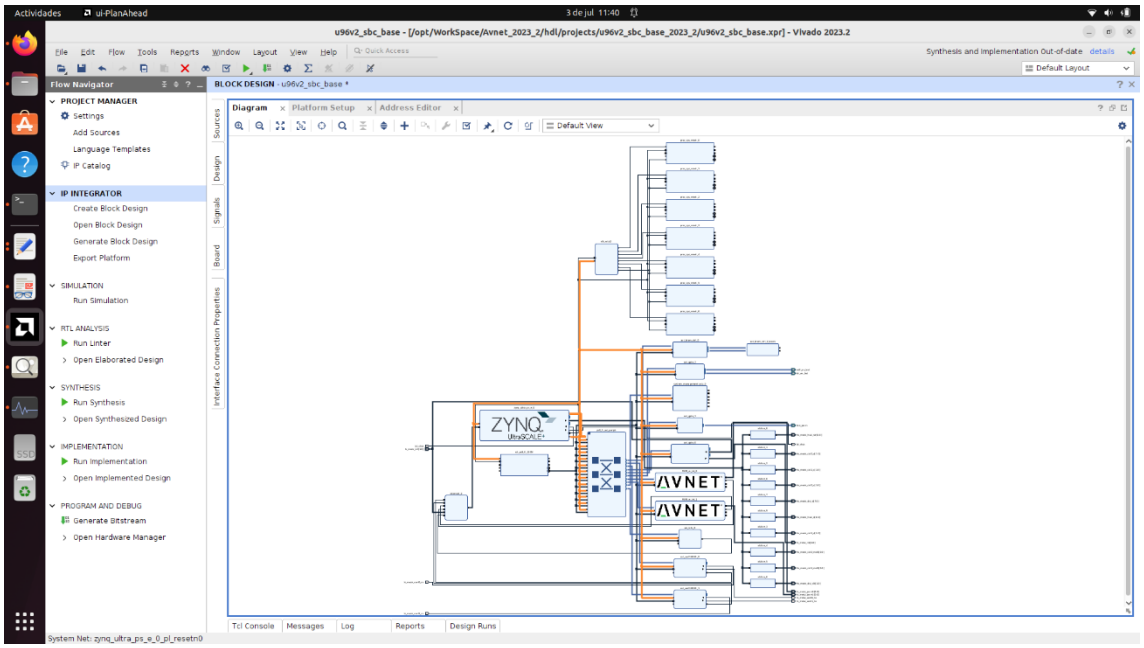

Figura 12. Conexión realizada de la salida LPD del MPSoC.

5. **Configuración de los Pines AXI**

Pasamos a la pestaña **Platform Setup**, luego a **AXI Port**, seleccionamos **zynq_ultra_ps_e_0** y activamos los puertos **M_AXI_HPM0_FPD**, **M_AXI_HPM1_FPD** y **S_AXI_HP3_FPD**. Además, debemos indicar los nombres de los puertos como se muestra en la imagen.

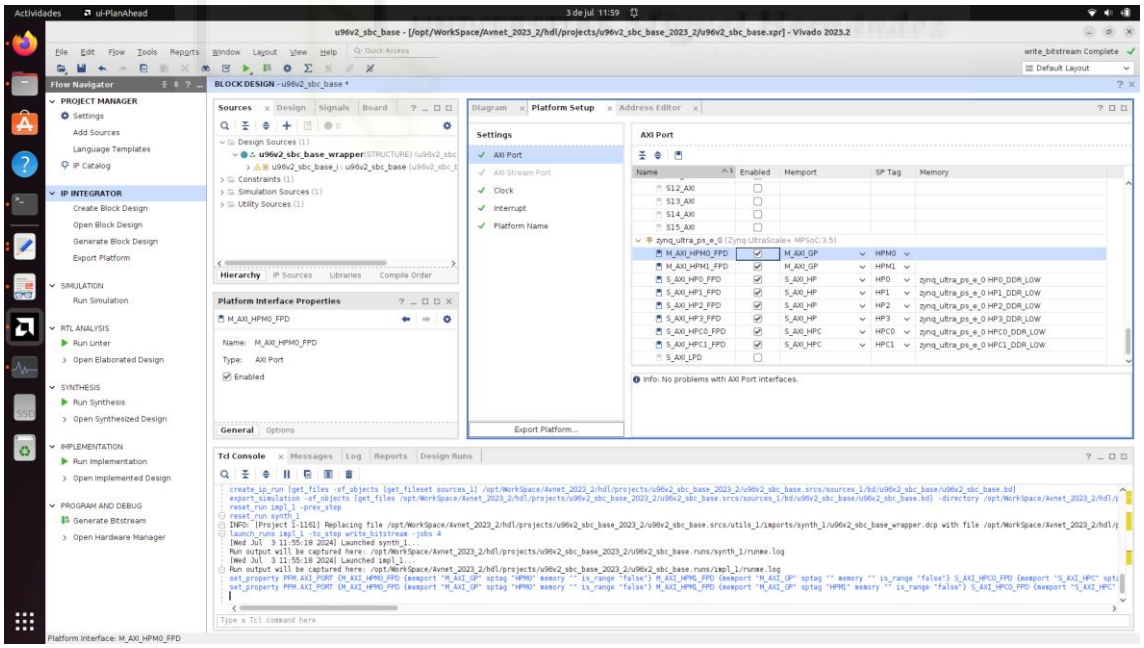

Figura 13. Configuración de los puertos AXI.

Una vez realizadas las modificaciones, damos clic en **Validate** y aceptamos a pesar de los errores que puedan aparecer.

6. **Generación de Productos de Salida**

Luego, en la sección **Sources**, hacemos clic derecho sobre el archivo `.bd` y seleccionamos **Generate Output Products**.

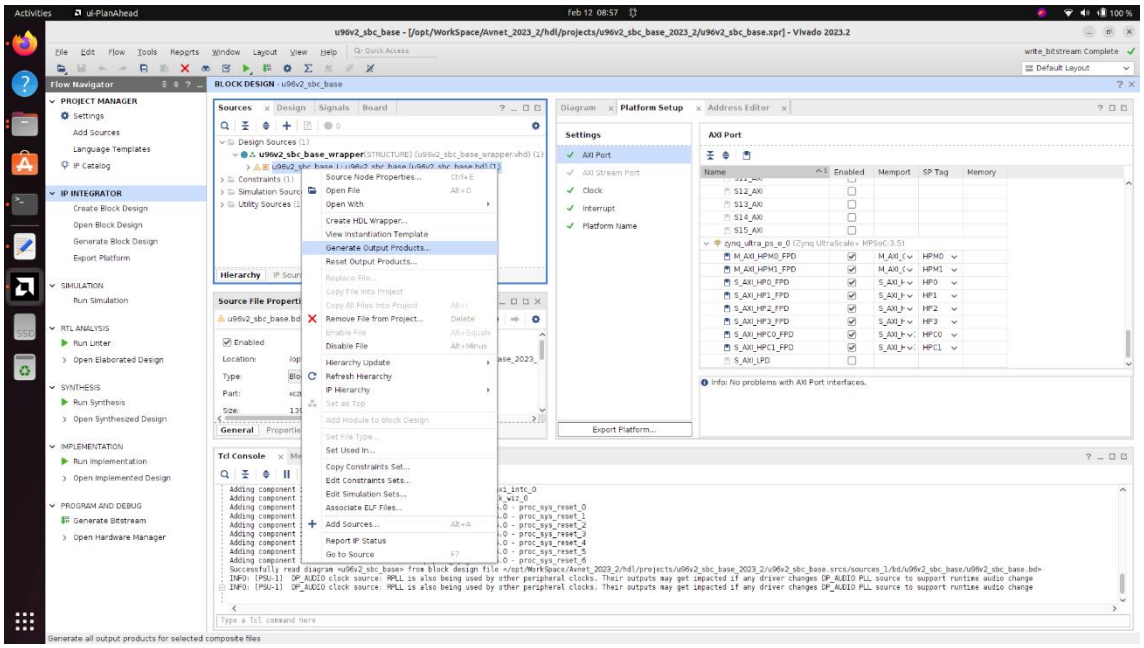

Figura 14. Ubicación de Generate Output Products.

Aquí, podemos optar por aumentar el número de trabajos simultáneos a 4 para acelerar el proceso, aunque no es estrictamente necesario. Simplemente le damos a **Generate**.

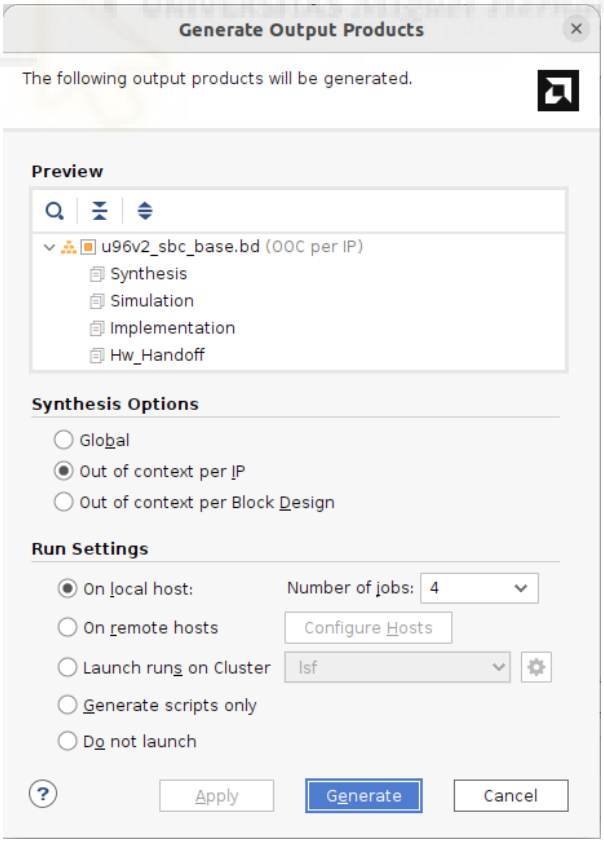

Figura 15. Configuración de Generate Output Products.

Una vez finalizado este proceso, hacemos clic derecho nuevamente y seleccionamos **Create HDL Wrapper**. Dejamos la opción por defecto y aceptamos. Esta operación transcribirá el modelo creado en la interfaz gráfica a código HDL, proporcionando así la descripción completa del hardware.

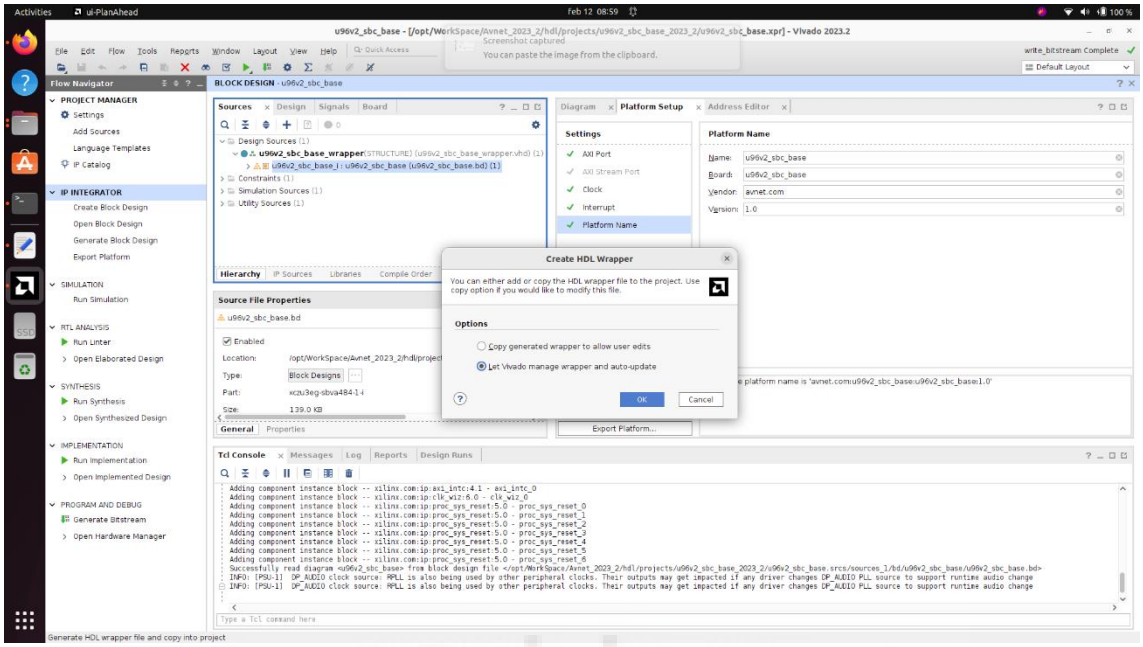

Figura 16. Configuración del HDL Wrapper.

7. **Generación del Bitstream**

El siguiente paso es crear el bitstream, el cual permitirá comprobar la definición del modelo generado. Para ello, en el panel de **Flow Navigator**, seleccionamos la opción **Generate Bitstream** (la última opción en el panel).

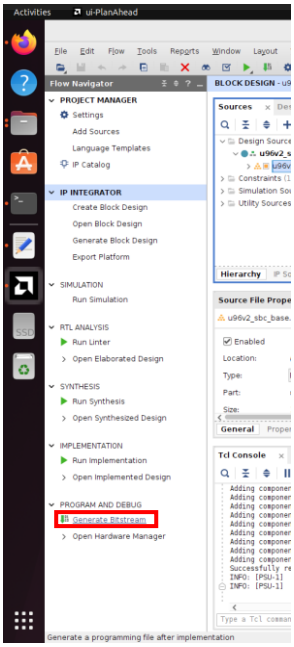

Figura 17. Ubicación Generate Bitstream.

Al seleccionarla, podremos elegir una carpeta donde se guardarán los resultados de síntesis e implementación, o podemos dejar la ruta por defecto.

Al igual que en los pasos anteriores, podemos configurar el número de trabajos simultáneos a 4 para acelerar el proceso, aunque nuevamente esto no es obligatorio. Cuando el proceso termine sin errores, podremos confirmar que el modelo está bien definido.

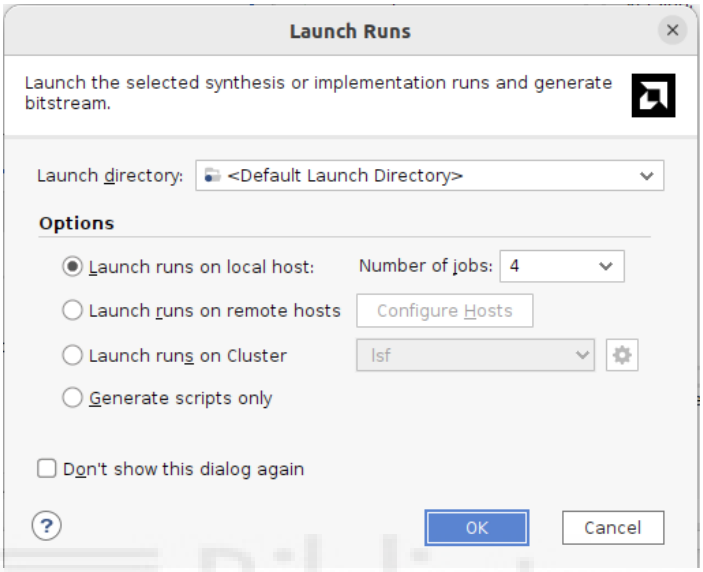

Figura 18. Configuración de Generate Bitstream.

8. **Exportación del Proyecto**

Una vez que el bitstream ha sido generado correctamente, es necesario exportar el proyecto para usarlo en las demás herramientas de Xilinx. Para ello, buscamos la opción **Export**, la cual podemos encontrar en el **Flow Navigator** a media altura, en el menú desplegable de **File**, o en **Platform Setup**.

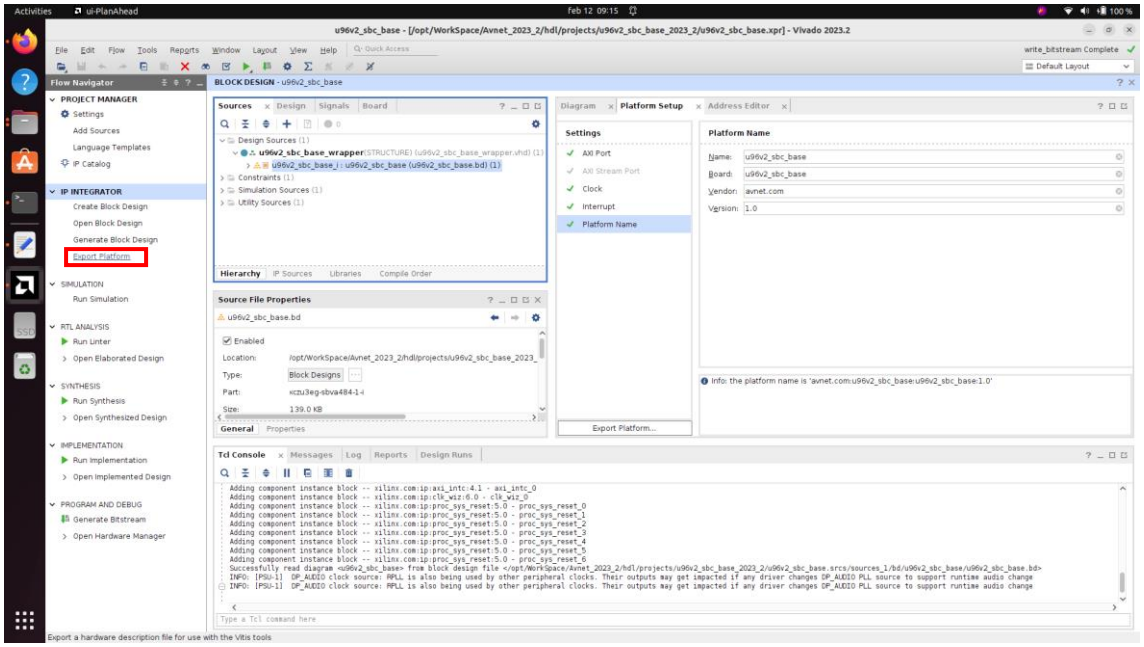

Figura 19. Ubicación de Export Platform.

Al iniciar el proceso de exportación, seleccionamos **Pre-Synthesis** como tipo de exportación, ya que el archivo se usará en Vitis. Es importante añadir también el bitstream, para que Vitis no tenga que volver a generarlo más tarde.

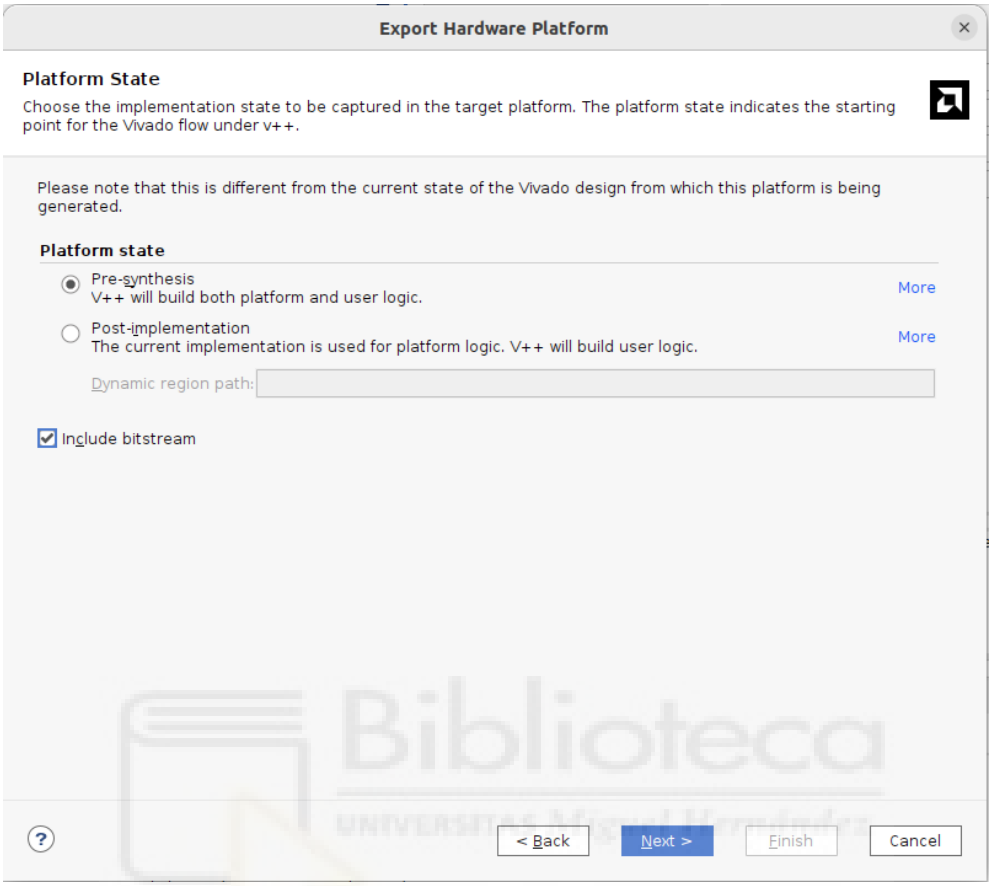

Figura 20. Configuración de Export Platform.

A continuación, indicamos el nombre de la plataforma, el nombre del archivo XSA y la ruta donde se guardará. Yo he optado por u96v2_avnet.xsa.

Guardamos el archivo en una ubicación de fácil acceso para facilitar su uso en PetaLinux y luego hacemos clic en **Exportar**.

9. **Verificación de la Exportación**

Podemos verificar si la exportación se ha realizado correctamente comprobando si el archivo XSA ha aparecido en la ubicación seleccionada. Esto indicará que el modelo ha sido correctamente generado y está listo para ser utilizado en otras herramientas de Xilinx.

Figura 21. Archivo XSA exportado.

4.1.9. Generación del proyecto de Petalinux y modificaciones necesarias de meta-avnet. Obtención de la imagen base

Antes de empezar a generar el proyecto tuvimos que valorar lo siguiente: si usáramos la imagen generada por el script de Avnet y la modificaríamos para añadir los paquetes que faltan o crearíamos el proyecto desde cero, como hace José María. Optamos por evitar usar el script de Avnet dado que este añadiría muchas librerías que harían la imagen más pesada y, además, podría no haber compatibilidad con las herramientas que instalaríamos. Así pues, comenzamos creando un directorio en `/opt/WorkSpace/` llamado `Petalinux`, donde se creará el proyecto.

Para la creación del proyecto, usamos las herramientas y generamos un proyecto utilizando el template correspondiente a nuestro dispositivo. El comando utilizado fue:

```
source /opt/xilinx/Petalinux/2023.2/settings.sh
petalinux-create --type project --template zynqMP --name u96v2_avnet
```

En el proyecto creado, agregamos la carpeta `meta-avnet` dentro de `u96v2_avnet/project-spec/`. Esta carpeta puede copiarse desde una versión descargada previamente o descargarse de nuevo; lo importante es que esté presente antes de importar el archivo `.xsa`. Una vez hecho esto, se procede a importar la descripción de hardware con el siguiente comando:

```
petalinux-config --get-hw-description=/opt/WorkSpace/Avnet_2023_2/hdl/projects/u96v2_sbc_base_2023_2/u96v2_avnet.xsa
```

Al ejecutar el comando anterior, se abre la pantalla básica de configuración del proyecto, donde realizamos las siguientes modificaciones:

1. **Agregar la carpeta meta-avnet**:

   En "Yocto settings", dentro de "User layers", agregamos la capa `${PROOT}/project-spec/meta-avnet`.

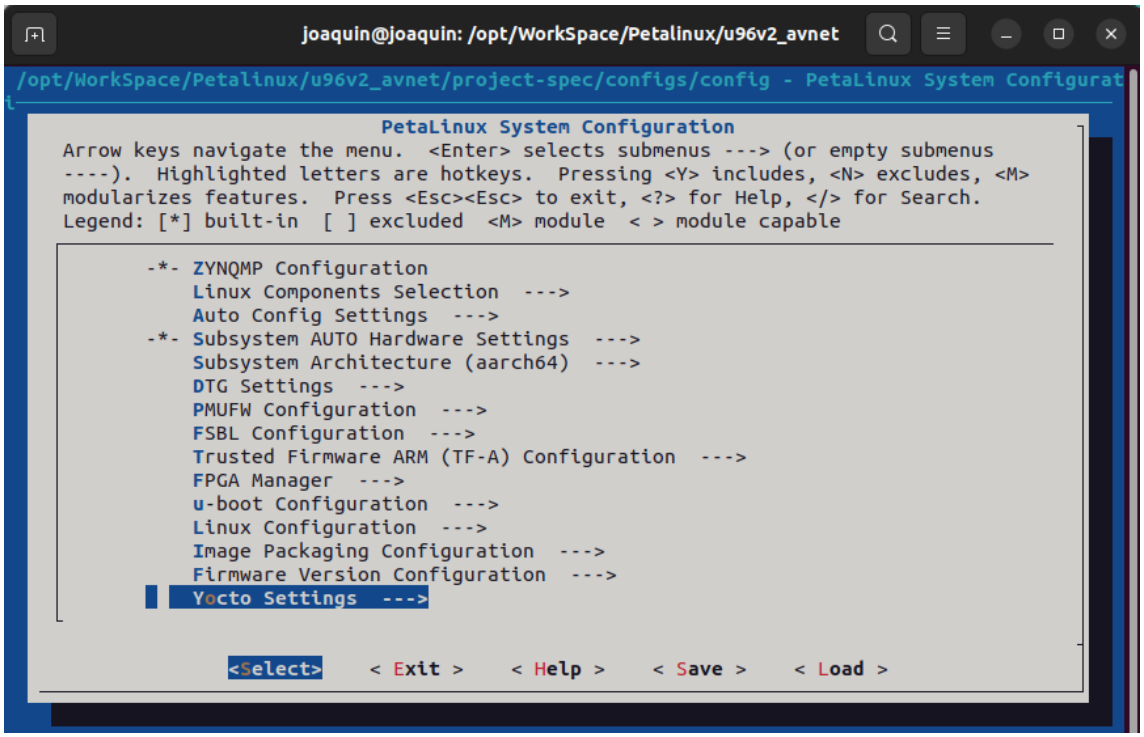

Figura 22. Panel de petalinux-config.

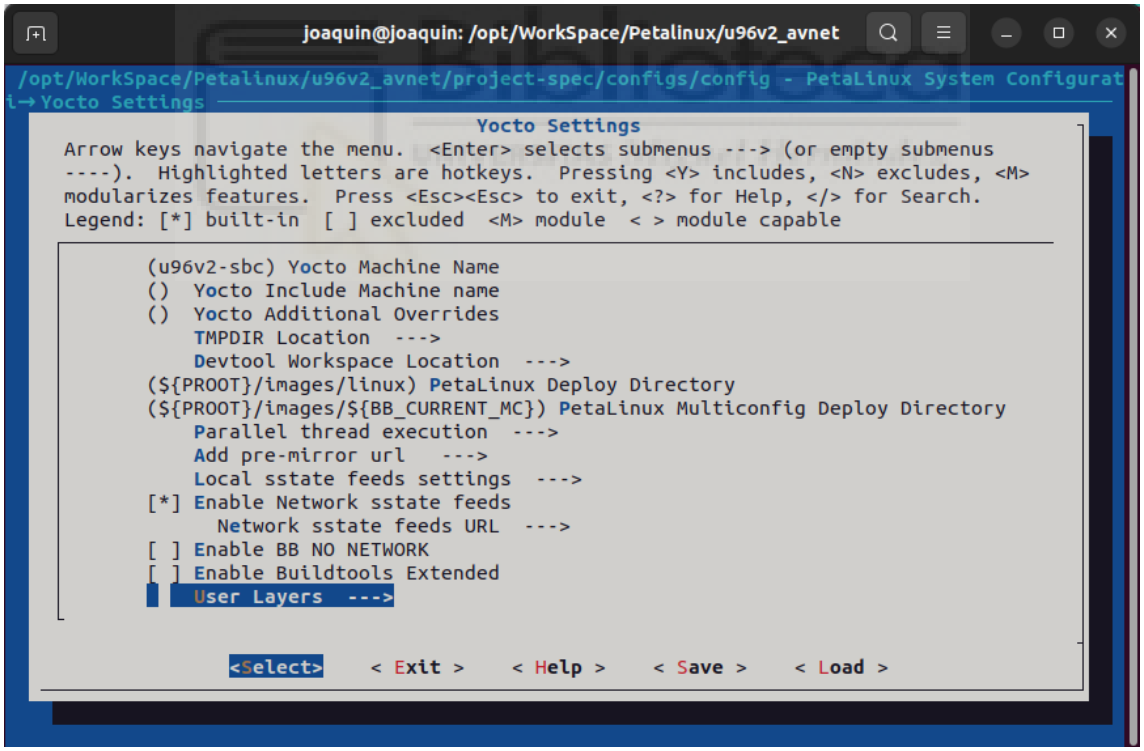

Figura 23. Sección de Yocto Settings.

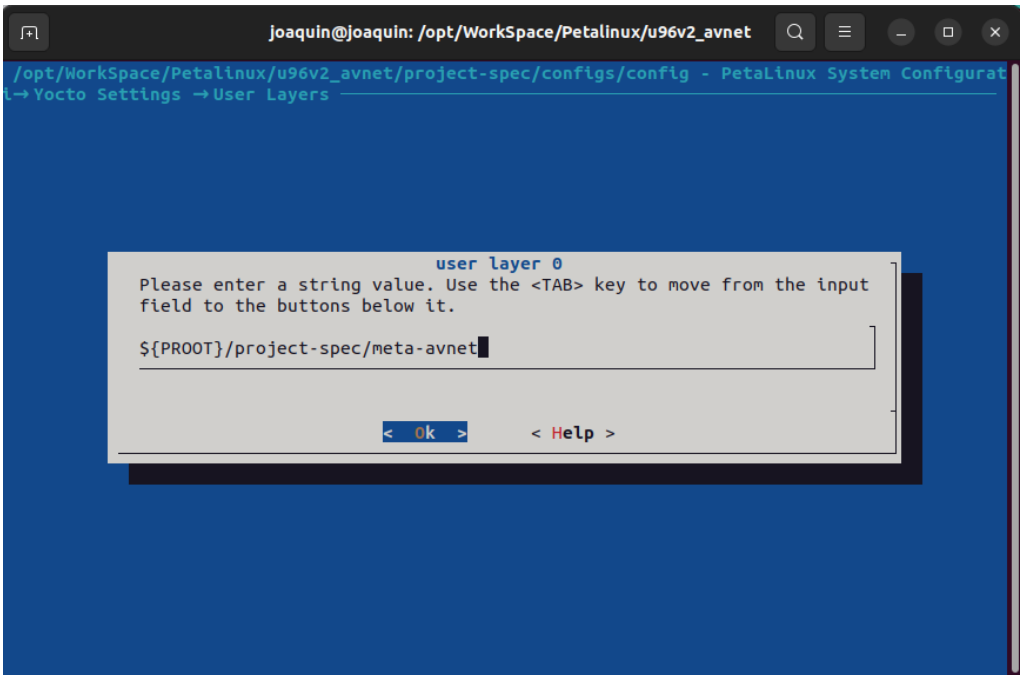

Figura 24. Configuración de la capa de usuario en Yocto Settings→user layers.

2. **Cambiar el nombre del proyecto**:

   Cambiamos el nombre a `u96v2-sbc`.

3. **Configurar formatos de empaquetado**:

   En "Image Packaging Configuration", seleccionamos EXT4 como tipo de empaquetado para el sistema de archivos raíz. Eliminamos todos los formatos excepto `ext4` y, por seguridad, dejamos también `tar.gz`. Esto reducirá el tiempo necesario para generar la imagen.

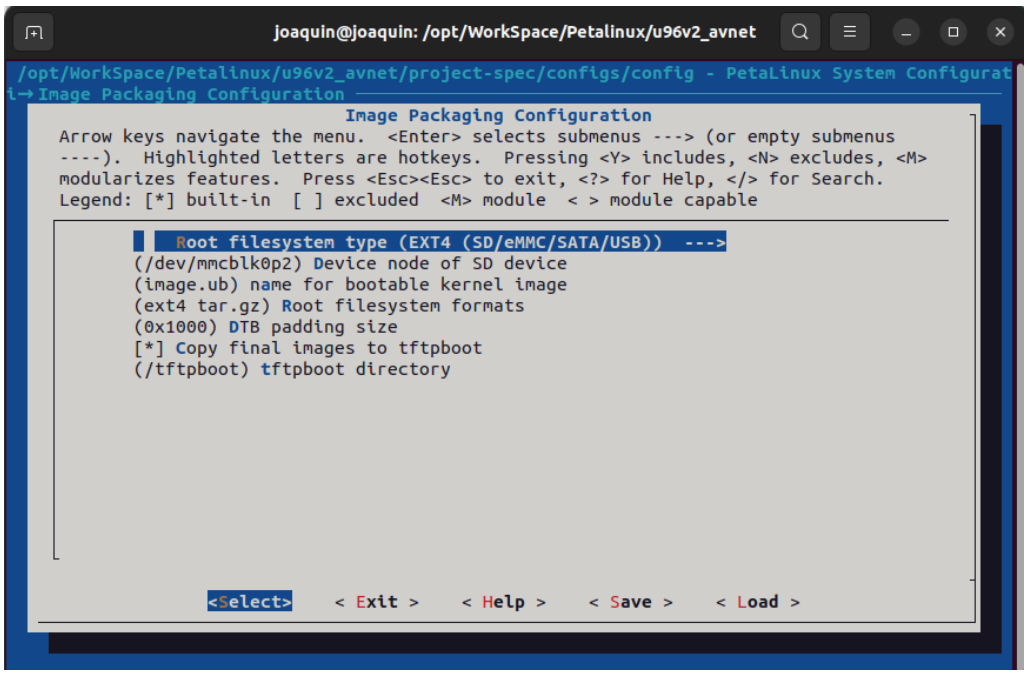

Figura 25. Configuración de Image Packaging Configuration.

4. **Configurar la conexión serie**:

   Configuramos la conexión en "Subsystem AUTO Hardware Settings → Serial Settings" para usar solo el adaptador UART conectado en la conexión 1, con una velocidad de 115200 bps.

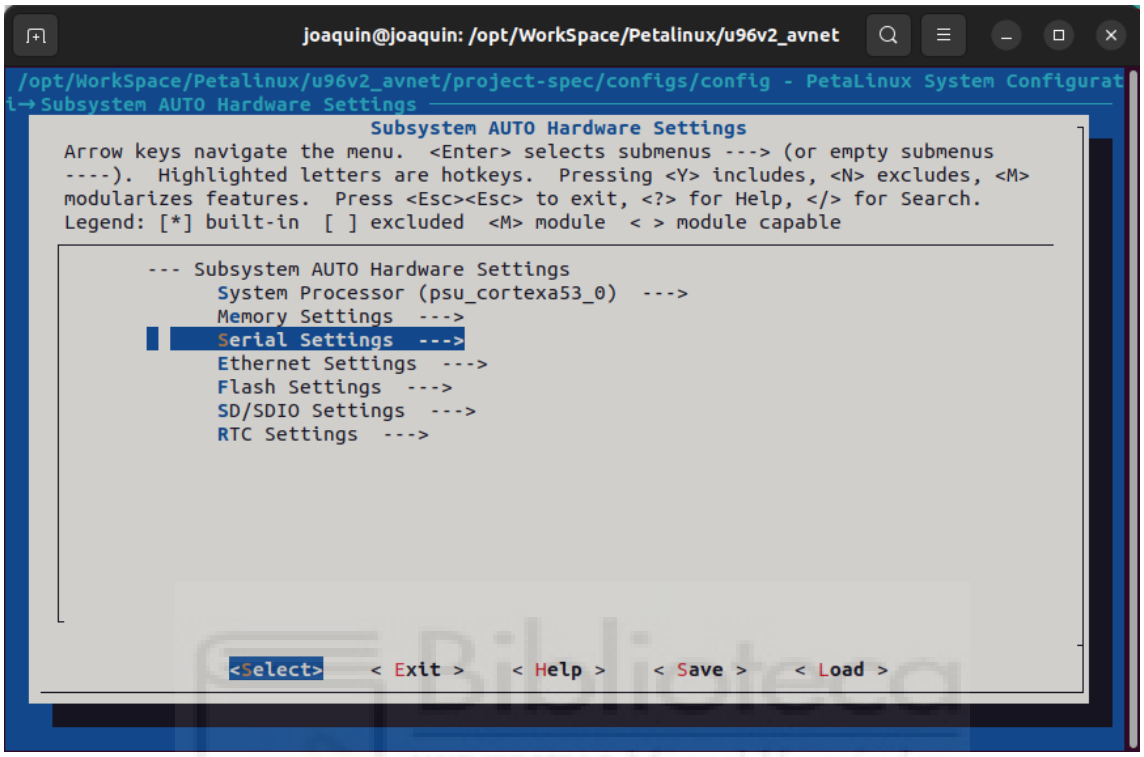

Figura 26. Acceso a Serial Settings en Subsystem AUTO Hardware Settings.

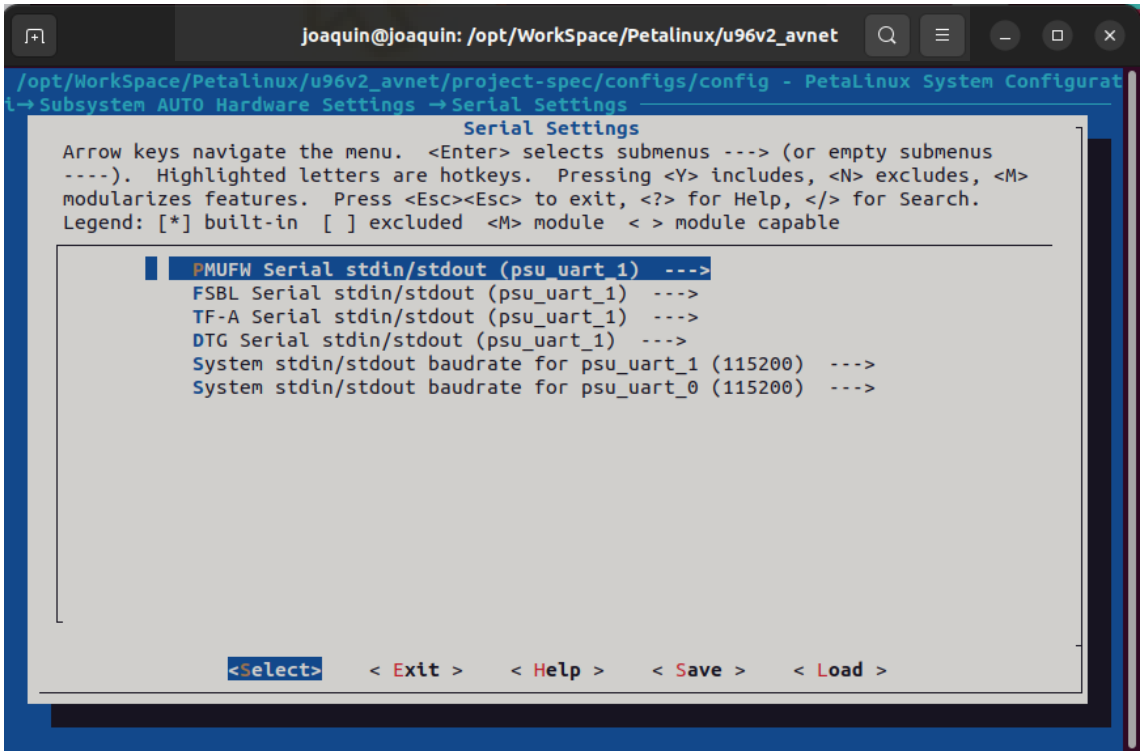

Figura 27. Configuración de Serial Settings.

Guardamos los cambios y salimos de esta configuración.

A continuación, accedemos a la configuración del kernel ejecutando el siguiente comando:

```
petalinux-config -c kernel
```

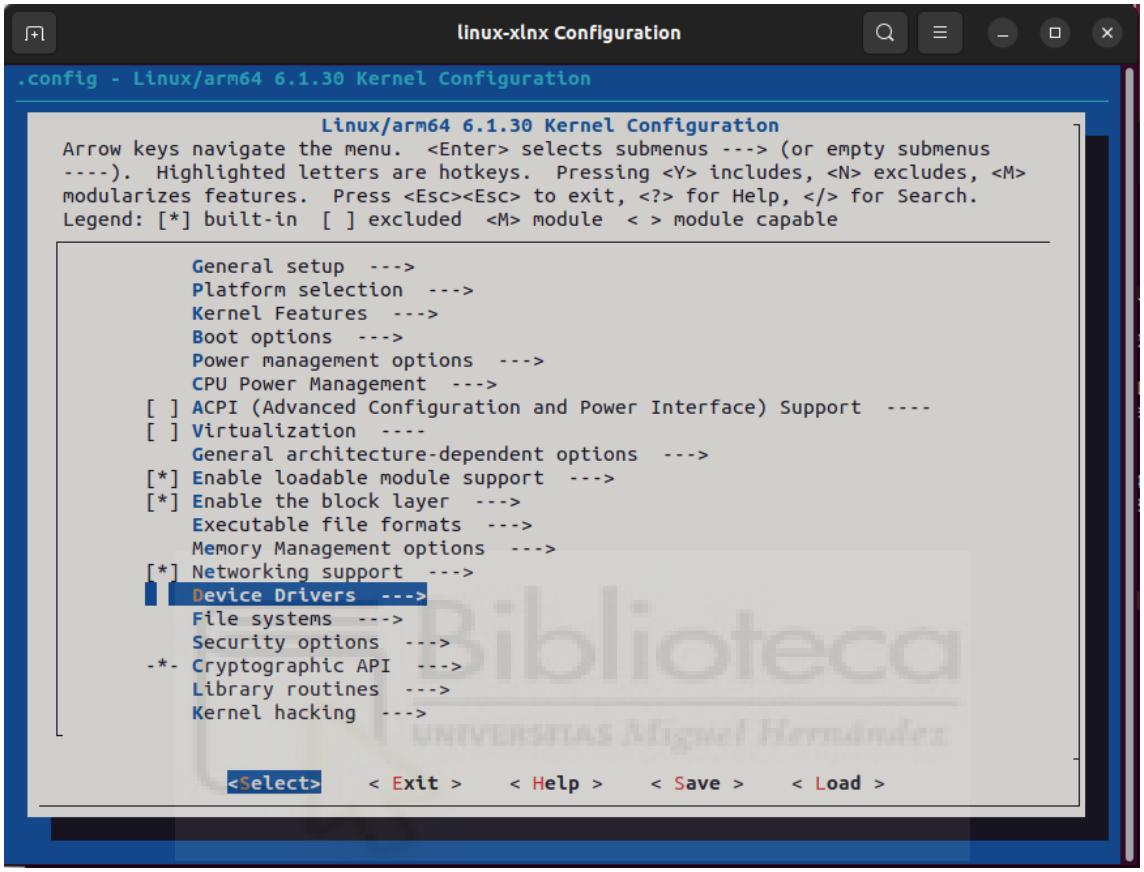

Figura 28. Panel de petalinux-config -c kernel.

En esta pantalla realizamos las siguientes configuraciones:

1. **Drivers para la DPU**:

   Activamos el driver de la DPU en "Device Drivers -> Misc Devices -> <*> Xilinx Deep Learning Processing Unit (DPU) Driver".

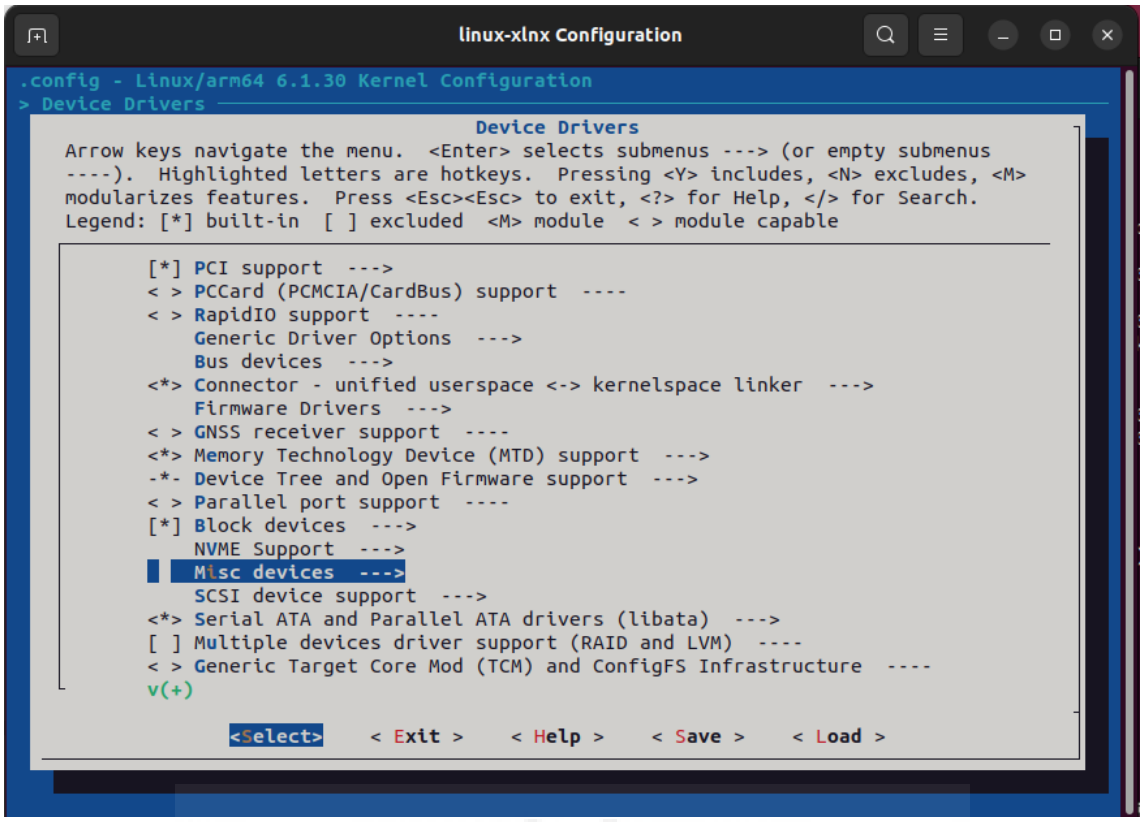

Figura 29. Ubicación de Misc devices.

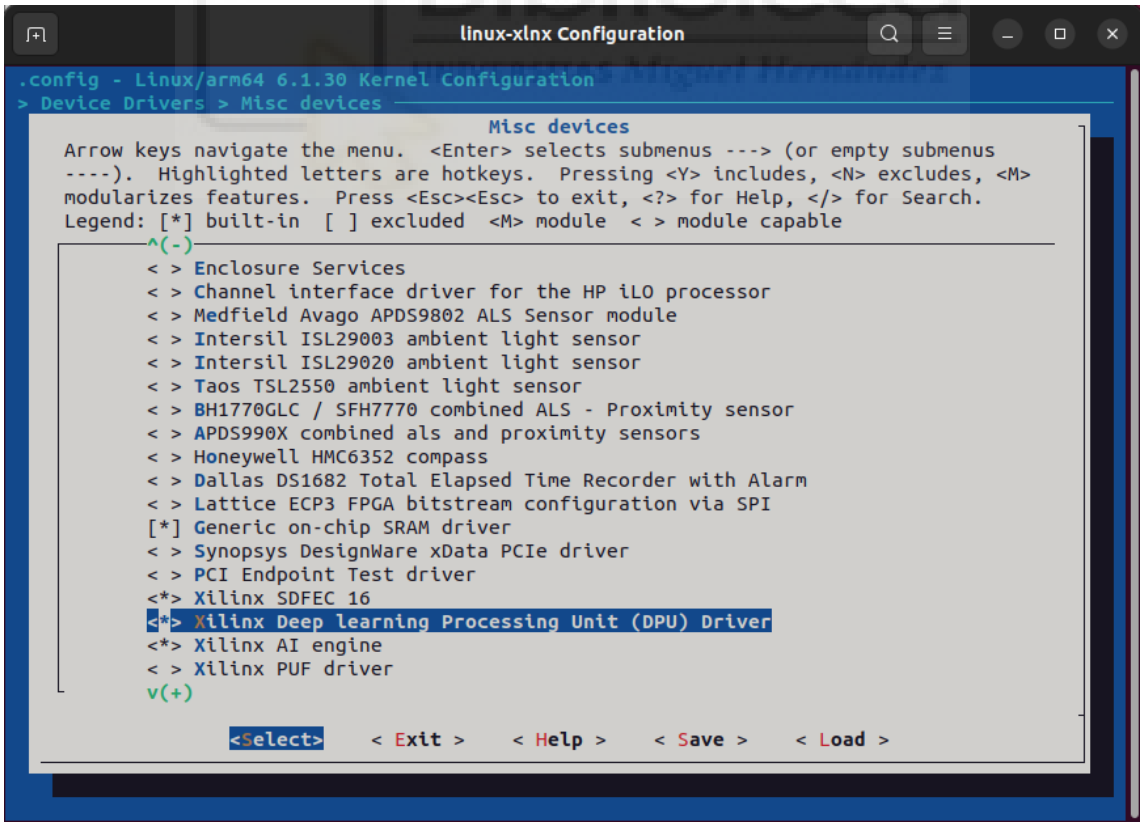

Figura 30. Configuración del controlador de la DPU.

2. **Aumentar el espacio contiguo de memoria DMA**:

   En "Library routines", ajustamos el tamaño a 1024 MB, repartiendo la memoria del dispositivo al 50/50 entre buffers y otros procesos.

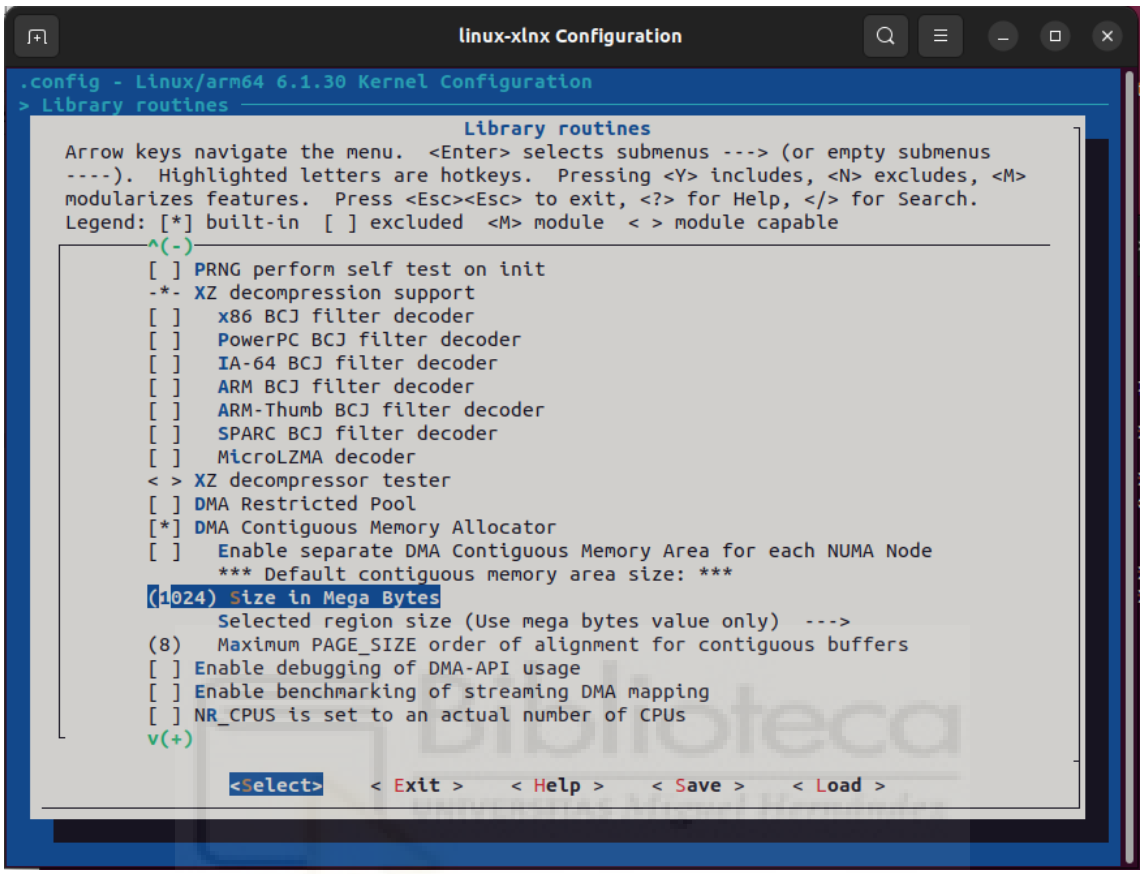

Figura 31. Configuración de la memoria contigua DMA en Library Routines.

3. **Desactivar la gestión de consumo de la CPU**:

   Desactivamos las opciones en "CPU Power Management", tanto para CPU idle como para CPU frequency scaling.

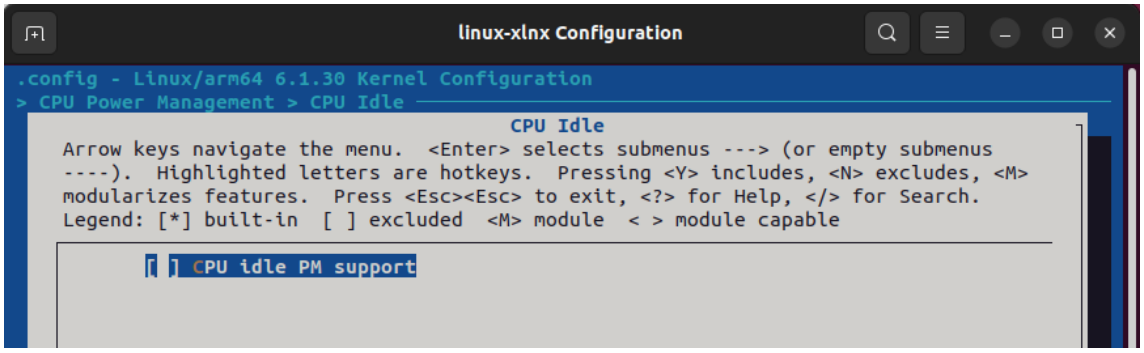

Figura 32. Configuración de CPU idle PM support.

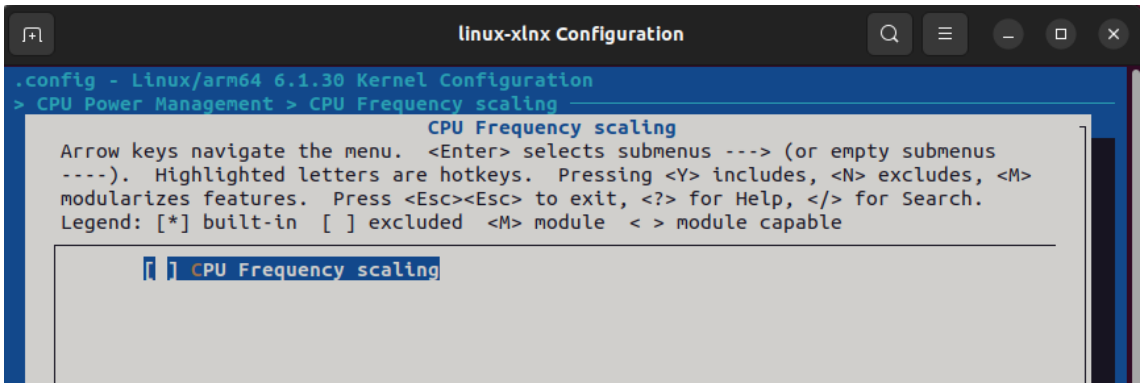

Figura 33. Configuración de CPU Frequency Scaling.

Guardamos los cambios y salimos.

Tras configurar el kernel, procedemos a modificar manualmente los archivos del proyecto Petalinux.

- **Corrección en meta-avnet**:

  Eliminamos `u96v2-sbc-factest` para que se detecte correctamente `u96v2-sbc`, esto es un error concreto del momento que la descargué que ya ha sido subsanado, pero consistía en modificar el archivo de la siguiente ruta `meta-avnet/recipes-bsp/device-tree/device-tree.bbappend`.

- **Agregar librerías necesarias**:

  Editamos el archivo `project-spec/meta-user/conf/user-rootfsconfig` y añadimos la siguiente lista de paquetes del Anexo 1. User-rootfsconfig.

- **Modificar archivos del proyecto**:

  - Comentamos las líneas 104-123 en `project-spec/meta-avnet/recipes-bsp/device-tree/files/u96v2-sbc/system-bsp.dtsi`.
  - Eliminamos referencias a `ultra96-ap-setup`, `ssh-server-dropbear` y `packagegroup-core-ssh-dropbear` en los siguientes archivos:
    - `avnet-image-full.inc`
    - `avnet-image-minimal.inc`
    - `petalinux-image-minimal.bbappend`

Para configurar el sistema de archivos raíz y agregar las librerías, ejecutamos:

```
petalinux-config -c rootfs
```

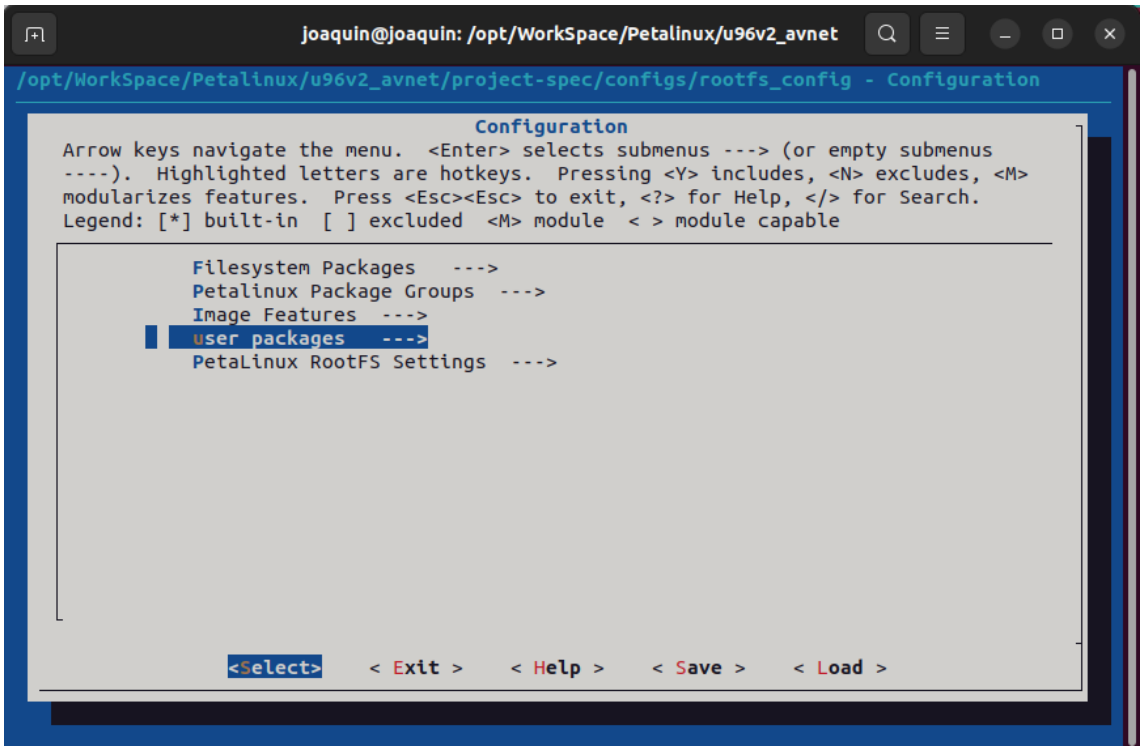

Figura 34. Panel petalinux-config -c rootfs.

Activamos todos los paquetes de `user-rootfsconfig` que estarán presentes en "user packages"y habilitamos las opciones en `Petalinux Package Group → packagegroup-petalinux-vitis-acceleration-essential`.

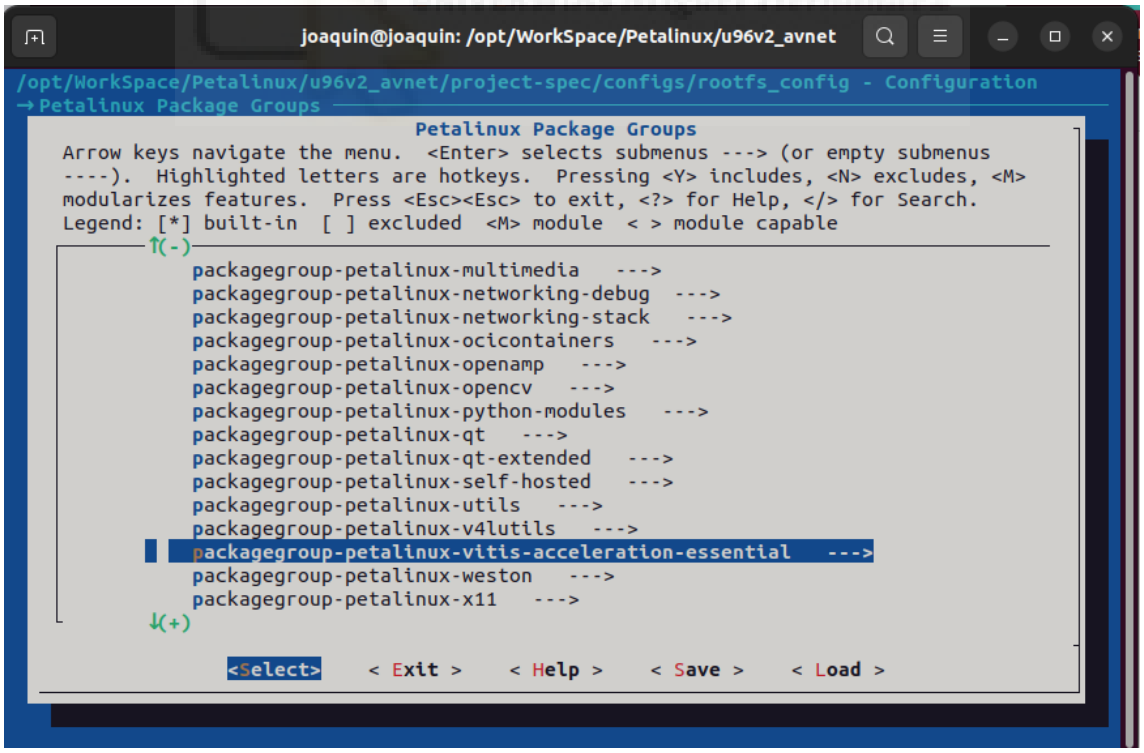

Figura 35. Paquete para aceleradores de hardware de vitis.

Finalmente, generamos la imagen:

```
petalinux-build -c avnet-image-full
```

Puede ser necesario repetir este paso debido a posibles errores en la descarga de paquetes por fluctuaciones en la conexión a internet.

Con la imagen lista, generamos los archivos necesarios para Vitis:

```
petalinux-package --boot --u-boot --fpga images/linux/system.bit --force
petalinux-package --prebuilt --fpga images/linux/system.bit --force
```

Por último, guardamos el proyecto para futuras modificaciones:

```
petalinux-package --bsp -p u96v2_avnet --hwsource /opt/WorkSpace/Avnet_2023_2/hdl/projects/u96v2_sbc_base_2023_2/u96v2_avnet.xsa --output u96v2_avnet.bsp
```

4.1.10. Uso de BalenaEtcher para quemar la imagen en una SD

Como indicamos anteriormente en el capítulo de Software, **BalenaEtcher** no requiere instalación; basta con ejecutar su archivo **AppImage**. Para ello, navegamos hasta la carpeta donde está guardado y ejecutamos el archivo desde la terminal con el siguiente comando:

```
./balenaEtcher-1.18.11-x64.AppImage
```

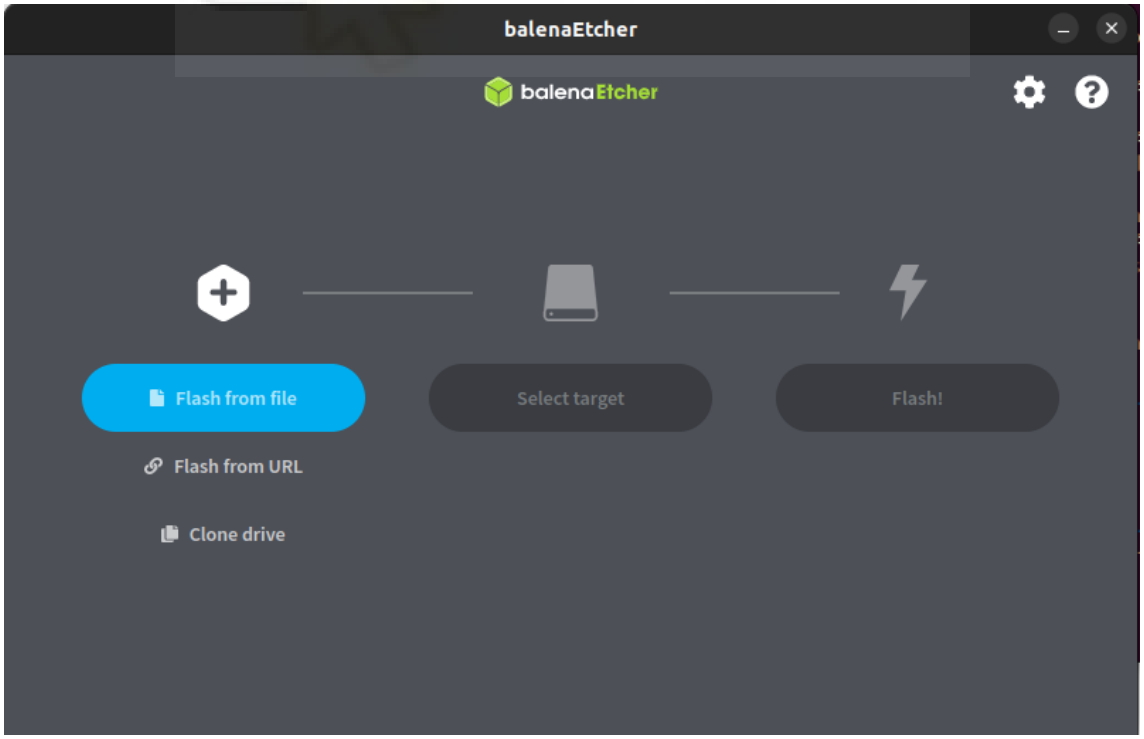

Figura 36. Panel de BalenaEtcher.

Una vez abierto BalenaEtcher, seguimos estos pasos:

1. **Seleccionar la imagen**

   Localizamos la imagen generada por Petalinux. Esta imagen puede estar guardada en el directorio del proyecto Petalinux (generalmente dentro de `images/linux/`) o en una ubicación más accesible. Para seleccionarla, hacemos clic en **'Select Image'** en la interfaz de BalenaEtcher y utilizamos el explorador de archivos que se abre para navegar hasta la imagen correspondiente.

2. **Conectar y seleccionar la tarjeta microSD**

   Insertamos la tarjeta microSD en el ordenador. A continuación, en BalenaEtcher, hacemos clic en **'Select Target'** para elegir el dispositivo adecuado. Es fundamental asegurarnos de seleccionar la microSD correcta para evitar sobrescribir otros dispositivos de almacenamiento.

3. **Quemar la imagen**

   Una vez seleccionada la imagen y el dispositivo, hacemos clic en **'Flash!'**. BalenaEtcher comenzará el proceso de quemar la imagen en la microSD y, al finalizar, realizará automáticamente una verificación para asegurarse de que la imagen se haya grabado correctamente.

4. **Verificar las particiones**

   Una vez completado el proceso, podemos comprobar en el explorador de archivos que las particiones dentro de la microSD se han generado correctamente. Esto confirmará que la imagen ha sido grabada de manera exitosa.

Con la microSD preparada, procedemos a insertarla en el dispositivo Ultra96-V2 y lo encendemos. El sistema debería iniciar utilizando la configuración almacenada en la microSD. En este paso, verificamos que el dispositivo arranque correctamente y que todos los componentes funcionen según lo esperado.

Si todo ha sido realizado correctamente, el dispositivo estará listo para ser configurado y utilizado en los siguientes pasos del proyecto.

4.1.11. Creación del proyecto usando Vitis classic, pero compilando desde el exterior usando cmake para evitar los defectos de Vitis de la nueva versión. Obtención de la aplicación (solo modifica en la imagen el RootFileSystem).

En este apartado, explicaremos cómo crear un proyecto en **Vitis Classic** utilizando la compilación externa con **CMake** para evitar los defectos encontrados en la versión más reciente de Vitis (2023.2). A continuación, se detallan los pasos necesarios para configurar y obtener una aplicación funcional, modificando únicamente el **RootFileSystem** en la imagen generada.

4.1.11.1. Preparación del Entorno y Creación del Proyecto

1. **Configurar el Entorno de Vitis**

   Creamos una carpeta en nuestro espacio de trabajo (Workspace) para almacenar los archivos generados por Vitis. Por ejemplo, llamaremos a esta carpeta `DPU`.

Abrimos un terminal, cargamos el entorno de Vitis y ejecutamos Vitis Classic con los siguientes comandos:

```
source /opt/xilinx/Vitis/2023.2/settings64.sh
vitis -classic
```

2. **Crear el Proyecto de Plataforma**

   Seleccionamos la carpeta creada (`DPU`) como Workspace. En la pantalla inicial de Vitis, seleccionamos la opción de crear un proyecto de **Plataforma**. Seguimos los pasos:

   - Asignamos un nombre al proyecto.
   - Indicamos el archivo `.xsa` generado previamente.
   - Configuramos:
     - Sistema operativo: **Linux**.
     - Procesador: **psu_cortexa53**.
     - Arquitectura: **64 bits**.
     - Deshabilitamos la generación de componentes de arranque.

2. **Configuración del Dominio del Sistema Operativo**

   Una vez generada la plataforma, en el explorador de Vitis, navegamos hasta **linux on psu_cortexa53** para editar las descripciones del dominio.

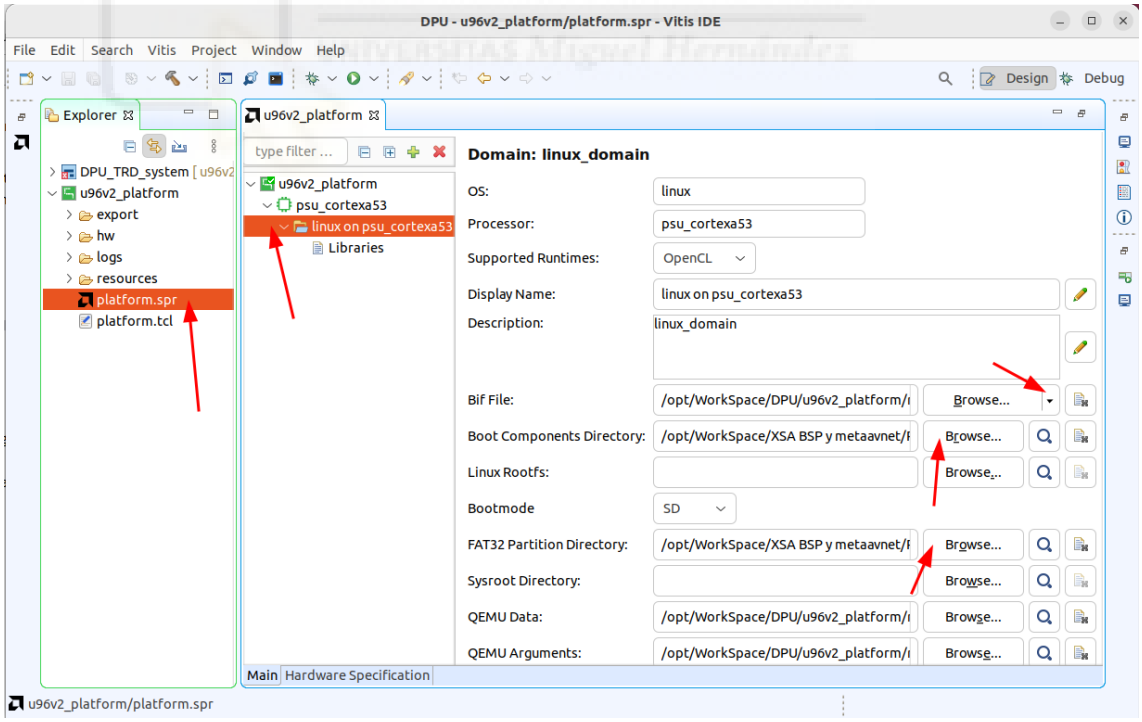

Figura 37. Configuración de la plataforma en Vitis Classic 2023.2.

- **Generar BIF File**: Configuramos para que se genere automáticamente a partir del archivo `.xsa`.

- **Archivos de Arranque**: Creamos una carpeta llamada `boot` donde almacenamos los siguientes archivos:
    - `bl31.elf`
    - `pmufw.elf`
    - `system.dtb`
    - `u-boot.elf`
    - `zynqmp_fsbl.elf`
- Otra carpeta, llamada `sd_dir`, almacenará:
    - `boot.scr`
    - `system.dtb`
- **Configuramos en Vitis**:
    - Carpeta de arranque (Boot Components Directory): `boot`.
    - Carpeta de la partición FAT32 (FAT32 Partition Directory): `sd_dir`.

3. **Construcción Inicial de la Plataforma**

   Para generar la plataforma completa con un sistema operativo Linux, hacemos clic derecho en el proyecto principal y seleccionamos **Build Project**. Esto genera una plataforma lista para cargar la aplicación DPU.

4.1.11.2. Preparación de la Aplicación DPU

Antes de cargar el template de la DPU en Vitis necesitamos hacer

1. **Generamos un repositorio de archivos raíz específico para la DPU**

    - Localizamos y descargamos el script `host_cross_compiler_setup.sh` ubicado en el github de Vitis-AI 3.0, estará en el apartado board_setup/mpsoc/.
    - Modificamos el script para que nos genere un directorio con los archivos raíz en una ruta general más cómoda y accesible, como en el WorkSpace, y ejecutamos el script.
    - Cerramos Vitis, y antes de volver a llamarlo ejecutamos los siguientes comandos:

```
unset LD_LIBRARY_PATH
source /opt/WorkSpace/petalinux_sdk_2022.2/environment-setup-cortexa72-cortexa53-xilinx-linux
```

2. **Descargar la librería específica de la DPU para nuestro dispositivo y configurarla**

    - En el github de Vitis-AI, en la rama 3.0 (https://github.com/Xilinx/Vitis-AI/tree/3.0/dpu) descargaremos el Diseño de referencia para MPSoC DPUCZDX8G. Tras finalizar la descarga lo descomprimiremos en el WorkSpace para que quede accesible.
    - En la barra de herramientas de Vitis, en el apartado Window > Preferences buscaremos Library Repositories y añadiremos la configuración de la DPU creando un nuevo repositorio al que podemos llamar DPU, y lo más importante, indicar la ruta a la carpeta que hemos extraído en el WorkSpace.

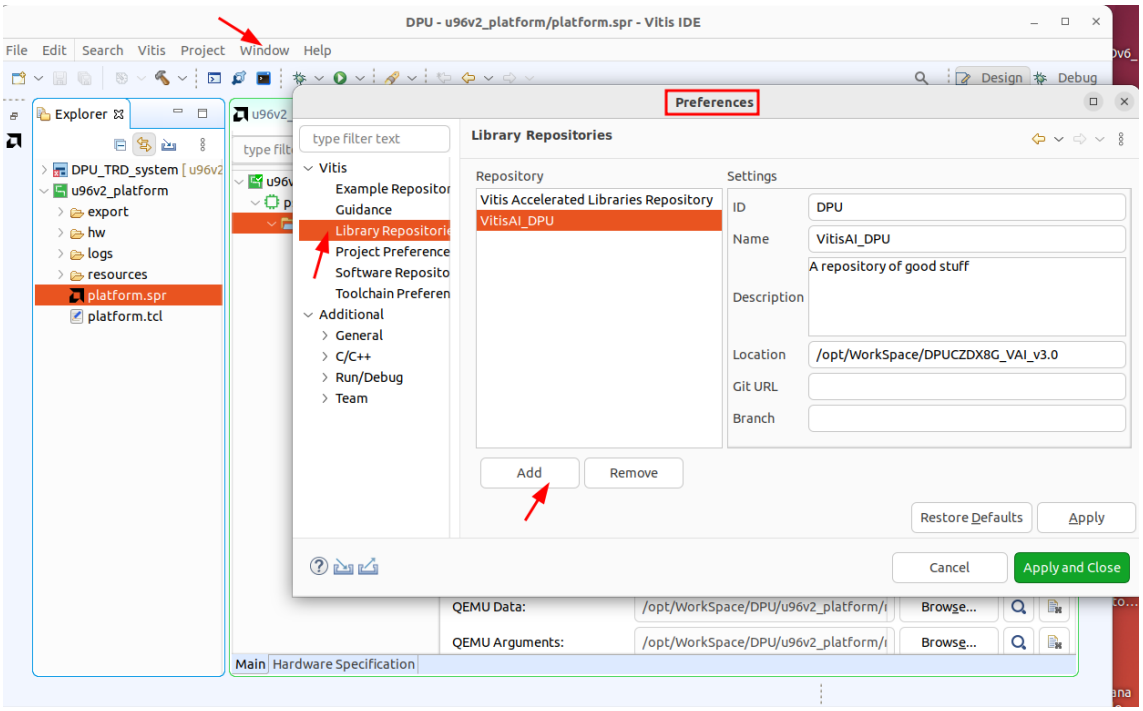

Figura 38. Panel de Windows→Preferences con la librería de la DPU.

4.1.11.3. Creación de la Aplicación y su Configuración

1. **Creación de la Aplicación**

- Haciendo click en File>New>Application project de la barra de herramientas se nos abrirá una ventana para configurar nuestra aplicación.

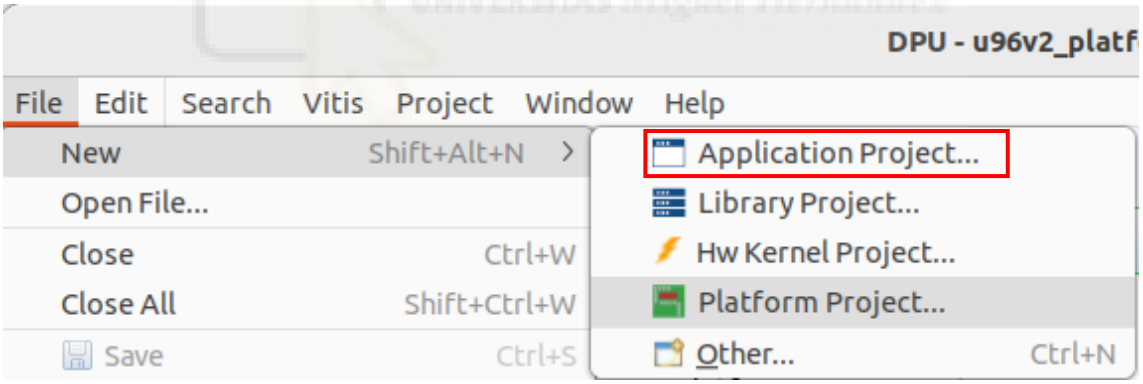

Figura 39. Crear el proyecto de aplicación.

- Después de un panel de introducción seleccionaremos nuestra plataforma.
- Nos pedirá que elijamos un nombre para nuestro proyecto, y dado que estamos siguiendo el flujo de trabajo de Vitis-AI de integración por Vitis, el nombre que usaremos será DPU_TRD. También podemos confirmar que el procesador elegido es el de mayor tamaño del chip, en este caso el psu_cortexa53.
- En el siguiente panel, en 'configuración de la aplicación', en la ruta al Sysroot, indicaremos la carpeta creada en el WorkSpace por el script de host_cross_compiler.sh,

`/opt/WorkSpace/petalinux_sdk_2022.2/sysroots/cortexa72-cortexa53-xilinx-linux`. En cuanto a Root FS y la Imagen del Kernel, podemos usar la de nuestro proyecto o no usar ninguna, los archivos estarán en la carpeta dentro de nuestro proyecto de Petalinux `images/linux/`.

- Dentro de la lista de Templates, elegiremos DPU Kernel (RTL Kernel) y le daremos a finalizar.

2. **Configurar los ajustes del proyecto**

- Active Build Configuration → Hardware.
- En la ventana de Explorer a la izquierda, en DPU_TRD_system> DPU_TRD_kernels> src> prj> Vitis> dpu_conf.vh modificaremos las dimensiones de la unidad DPU y el uso de multiplicadores como bajo. Anexo 2. Dpu_conf.vh

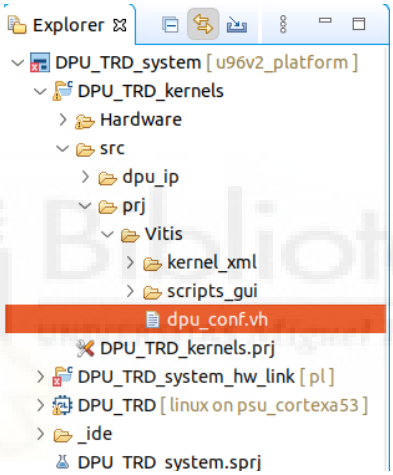

Figura 40. Ubicación de dpu_conf.vh en el Explorer.

- En la pestaña de explorer, En DPU_TRD_system_hw_link> DPU_TRD_system_hw_link.prj> Hardware Functions, seleccionamos sfm_xrt_top y la eliminamos, luego, el número de unidades DPU lo reducimos a 1.

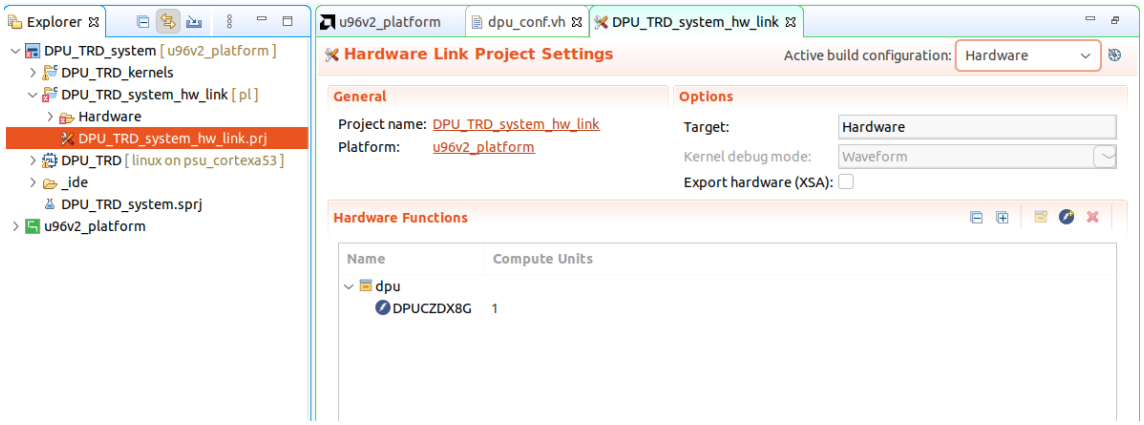

Figura 41. Configuración de DPU_TRD_system_hw_link.prj.

- A continuación, vamos a configurar el uso de las librerías para la compilación C++ del proyecto. Para ello haciendo click derecho sobre DPU_TRD [linux on psu_cortexa53] seleccionaremos 'C/C++ Build Settings'.

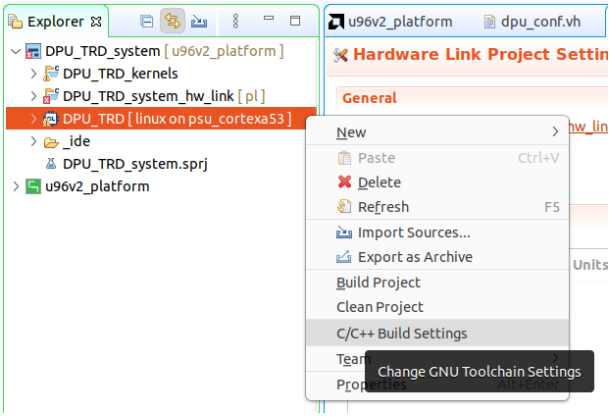

Figura 42. Acceso a la configuración del compilador C++/C.

- En la pestaña de la izquierda seleccionamos Settings, en la central Libraries y en la inferior derecha seleccionamos el icono para agregar una nueva ruta de librería. En el panel seleccionamos File system y navegamos en el explorador hasta /opt/WorkSpace/DPUCZDX8G_VAI_v3.0/app/samples/lib, después de seleccionarlo lo agregamos y desde el panel anterior, subimos la preferencia a la ruta recién agregada.

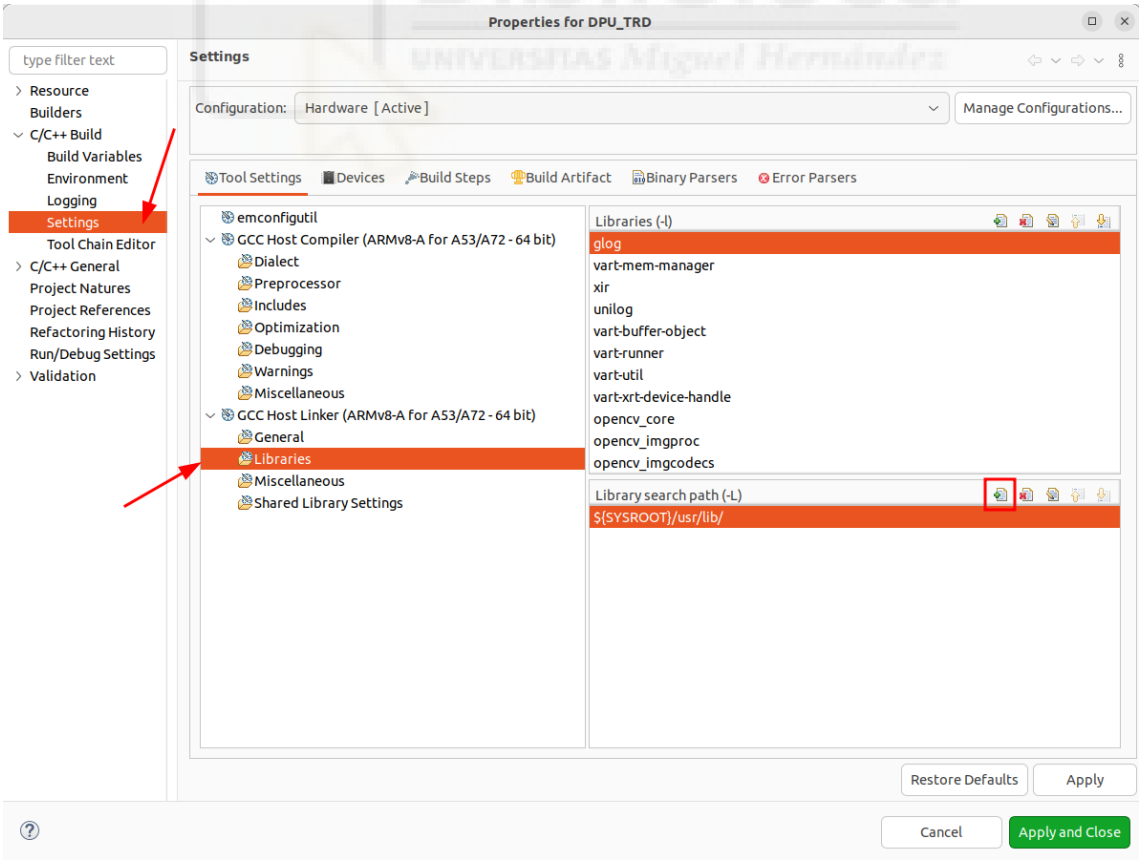

Figura 43. Panel de configuración del compilador C++/C.

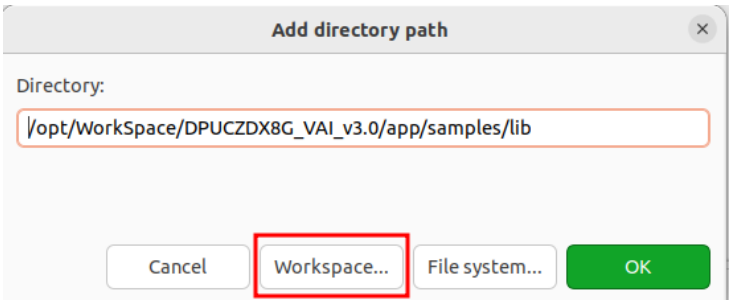

Figura 44. Enrutado de los archivos del compilador de la DPU.

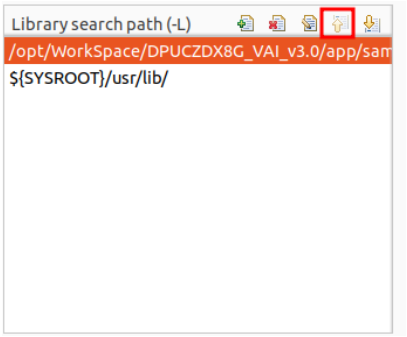

Figura 45. Cambiando la prioridad a las librerías de la DPU.

- Para configurar el último archivo necesitamos que Vitis lo genere y así podremos modificarlo desde el exterior de la aplicación, para ello provocaremos una construcción fallida del proyecto para que genere los archivos dentro de la carpeta DPU_TRD_system_hw_link/Hardware/. Para hacerlo, desde el panel Explorer, seleccionamos la carpeta de más alto rango de la aplicación, DPU_TRD_system [u96v2_platform], y en la barra de herramientas seleccionamos el martillo.

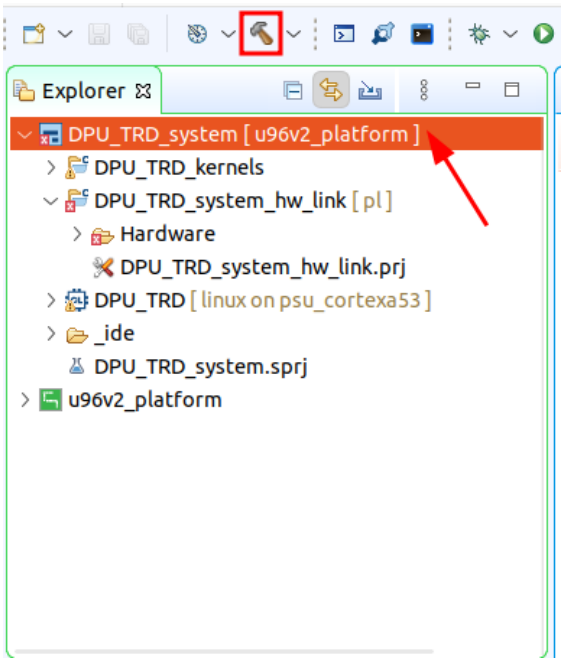

Figura 46. Construcción preliminar del proyecto de aplicación.

- Finalmente, tras la construcción, podemos cerrar Vitis. Desde la misma terminal que hemos usado para ejecutar Vitis navegaremos hasta /opt/WorkSpace/DPU/DPU_TRD_system_hw_link/Hardware y abriremos dpu-ling.cfg y dentro agregaremos el Anexo 3. Dpu-link.cfg y guardaremos

```
cd /opt/WorkSpace/DPU/DPU_TRD_system_hw_link/Hardware
gedit dpu-link.cfg
```

- **Creación de la aplicación**

  - Desde esta misma carpeta ejecutaremos el siguiente comando para la creación completa de la aplicación y modificación de la imagen. Este proceso toma algo de tiempo así que se recomienda tenerlo en cuenta. Lanzaremos el siguiente comando, indicando después de jobs el número de procesos simultáneos que quieres llevar a cabo, en mi caso por limitaciones de hardware lo he dejado en 4:

```
make all --jobs 4
```

  - Cuando finalice tendremos la imagen en DPU_TRD_system/Hardware/package/sd_card.img.

4.1.12. Actualización de imagen generada de Petalinux usando el software de reparación nativo de Linux. Obtención de la imagen definitiva.

En esta etapa del desarrollo, contamos con dos imágenes funcionales:

1. Una preparada específicamente para el chip con todos los detalles necesarios.
2. Otra generada a partir de un **sysroot genérico**, diseñada para permitir el proceso de **cross compiling**.

Aunque podríamos quemar directamente la nueva imagen en la tarjeta SD, optamos por actualizar la imagen original ya existente. Esto asegura que mantengamos el estado base del producto y sólo incorporemos los cambios relacionados con el nuevo **sysroot** y la integración de la aplicación DPU.

Para llevar a cabo esta tarea, empleamos la aplicación nativa de Linux Ubuntu llamada **Disks**. Este software facilita la gestión y reparación de particiones en unidades de memoria externas. A continuación, se describen los pasos seguidos:

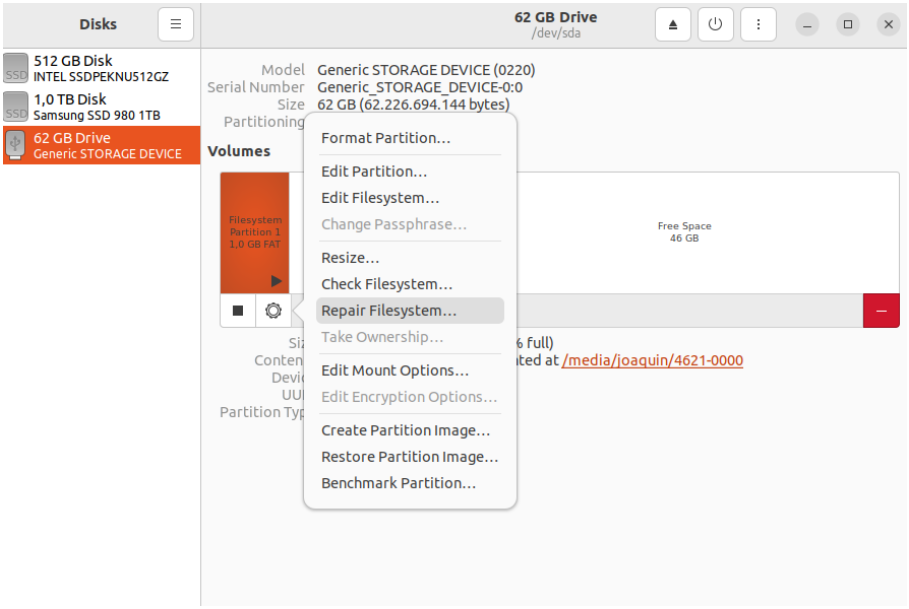

Figura 47. Ubicación de la opción de reparación del gestor de archivos.

1. **Acceso a la Partición FAT32**
    - Con la tarjeta SD insertada, abrimos la herramienta **Disks**.
    - Seleccionamos la partición **FAT32** correspondiente a la imagen base.
2. **Reparación de la Partición**
    - En la parte inferior de la interfaz, hacemos clic en el icono de la **tuerca**.
    - Elegimos la opción **"Reparar sistema de archivos"**.
    - Seleccionamos la imagen generada previamente por **Vitis**, asegurándonos de que incluya tanto el **nuevo sysroot** como la aplicación DPU.
3. **Proceso de Actualización**
    - La herramienta ejecutará automáticamente el proceso, actualizando únicamente los elementos mencionados sin modificar el resto de la estructura de la imagen.

Después de verificar que la imagen actualizada funciona correctamente, procedimos a extraerla de la tarjeta SD y guardarla en un lugar seguro para preservar el estado actualizado. Los pasos realizados fueron:

1. Abrimos nuevamente la aplicación **Disks**.
2. En el panel izquierdo, seleccionamos la tarjeta SD.
3. Hacemos clic en los tres puntos ubicados en el panel derecho y elegimos la opción **"Create Disk Image..."**.

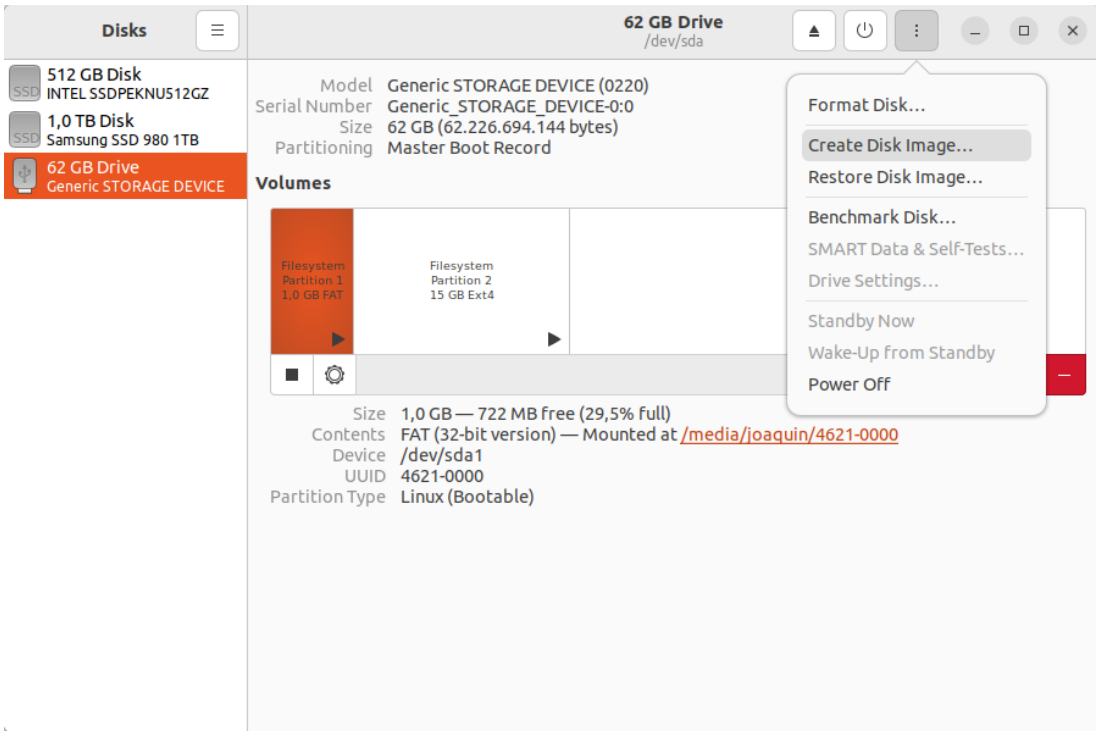

Figura 48. Ubicación del creador de la imagen de disco.

4. Especificamos:
    - **Nombre del archivo:** `U96v2_2023_2_DPU`
    - **Ubicación de almacenamiento:** una carpeta destinada a imágenes finales del proyecto, yo la dejé provisionalmente en el WorkSpace.
5. Finalmente, presionamos **"Start Creating"** para iniciar la extracción.

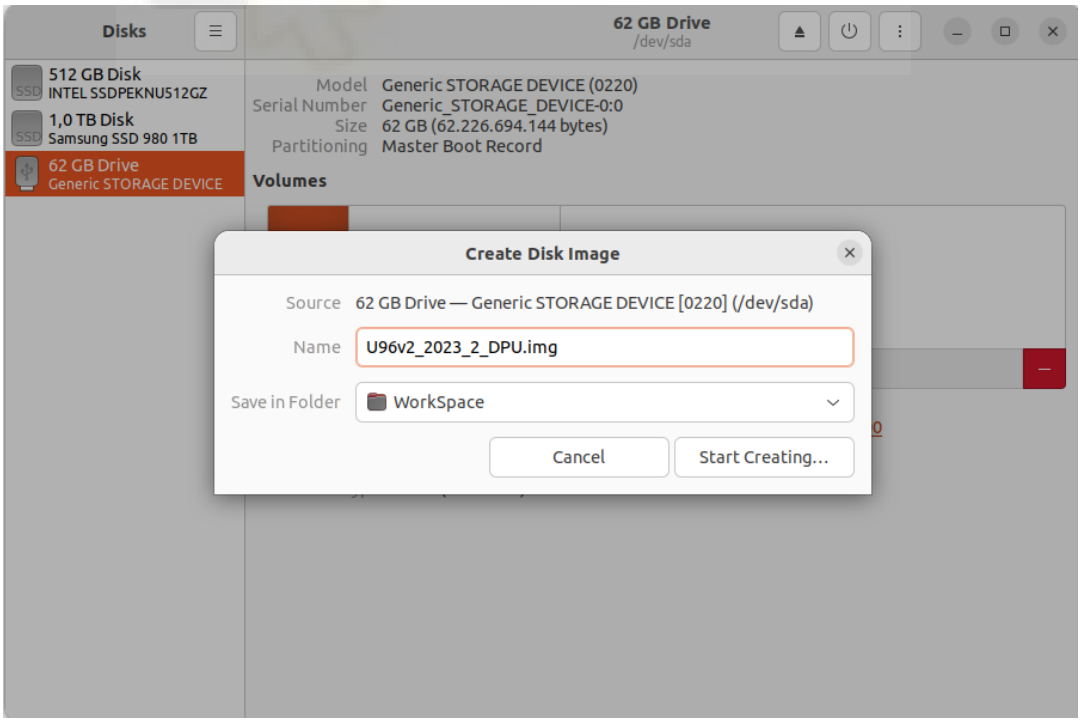

Figura 49. Configuración de la exportación de la imagen.

4.2. OBTENCIÓN DE LAS REDES NEURONALES CONVOLUCIONALES YOLO

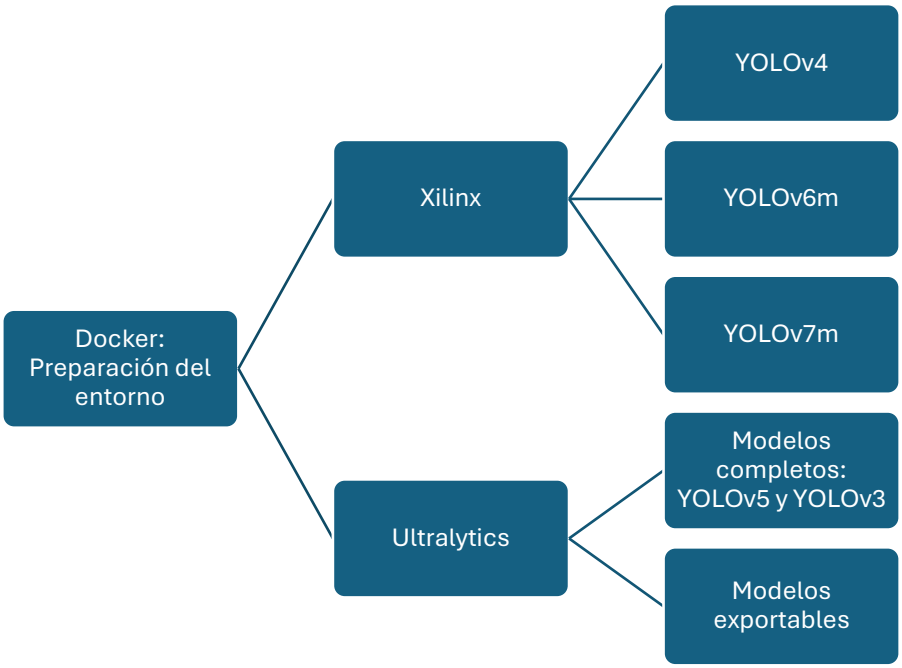

Figura 50. Infografía de la estructura del apartado 4.2..

En este capítulo, abordaremos el proceso de obtención y preparación de las redes neuronales YOLO (*You Only Look Once*). Exploraremos dos principales fuentes para trabajar con estos modelos:

1. **Repositorio de Xilinx Vitis-AI-Copyleft-Model-Zoo**

   Este repositorio proporciona modelos preconfigurados y adaptados para su uso con herramientas de Xilinx, como los modelos YOLOv4-csp, YOLOv6m y YOLOv7. Los modelos están listos para personalización y cuantización.

2. **Repositorio de Ultralytics**

   Ofrece modelos como YOLOv3 y YOLOv5 con acceso abierto para modificaciones, y versiones más recientes como YOLOv8, YOLOv9, YOLOv10 y YOLOv11, que pueden ser entrenados y exportados en formatos como ONNX. Otros modelos, como YOLO-NAS y YOLO-NAS-POSE, presentan ciertas limitaciones que exploraremos en detalle.

Modelos del Repositorio de Xilinx

El **Vitis-AI-Copyleft-Model-Zoo** es una herramienta fundamental para trabajar con los modelos YOLO en hardware específico de Xilinx. Este repositorio incluye:

- **Modelos preconfigurados para cuantización:**

  Estos modelos han sido modificados internamente para que sean compatibles con las herramientas de cuantización de Xilinx.

- **Tutoriales detallados:**

  Cada modelo incluye una guía paso a paso para personalizar y entrenar la red neuronal.

- **Sub-repositorios especializados:**

  Los modelos cuentan con sub-repositorios que interactúan directamente con el repositorio original, aprovechando las herramientas previas de estos proyectos. Sin embargo, debido a las diferencias en los desarrollos originales de cada modelo, el proceso de entrenamiento puede variar considerablemente.

Modelos del Repositorio de Ultralytics

Ultralytics, por su parte, ofrece modelos con una amplia flexibilidad para personalización y entrenamiento. Sus principales características son:

1. **Modelos modificables:**
   - YOLOv3 y YOLOv5 son ejemplos de redes accesibles que permiten modificaciones y personalización completa.
   - Los repositorios incluyen herramientas para entrenamiento y exportación.
2. **Modelos entrenables y exportables:**
   - Versiones más recientes como YOLOv8 a YOLOv11 pueden ser entrenadas con datasets personalizados y exportadas en formatos como ONNX.
   - Sin embargo, para hacerlos cuantizables, se requiere manipular directamente las capas internas del modelo, lo cual añade complejidad.
3. **Casos específicos:**
   - Modelos como YOLOv7-p6, YOLO-NAS y YOLO-NAS-POSE presentan limitaciones y podrían requerir métodos adicionales para su procesamiento.

Herramientas Complementarias

El procesamiento y preparación de los modelos YOLO, especialmente los provenientes del repositorio de Ultralytics, pueden requerir herramientas específicas como:

- **VitisAI y PyTorch:**

  Vitis AI es compatible con modelos exportados en ONNX y ofrece herramientas para su modificación y cuantización. PyTorch, un framework ampliamente utilizado en la investigación en inteligencia artificial, presenta una integración nativa con Vitis AI y una sintaxis más intuitiva y cercana a Python. En comparación con TensorFlow, también compatible con Vitis AI, PyTorch permite una escritura de código más concisa y facilita la implementación de modelos en hardware acelerado.

- **Procesos especializados:**

  Para casos específicos como YOLOv7-p6, se buscará desarrollar métodos que permitan su integración, en caso de disponer de tiempo suficiente.

Con estos elementos podremos completar el proceso de entrenamiento y cuantización de los modelos haciendo uso de las herramientas de VitisAI. El modelo se entrenará cuando aún esté en coma flotante, y posteriormente entrará en el proceso de cuantización de VitisAI.

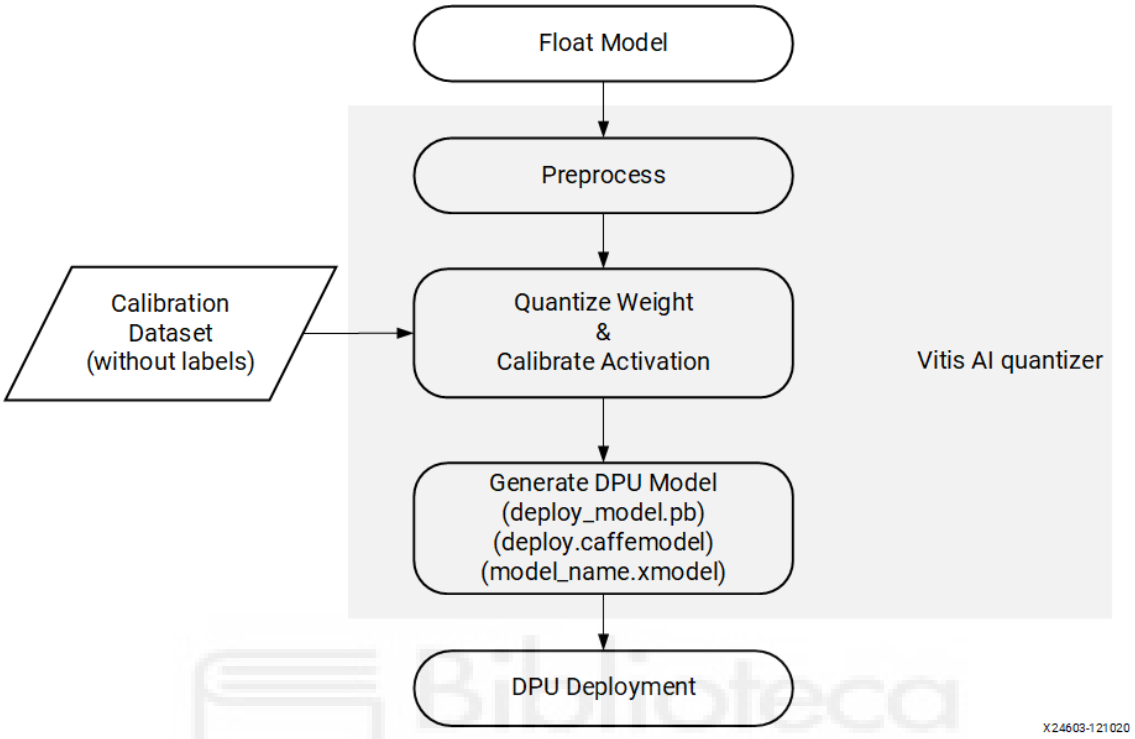

Figura 51. Infografía del proceso de cuantización usando Vitis AI quantizer.

4.2.1. ENTRENAMIENTO Y CUANTIZACIÓN

Todos los modelos serán entrenados y cuantizados utilizando el contenedor Docker `vitis-ai-pytorch:gpu`. Esta elección responde a las siguientes razones:

1. **Compatibilidad con los modelos del COPYLEFT:**

   El proceso de entrenamiento y cuantización emplea herramientas basadas en PyTorch, que son compatibles con todos los modelos proporcionados por el repositorio Vitis-AI-Copyleft-Model-Zoo.

2. **Referencias confiables:**

   La guía tutorial de LogicTronix, ampliamente utilizada en este tipo de proyectos, también emplea esta configuración para el entrenamiento y cuantización de modelos YOLO.

En cuanto a los modelos exportados en formato ONNX, será necesario adaptar el mismo contenedor Docker. Esto incluirá la actualización e instalación de herramientas específicas de Vitis para el procesamiento de ONNX, como se detallará en el apartado correspondiente.

Por ende, antes de proceder con el entrenamiento siempre deberemos seguir los siguientes pasos para la instalación de las herramientas necesarias y los controladores.

4.2.1.1. Instalación de Docker

Docker es una herramienta que, como ya hemos explicado en el capítulo 3, crea contenedores a partir de una imagen, y cada vez que abrimos un contenedor carga esa imagen, generando un espacio de trabajo que iniciará siempre en el mismo punto. Por otra parte, Docker no es la única herramienta que hace esto, así como CRI-O o Podman, también bastante conocidas, es posible que hagan uso de las mismas librerías, pero las modifican ligeramente. Por lo tanto, dado que este proyecto se realizó sobre un sistema operativo limpio no entraré en detalles de que librerías sería mejor reinstalar para asegurarnos una correcta instalación.

- Así pues, comenzaríamos agregando el repositorio apt a nuestra lista. Para ello, la guía propia de Docker (Docker) ofrece un script que tras ejecutarse se agregará Anexo 4. Docker_apt_repo_installer.sh, lo ejecutaremos de la siguiente manera:

```
./docker_apt_repo_installer.sh
```

- A continuación, instalaremos las últimas versiones de todas sus librerías

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

- En este punto Docker ya está instalado, pero deberemos hacer una serie de configuraciones más para poder ejecutarlo usando Vitis AI. Para ello, crearemos un grupo de usuarios capaz de ejecutar Docker y en él agregaremos al usuario que estemos usando para el desarrollo del proyecto. Tras configurarlo actualizaremos la lista de usuarios del grupo y probaremos a lanzar una demostración

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Con esa última línea, si no nos sale ningún error tendríamos Docker instalado perfectamente.

4.2.1.2. Instalación de CUDA

CUDA es un framework desarrollado por NVIDIA que facilita hacer un uso completo de las capacidades de la GPU desde las propias líneas de código. Por tanto, es utilizada en diferentes herramientas para optimizar el uso de recursos, en nuestro caso, los códigos usados en Copyleft hacen uso de él, así que para aprovecharlo vamos a instalarlos.

- Lo primero sería revisar los prerrequisitos de instalación, que para este caso solo se pide tener instalado un compilador de C++, gcc, que ya viene instalado por defecto en Ubuntu

- Lo siguiente que deberíamos hacer es ir a la siguiente web e indicar las características de nuestro sistema, que, mientras se trate de un ordenador común, deberían resultar en el siguiente enlace: https://developer.nvidia.com/cuda-

downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=22.04. Dado que la guía está planteada para Ubuntu en mi ordenador personal, la arquitectura común es x86_64.

- A continuación, tendremos que elegir el tipo de instalación, que ya depende del interés personal, sin embargo, no recomiendo el uso de la opción 'runfile' dado que es la más complicada para actualizar a versiones posteriores. En mi caso, por limitaciones de almacenamiento, opté por 'deb(network)' y seguí los siguientes pasos:

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204
/x86_64/cuda-keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt-get update
sudo apt-get -y install cuda-toolkit-12-6
```

- Una vez instalado el repositorio también podrás instalar los controladores de la tarjeta gráfica desde el terminal y dado que vamos a hacer uso de herramientas externas, recomiendo el uso de los controladores abiertos. Para instalarlos lanzaríamos el siguiente comando:

```
sudo apt-get install -y nvidia-open
```

Con esto, CUDA estaría instalado para su funcionamiento en el propio ordenador, sin embargo, como su uso será dentro de un contenedor, tendremos que hacer una serie de configuraciones adicionales que veremos en el siguiente apartado. También, si escribimos en la terminal `nvidia-smi` podremos ver información acerca de nuestra GPU.

4.2.1.3. Configuración de Docker con CUDA y creación del contenedor de Vitis AI en Docker

- Lo primero que tendremos que hacer será instalar el conjunto de herramientas de CUDA para trabajar en contenedores. Para instalarlo, debemos agregarlo al repositorio apt como hemos hecho con CUDA, sin embargo, lo haremos esta vez de forma local.

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --
dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
 && curl -s -L https://nvidia.github.io/libnvidia-
container/stable/deb/nvidia-container-toolkit.list | \
   sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-
container-toolkit-keyring.gpg] https://#g' | \
  sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit
```

- A continuación, configuraremos CUDA para que pueda usar el lanzador de Docker para inicializarse, tanto a nivel usuario como a superusuario.

```
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

```
nvidia-ctk runtime configure --runtime=docker --
config=$HOME/.config/docker/daemon.json
systemctl --user restart docker
sudo nvidia-ctk config --set nvidia-container-cli.no-cgroups --in-place
```

- Finalmente, deberemos hacer una comprobación final para ver si han realizado los cambios. Después de hacerlo probaremos a lanzar Docker y ejecutar la comprobación de NVIDIA

```
sudo gedit /etc/nvidia-container-runtime/config.toml
```

Aquí deberemos comprobar si en la línea donde se revisan los grupos, y habrá que garantizar que esté así: (no-groups = false) Anexo 5. Config.toml

- Para finalizar haremos una comprobación lanzando el siguiente ejemplo:

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

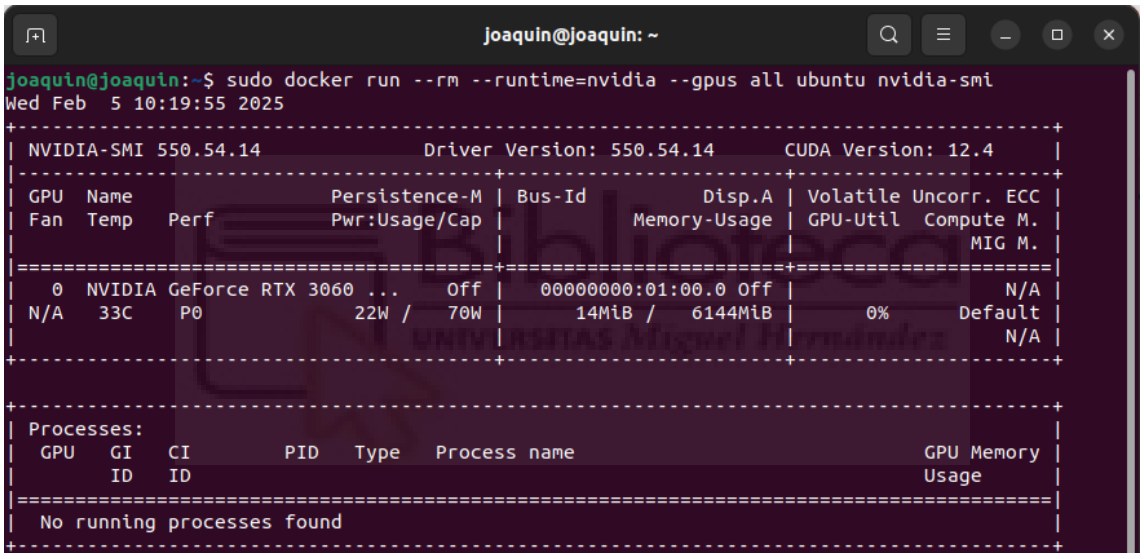

Figura 52. Comprobación del acceso a la GPU desde Docker.

- Nos descargaremos la versión más actualizada del repositorio de github de Vitis AI, por mantener un orden, yo lo descargué en el WorkSpace.

```
cd /opt/WorkSpace/
git clone https://github.com/Xilinx/Vitis-AI.git
```

- Nos desplazaremos desde la terminal hasta dicha carpeta y ejecutamos el siguiente comando

```
cd Vitis-AI/
./docker_run.sh xilinx/vitis-ai-pytorch-cpu:latest
```

- Tras la ejecución del comando anterior habremos entrado al Docker de la versión CPU y habremos aceptado las condiciones y la licencia de uso. Lo siguiente sería

instalar el Docker de GPU. Dado que este usará más recursos del sistema se tiene que generar desde el sistema haciendo uso de las herramientas que nos facilita AMD. Siguiendo los siguientes pasos se iniciará el proceso de generación del Docker compatible con los controladores de nuestra tarjeta.

```
cd docker/
./docker_build.sh -t gpu -f pytorch
```

Nota: Dentro del apartado de conclusiones, durante de la discusión de las dificultades encontradas se solucionará un problema específico que surgió durante la configuración de este apartado

- Tras configurar el Docker e incluirlo a la lista de contenedores procederemos a abrirlo y comprobar que funcione. Para ello nos desplazaremos al directorio raíz del repositorio descargado y ejecutaremos los siguientes comandos.

```
cd ..
./docker_run.sh xilinx/vitis-ai-pytorch-gpu:latest
($Docker) nvidia-smi
```

Nota: La salida por pantalla debería ser muy similar a la que nos apareció al comprobar el funcionamiento del Docker con CUDA

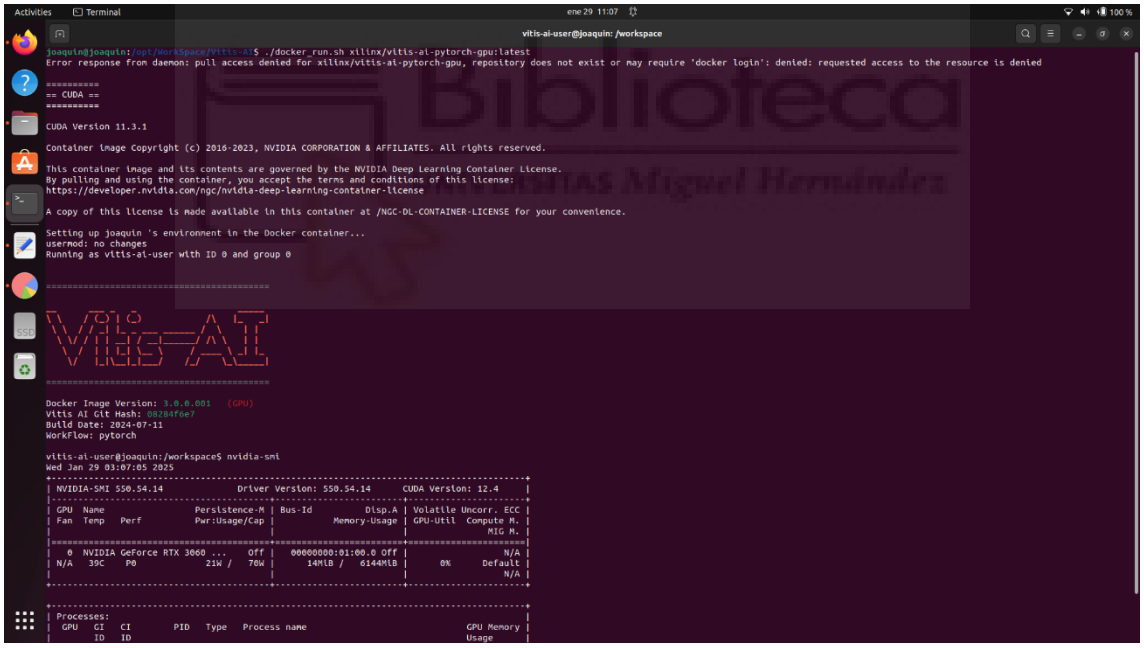

Figura 53. Acceso al Docker GPU de Vitis AI y su acceso a la GPU.

Con esto realizado, ya tendríamos todo listo para empezar con el entrenamiento.

4.2.1.4. MODELOS DEL COPYLEFT: YOLOv4-csp

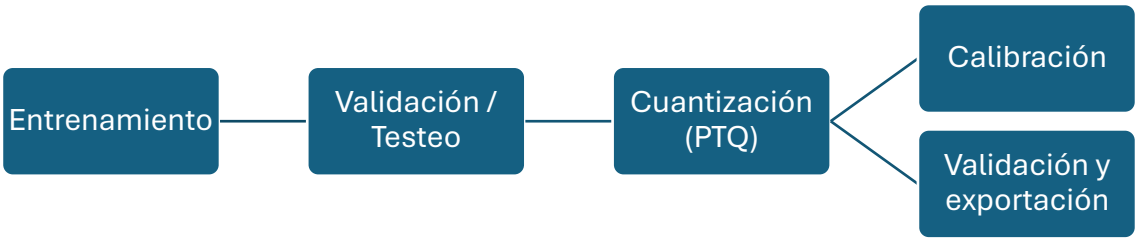

Figura 54. Infografía de las etapas para la obtención de YOLOv4.

El entrenamiento del modelo **YOLOv4-csp** comienza con la configuración del entorno y la instalación de los paquetes necesarios. A continuación, se describen las etapas principales del proceso:

1. **Configuración del entorno**

El primer paso consiste en activar el entorno `vitis-ai-pytorch` de Conda, que ya contiene herramientas como `torch` y `torchvision`, esenciales para trabajar con este modelo. Es importante asegurarse de que las versiones de estos paquetes sean compatibles con la GPU utilizada y con las herramientas de Vitis-AI.

En mi caso, fue necesario actualizar las versiones de `torch` y `torchvision` a versiones compatibles con mi GPU, pero lo menos recientes posible, para evitar problemas de compatibilidad con otros paquetes. Las versiones elegidas fueron:

- **torch:** 12.0
- **torchvision:** 0.13.1

2. **Preparación del dataset**

El modelo requiere un dataset bien configurado. Para este proyecto, decidí usar el segmento **COCO128** de Ultralytics, un subconjunto preparado del dataset COCO. Para utilizarlo, fue necesario configurar un archivo **YAML** adecuado Anexo 6. Coco128.yalm.

3. **Entrenamiento del modelo**

Una vez configurado el entorno y el dataset, es posible iniciar el entrenamiento. Aunque el repositorio incluye un script Bash para este propósito, preferí ejecutar el código manualmente para tener un mayor control sobre los recursos utilizados. El comando utilizado fue el siguiente:

```
cd code
python train.py --data data/coco128.yaml --cfg models/yolov4-csp-sppf.cfg
--weights "" --img 640 --batch-size 8 --device 0
```

Descripción de los argumentos:

- `--data`: Indica el dataset que se usará (en este caso, COCO128).
- `--cfg`: Especifica la configuración del modelo (modificable según los requisitos).
- `--weights`: Define los pesos iniciales. En este caso, se inicia desde cero (`""`).
- `--img`: Tamaño de las imágenes de entrada al modelo (640 píxeles).

- `--batch-size`: Número de imágenes que se procesan simultáneamente durante el entrenamiento.
- `--device`: Especifica la GPU o GPUs utilizadas (0 indica la GPU principal).

4. **Problemas durante el entrenamiento**

El entrenamiento en mi dispositivo falló debido a limitaciones de hardware. El modelo requería aproximadamente 7.5 GiB de memoria GPU, mientras que mi ordenador solo podía proporcionar 5.1 GiB.

Para reducir el consumo de GPU, probé las siguientes opciones:

- **Reducir el número de trabajadores:** Disminuyendo el paralelismo en la carga de datos.
- **Reducir el** `batch_size`**:** Sin embargo, no podía bajarlo más allá de 7, ya que ello ocasionaba una pérdida significativa de precisión en el modelo.

A pesar de estos intentos, no fue posible completar el entrenamiento en mi dispositivo por restricciones de hardware y falta de conocimientos avanzados sobre redes neuronales para ajustar las capas y recursos necesarios.

5. **Alternativas exploradas para completar el entrenamiento del modelo**

Dado que no pude entrenar el modelo YOLOv4-csp-sppf, consideré usar redes neuronales más pequeñas, como YOLOv4-csp o YOLOv4. Cada uno de estos modelos incorpora diferentes características arquitectónicas que influyen en su rendimiento, eficiencia y capacidad de adaptación a dispositivos con recursos limitados.

Los componentes clave son:

- **Cross Stage Partial Networks (CSP):** Este módulo mejora la eficiencia de la red dividiendo las características extraídas en dos partes: una que se procesa en la red y otra que se combina posteriormente. Este enfoque reduce la redundancia de información y optimiza el uso de parámetros, lo que aumenta la capacidad de aprendizaje del modelo sin incrementar significativamente su tamaño.
- **Spatial Pyramid Pooling Fast (SPPF):** Este módulo permite a la red captar información contextual a múltiples escalas al combinar características de diferentes tamaños de ventanas de recepción. Esto mejora la detección de objetos pequeños y en entornos complejos, mientras mantiene una alta velocidad de procesamiento.

Otro punto clave a tener en cuanta es la relación entre los componentes y el proceso de cuantización. El módulo SPPF añade nuevas relaciones entre las capas haciendo al modelo mucho más sensible a la cuantización, mientras que el módulo de CSP funciona como una simplificación de las capas lo que en ocasiones puede aumentar el rendimiento y precisión del modelo cuantizado.

Para intentar realizar el entrenamiento del modelo con distintos módulos los comandos utilizados fueron:

Para YOLOv4-csp:

```
python train.py --data data/coco128.yaml --cfg models/yolov4-csp.cfg --weights "" --img 640 --batch-size 8 --device 0
```

Para YOLOv4:

```
python train.py --data data/coco128.yaml --cfg models/yolov4.cfg --weights yolov4.pt --img 640 --batch-size 8 --device 0
```

Nota: Los pesos para YOLOv4 no estaban disponibles en el repositorio, por lo que tuve que obtenerlos de una fuente externa.

Estas opciones presentaron un consumo de recursos menor, pero igualmente requerían más memoria GPU de la que mi equipo podía ofrecer. Como resultado, tuve que posponer el entrenamiento de estos modelos para ejecutarlo en equipos más potentes.

4.2.1.5. MODELOS DEL COPYLEFT: YOLOv6m

Figura 55. Infografía de las etapas para la obtención de YOLOv6m.

A diferencia de modelos anteriores como YOLOv4, la configuración e implementación de YOLOv6m incluye pasos más complejos que requieren modificaciones significativas en las librerías y dependencias. En este apartado se detalla el proceso llevado a cabo para configurar, entrenar y cuantizar YOLOv6m, así como los retos enfrentados y las soluciones implementadas.

1. CONFIGURACIÓN DEL ENTORNO

Paso 1: Instalación de dependencias

Para iniciar el proceso, se abrió un contenedor Docker y se navegó a la carpeta del modelo YOLOv6m. Una vez dentro, se ejecutó un script de instalación de librerías, que requirió modificaciones debido a su antigüedad. En particular, se realizaron los siguientes ajustes:

- Actualización de las versiones de NumPy para garantizar compatibilidad con las herramientas modernas. Anexo 7. Requeriments.txt
- Ajustes similares a los realizados en YOLOv4 para garantizar que las dependencias estuvieran alineadas con las versiones actuales de las librerías utilizadas en PyTorch y CUDA.

Tras realizar las modificaciones, se ejecutó el siguiente código para instalar las dependencias necesarias:

```
python setup.py install
```

```
sudo pip install --no-cache-dir --extra-index-url https://pypi.nvidia.com pytorch-quantization
```

Paso 2: Modificación de PyTorch

En la carpeta `torch_rewriters`, se ejecutó el script de configuración para modificar PyTorch y adaptarlo a las necesidades de YOLOv6m. Este proceso incluyó la ejecución de `setup.py` y la resolución manual de errores indicados en pantalla.

Paso 3: Preparación para cuantización

Dentro de la carpeta `yolov6_nndct`, se encontró una versión del modelo acondicionado para el entrenamiento consciente de cuantización (QAT). Este paso fue fundamental para garantizar que las operaciones fueran cuantizables y que el modelo pudiera entrenarse eficazmente con posterioridad.

2. ENTRENAMIENTO DEL MODELO

Primera etapa: Optimización del modelo

Dado que la configuración de hardware disponible presentaba limitaciones, se decidió iniciar el entrenamiento con el dataset reducido COCO128 y, dependiendo de los tiempos, escalar a COCO completo. El entrenamiento inicial fue lanzado con el siguiente comando:

```
python tools_nndct/train.py --batch 4 --conf configs/repopt/yolov6m_hs.py --data data/coco128.yaml --device 0 --name yolov6m_hs
```

Tras 2 horas de entrenamiento, se obtuvieron los siguientes resultados:

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.264
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.406
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.300
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.099
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.201
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.412
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.278
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.434
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.457
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.152
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.398
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.629
Epoch: 399 | mAP@0.5: 0.406005077530348 | mAP@0.50:0.95: 0.2639658345144499
```

Interpretación de mAP

- **mAP@0.5**: Este valor indica la precisión promedio de detección para todos los objetos cuando se utiliza un umbral de confianza de 0.5. En la práctica, esto significa que el modelo identifica correctamente los objetos que tienen una probabilidad asignada de al menos el 50%. Este indicador es especialmente útil para evaluar el desempeño del modelo bajo condiciones menos estrictas y para realizar ajustes en los umbrales de confianza durante el proceso de

posprocesamiento. Por ejemplo, puede emplearse para determinar si los falsos positivos predominan o si se están perdiendo detecciones importantes.

- **mAP@0.5:0.95**: Este métrico proporciona una evaluación más completa del desempeño del modelo, ya que calcula la precisión promedio en un rango de umbrales de confianza que varía desde 0.5 hasta 0.95, con incrementos de 0.05. Este enfoque más exhaustivo mide la capacidad del modelo para equilibrar precisión y recall en diferentes niveles de confianza, proporcionando una visión integral de su eficacia general. Es considerado el estándar más importante para comparar modelos, especialmente en competiciones de benchmark como COCO, ya que refleja de manera más precisa el rendimiento real del modelo en situaciones prácticas.
- **Otros datos**: indicaciones como el IoU, area o maxDets hacen referencia a las distintas restricciones sobre las cajas de los objetos detectados. Indican el umbral para eliminar cajas superpuestas, el tamaño de las cajas y el número total de cajas antes del NMS. Por otra parte, Average Precision y Average Recall, la precisión hace referencia a las detecciones correctas sobre el total de detecciones, eso quiere decir que es una métrica más orientada a reducir el número de falsos positivos (detecciones erróneas), y el recall nos indica la capacidad del modelo para detectar los objetos correctos sobre todas las detecciones correctas indicadas en los labels, por tanto, orientada a optimizar los falsos negativos.

Estos resultados iniciales reflejan un modelo apenas ajustado, ya que los pesos no se han optimizado completamente.

Segunda etapa: Entrenamiento optimizado

En esta fase, se utilizó una configuración mejorada para ajustar los pesos del modelo:

```
python tools_nndct/train.py --batch 4 --conf
configs/repopt/yolov6m_opt_wodfl.py --data data/coco128.yaml --device 0
--name yolov6m_opt_wodfl
```

Cuyos resultados son:

```
Epoch: 399 | mAP@0.5: 0.30305028489375124 | mAP@0.50:0.95: 0.18422787652178713
```

El descenso en el mAP es esperado, ya que el modelo ahora está corrigiendo los pesos iniciales y ajustándolos con base en las imágenes del dataset.

Tercera etapa: Entrenamiento con destilación

Se implementó un entrenamiento avanzado con destilación. En este proceso, un modelo maestro entrena a un modelo estudiante más ligero, optimizando recursos y comprimiendo operaciones matriciales. La configuración utilizada fue la siguiente:

```
python tools_nndct/train.py --batch 5 --conf
configs/repopt/yolov6m_opt_wodfl.py --data data/coco128.yaml --device 0
```

```
--name yolov6m_opt_wodfl_distill --distill --teacher_model_path
runs/train/yolov6m_opt_wodfl/weights/best_ckpt.pt
```

Cuyos resultados son:

```
Epoch: 399 | mAP@0.5: 0.4491614186789837 | mAP@0.50:0.95: 0.2906631792533739
```

Tras obtener resultados iniciales prometedores en el entrenamiento con destilación, se planteó llevar el modelo hacia un flujo de cuantización para optimizar su rendimiento en dispositivos de hardware específicos como la FPGA. Este proceso incluyó dos enfoques principales: **Post Training Quantization (PTQ)** y **Quantization Aware Training (QAT)**, cada uno con sus respectivas ventajas y limitaciones. Siguiendo el flujo propuesto por el repositorio, haremos primero PTQ y terminaremos con QAT.

El primer paso fue realizar la cuantización del modelo maestro mediante el script `export_nndct.py`. Sin embargo, debido a problemas con la actualización de paquetes y dependencias, fue necesario realizar modificaciones manuales en la biblioteca `torch_rewriters`. Anexo 8. Summary.py:

- Edición del archivo: `/opt/vitis_ai/conda/envs/vitis-ai-pytorch/lib/python3.7/site-packages/pytorch_nndct/utils/summary.py`.
- Cambios realizados:
  - Comentar la línea `spu.print.summary`.
  - Añadir la instrucción `import logging`.

Después de realizar los ajustes, se ejecutó el siguiente comando para generar un modelo cuantizado provisional:

```
python tools_nndct/quantization/export_nndct.py –conf
configs/repopt/yolov6m_opt_wodfl.py \
--weights runs/train/yolov6m_opt_wodfl_distill/weights/best_ckpt.pt \
--device 0 --batch-size 1 --calib-batch-number 1000 --eval-float --data
data/coco128.yaml
```

Cuyos resultados son:

```
Post Training Quantization model mAP0.5=0.4374, mAP0.5_0.95=0.2825
```

Este modelo cuantizado ya es funcional para implementarse en la FPGA, pero no explota todo su potencial en términos de precisión y capacidad de detección.

Para mejorar las limitaciones del PTQ, se empleó un enfoque de entrenamiento durante la cuantización (QAT), utilizando la técnica de destilación. Este método involucra un modelo maestro no cuantizado y un modelo estudiante cuantizado, donde se corrigen iterativamente las predicciones del estudiante. El proceso comenzó con el siguiente comando:

```
python tools_nndct/train.py --name yolov6m_opt_wodfl_distill_qat –conf
configs/repopt/yolov6m_opt_wodfl_qat.py --device 0 --quant --batch-size 3
--teacher_model_path
runs/train/yolov6m_opt_wodfl_distill/weights/best_ckpt.pt --distill --
distill_feat --epochs 24 --workers 32 --data data/coco128.yaml
```

Cuyos resultados son:

```
Epoch: 23 | mAP@0.5: 0.4023382982461384 | mAP@0.50:0.95: 0.2271219094806496
```

Aunque los resultados iniciales son inferiores al PTQ debido al número limitado de iteraciones, este proceso tiene un mayor potencial de mejora a medida que se incrementan las épocas de entrenamiento. El modelo final se exportó utilizando el siguiente comando:

```
python tools_nndct/quantization/export_nndct.py --conf
configs/repopt/yolov6m_opt_wodfl_qat.py --weights
runs/train/yolov6m_opt_wodfl_distill_qat/weights/best_ckpt.pt --device 0 --
batch-size 1 --qat --data data/coco128.yaml
```

El modelo QAT ofrece una mayor capacidad de detección y precisión en comparación con el modelo PTQ, especialmente cuando se dispone de un hardware más potente o se realizan más iteraciones de entrenamiento. Este flujo ha sido exitoso, logrando un modelo eficiente y optimizado para su implementación en FPGA.

En el capítulo del trabajo adicional haremos uso de este entrenamiento corregido para darle una verdadera finalidad.

4.2.1.6. MODELOS DEL COPYLEFT: YOLOv7

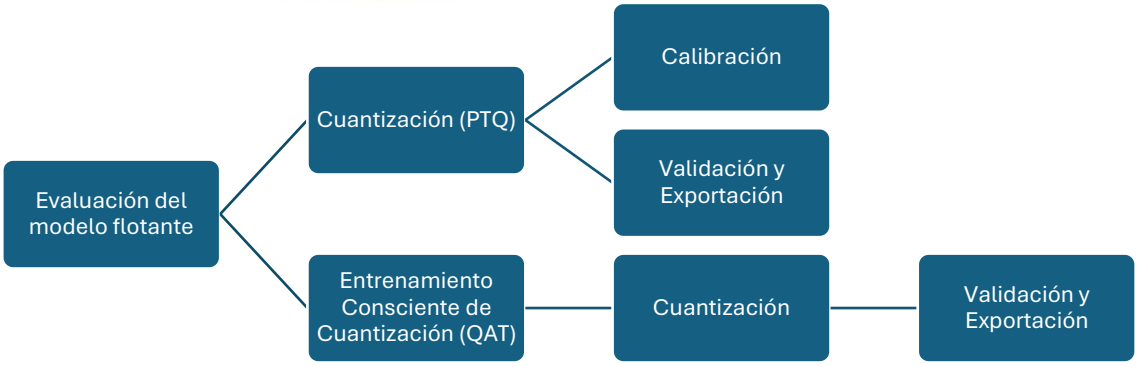

Figura 56. Infografía de las etapas para la obtención de YOLOv7.

Al igual que en el caso de YOLOv4, se comenzó activando el entorno de **Conda** e instalando las librerías necesarias para Python, recogidas en el archivo `requirements.txt` del repositorio de YOLOv7. Antes de proceder con la instalación, este archivo fue modificado para actualizar las versiones de las dependencias a más recientes, asegurando compatibilidad con los entornos utilizados. Una vez ajustadas, las dependencias se instalaron mediante el comando correspondiente.

Posteriormente, se preparó el conjunto de datos y se inició la configuración para el entrenamiento. El primer paso consistió en posicionarse en el directorio del repositorio (`yolov7/`) y descargar los pesos preentrenados de YOLOv7 (modelo *yolov7m*) desde el enlace del repositorio de original del proyecto de GitHub (https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt). Estos pesos se almacenaron en el directorio local del proyecto. A continuación, se procedió a evaluar el modelo preentrenado con el conjunto de datos COCO utilizando el siguiente comando:

```
cd yolov7/
wget
https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt
python test_nndct.py --data data/coco128.yaml --img 640 --batch 1 --conf
0.001 --iou 0.65 --device 0 --weights yolov7.pt --name yolov7_640_val --
quant_mode float
```

Este script permitió verificar el rendimiento del modelo con los pesos preentrenados. Si bien no se realizaron ajustes en esta etapa, el procedimiento ofreció una visión general del desempeño del modelo.

El primer método aplicado fue la **Cuantización Post-Entrenamiento (PTQ, Post-Training Quantization)**, un procedimiento que permite convertir un modelo ya entrenado en un formato cuantizado. Esto implica adaptar tanto los pesos como las activaciones a valores de menor precisión, generalmente de 32 bits flotantes a 8 bits enteros, manteniendo el modelo funcional en entornos con recursos limitados, como las DPU. Para realizar la cuantización en YOLOv7, se siguieron los pasos proporcionados por el repositorio oficial, transformando las activaciones no compatibles, como `Sigmoid` y `SiLU`, en equivalentes más optimizados: `Hard-Sigmoid` y `Hard-Swish`, respectivamente. Los comandos utilizados para este propósito fueron:

```
python test_nndct.py --data data/coco128.yaml --img 640 --batch 1 --conf
0.001 --iou 0.65 --device 0 --weights yolov7.pt --name yolov7_640_val --
quant_mode calib --nndct_convert_sigmoid_to_hsigmoid --
nndct_convert_silu_to_hswish
python test_nndct.py --data data/coco128.yaml --img 640 --batch 1 --conf
0.001 --iou 0.65 --device 0 --weights yolov7.pt --name yolov7_640_val --
quant_mode test --nndct_convert_sigmoid_to_hsigmoid --
nndct_convert_silu_to_hswish
```

La ejecución de estos comandos generó un modelo cuantizado en el formato requerido, además de corregir las activaciones del modelo para hacerlo compatible con la DPU. El modelo se obtuvo usando el método PTQ, y para obtener el archivo XModel era necesario usar la siguiente línea:

```
python test_nndct.py --data data/coco128.yaml --img 640 --batch 1 --conf
0.001 --iou 0.65 --device 0 --weights yolov7.pt --name yolov7_640_val --
quant_mode test --nndct_convert_sigmoid_to_hsigmoid --
nndct_convert_silu_to_hswish --dump_model
```

El segundo método evaluado fue el **Entrenamiento Durante la Cuantización (QAT, Quantization Aware Training)**. Este proceso tiene como objetivo mejorar el rendimiento del modelo cuantizado simulando directamente los efectos de la cuantización durante el entrenamiento. Esto permite ajustar los pesos y activaciones para minimizar la pérdida de precisión introducida por la reducción en la representación de los datos.

El procedimiento para lanzar la cuantización mediante QAT en YOLOv7 se realiza con el siguiente script:

```
python train_qat.py --workers 4 --device 0 --batch-size 1 --batch 1 --data
data/coco128.yaml --img 640 640 --cfg cfg/training/yolov7.yaml --weights
yolov7.pt --name yolov7_qat --hyp data/hyp.scratch.p5_qat.yaml --
nndct_convert_sigmoid_to_hsigmoid --nndct_convert_silu_to_hswish --
log_threshold_scale 100
```

A diferencia de YOLOv6, donde el procedimiento de QAT tiene un enfoque más eficiente en términos de carga computacional, YOLOv7 presenta mayores demandas de recursos. En YOLOv6, el modelo maestro (preentrenado) se carga, evalúa, descarga y cuantiza en pasos secuenciales. Posteriormente, se evalúa el modelo cuantizado y se comparan los resultados entre el maestro y el cuantizado, de manera similar al proceso de destilación de modelos. Este enfoque permite una gestión más eficiente de la memoria y reduce significativamente la carga computacional.

Por otro lado, en YOLOv7, el procedimiento para QAT es más exigente, ya que ambos modelos (maestro y cuantizado) se cargan simultáneamente en memoria y se evalúan de manera directa. Este enfoque, al evitar la destilación, implica un uso intensivo de recursos computacionales, lo que lo hace inviable en sistemas con hardware limitado como el utilizado en este proyecto. A pesar de los intentos de optimización, no se logró reducir la demanda de recursos lo suficiente para completar el procedimiento de QAT en YOLOv7.

En conclusión, mientras que el enfoque de QAT fue exitoso en YOLOv6 debido a su gestión más eficiente de los modelos, en YOLOv7 no se pudo llevar a cabo por las limitaciones de hardware. Esto resalta la importancia de adaptar los procesos de cuantización a las capacidades computacionales disponibles, especialmente en entornos con restricciones de recursos.

4.2.1.7. MODELOS MODIFICABLES DE ULTRALYTICS: YOLOv5

Se decidió comenzar con **YOLOv5** en lugar de YOLOv3 debido a que este cuenta con un tutorial oficial que describe cómo cuantizarlo para su uso en plataformas preparadas con **Vitis-AI**. Sin embargo, a pesar de seguir las recomendaciones del tutorial e intentar entrenar el modelo para lograr el resultado indicado, fue imposible obtener un modelo funcional. Esto se debe aparentemente a que el tutorial es incompleto o contiene errores, lo cual se descubrió tras múltiples intentos de subsanar los problemas encontrados.

El objetivo inicial era modificar las capas y nodos necesarios para hacer que el modelo fuera cuantizable. En YOLOv5, fue necesario alterar las operaciones en las capas finales, ya que estas contenían operadores incompatibles con la cuantización, como `meshgrid` y `split_with_sizes`.

Además, se realizaron cambios en la función de activación de las neuronas. Al adaptar YOLO para operar con datos enteros en lugar de flotantes, el rango de valores de entrada se ve limitado. Para minimizar la pérdida de precisión, se permitió que la red procesara valores negativos. Inicialmente, YOLOv5 utilizaba la función de activación **SiLU**, que convierte los valores negativos en valores cercanos a cero y luego aplica una operación lineal. Esto generaba un problema, ya que, al trabajar con valores negativos, las neuronas activadas por SiLU quedaban inutilizadas, o "muertas".

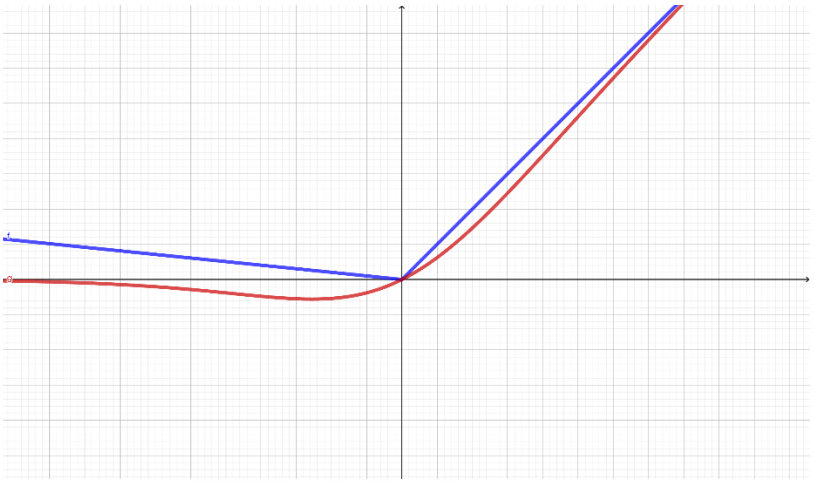

Figura 57. Diferencias entre SiLU (rojo) y LeakyReLU con pendiente negativa (azul).

La guía de usuario UG1414, proporcionada por Vitis-AI, no considera SiLU como una función compatible, por lo que se reemplazó por **LeakyReLU** con pendiente negativa. Esta función permite procesar valores negativos y, según el tutorial y foros de AMD, es la opción más eficiente en términos de rendimiento. Mientras que la función SiLU solo deja pasar los valores positivos manteniendo su magnitud, la función LeakyReLU, multiplica los valores negativos por el hiperparámetro que hemos llamado pendiente. Usualmente, este parámetro tendría solo valores positivos entre 0 y 1, pero debido a las restricciones de cuantización de PyTorch solo están permitidos valores negativos para la pendiente. El valor específico recomendado en la guía de Mario Bergeron parece haberse obtenido de manera empírica y no tiene una explicación clara detrás de su valor.

Tras implementar estas modificaciones, se entrenaría un modelo y se exportaría en formato `.pth`.

El siguiente paso sería realizar el proceso general de cuantización. El tutorial proporcionaba un script para llevar a cabo este procedimiento, pero este se tuvo que modificar para adaptarlo a nuestras necesidades específicas.

A pesar de seguir correctamente las instrucciones, el modelo no era generado debido a errores inherentes al tutorial que impedían obtener un modelo entrenado cuantizable. Para resolver estos problemas, se desarrollaron las modificaciones necesarias siguiendo estrictamente las directrices del UG1414. Esto permitió obtener un modelo cuantizable; sin embargo, surgieron nuevos problemas durante la cuantización y la compilación.

Los principales inconvenientes ocurrieron en la **capa 24**, que corresponde a las detecciones. En particular, los módulos de multiplicación y las permutaciones de matrices

con diferentes encabezados causaron problemas. Estos problemas forzaron al modelo a dividirse en bloques secuenciales que se ejecutaban parcialmente en la DPU y parcialmente en la CPU. Esta división introdujo cuellos de botella severos en el rendimiento, ya que no se aprovechaba plenamente la capacidad de procesamiento de la DPU.

Para resolver esto, habría sido necesario compactar y modificar algunas operaciones en las capas afectadas. Sin embargo, desarrollar las habilidades técnicas necesarias para realizar estos ajustes requería tiempo y conocimientos avanzados, que no estaban disponibles durante este proyecto.

Debido a las limitaciones mencionadas, no se logró completar la cuantización de los modelos completos de Ultralytics. Este apartado resalta las complejidades inherentes al proceso de cuantización en arquitecturas avanzadas como YOLOv5 y la necesidad de contar con tutoriales claros y completos para garantizar resultados exitosos.

4.2.1.8. MODELOS ENTRENABLES Y EXPORTABLES DE ULTRALYTICS: YOLOv5 Y YOLOv11

En este apartado, intentaremos explorar la posibilidad de entrenar un modelo y exportarlo como ONNX para posteriormente cuantizarla e implementarla en la DPU. Para la obtención de estos modelos, en lugar de hacer uso del repositorio específico de YOLOv5 o YOLOv3 haremos uso del repositorio general de Ultralytics que ofrece la posibilidad de acceder de forma remota a los archivos de configuración de los distintos modelos que desarrolla, entre los que cabe destacar YOLOv5, pues con cada avance de YOLO y el desarrollo de las nuevas versiones y módulos, han ido actualizando las capas y nodos de la quinta versión sin cambiar su estructura para integrarle estos nuevos avances. Así pues, consideramos probar a entrenar y exportar la versión de YOLOv5 con las últimas actualizaciones y también la última versión de YOLO que disponían, que era YOLOv11. Empezando por la preparación del entorno y del conjunto de datos tendremos todo listo para poder ejecutar el entrenamiento. Este proceso tendrá lugar en nuestro hardware por lo que deberemos tenerlo en cuenta a la hora de establecer los parámetros del entrenamiento, especialmente el batch_size, que limitará el desempeño final del modelo debido a la reducción de varianza en los estímulos del modelo, y las epochs, que podría limitar el desempeño final del modelo debido que a más épocas más posibilidades de encontrar los pesos ideales.

Desde el propio repositorio descargado de ultralytics, sin ninguna modificación, para YOLOv5 es posible entrenar el modelo solo instalando los requerimientos y lanzando el entrenamiento.

```
cd yolov5/
pip install -r requeriments.txt
python train.py --img 640 –epochs 50 --data coco128.yaml --weights
yolov5m.pt
```

Esto nos generaría un modelo entrenado con COCO128, sin embargo, no estaría en un formato accesible, pues estaría en formato `.pth` al que solo se puede acceder con

PyTorch. Por ende, para finalizar, tendríamos que exportar ese modelo al formato deseado, en nuestro caso ONNX.

```
python export.py --weights runs/exp1/weights/best.pth --include onnx
```

Esta librería nos permite también exportar los modelos preentrenados con COCO. Para ello, cuando indicamos los pesos, indicamos los pesos originales, como por ejemplo, para YOLOv5m sería:

```
python export.py --weights yolov5m.pt --include onnx
```

Por otra parte, para el entrenamiento de YOLOv11 así como para otras versiones, deberemos hacer uso del repositorio principal de Ultralytics (https://github.com/ultralytics/ultralytics.git). Sin embargo, para su ejecución tendremos que preparar un breve script donde indicaremos los parámetros. Por tanto, estos serían los pasos a seguir:

1. **Preparar el entorno**

```
git clone https://github.com/ultralytics/ultralytics.git
cd ultralytics/
pip install -r requeriments.txt
```

2. **Definir los parámetros y el script de entrenamiento**

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolo11m.pt")

# Train the model
train_results = model.train(
   data="coco8.yaml",  # path to dataset YAML
   epochs=50,  # number of training epochs
   imgsz=640,  # training image size
   device=0,   # device to run on, i.e. device=0 or device=0,1,2,3 or device=cpu
)

# Evaluate model performance on the validation set
metrics = model.val()

# Perform object detection on an image
results = model("../../datasets/dataset_test/flor.jpg")
results[0].show()

# Export the model to ONNX format
path = model.export(format="onnx")  # return path to exported model
```

Aquí definimos parámetros para el entrenamiento como el dataset o la imagen con la que queremos que se pruebe el modelo al terminar. Sin embargo, también podemos pedir simplemente exportar el modelo preentrenado, como para YOLOv5:

```
from ultralytics import YOLO

model = YOLO("yolo11m.pt")
model.export(format="onnx", dynamic=True)
```

Estos scripts los guardaremos en el propio directorio donde hemos descargado el repositorio y los ejecutaremos. Obteniendo así los modelos ONNX, en el caso de nuestro proyecto, solo entrenamos YOLOv5m con la intención de implementarlo primero y después entrenar más modelos, así como explorar las demás versiones y añadiendo módulos como NAS, que es muy útil para las imágenes médicas.

4.2.2. COMPILACIÓN DE XMODELS

Después de realizar los entrenamientos y obtener los modelos en formato **XModel**, el siguiente paso es compilar dichos modelos para que se adapten a las dimensiones específicas del chip. Este proceso se realiza utilizando el compilador incluido en las herramientas de **Vitis-AI** de Xilinx.

En Vitis-AI, contamos con dos compiladores principales:

1. `vai_c_xir`: Convierte los **XModels** al formato específico del chip.
2. `vai_c_tensorflow2`: Convierte archivos en formato **PB** (TensorFlow) a **XModels** adaptados al chip.

La diferencia radica únicamente en el archivo de entrada que se utiliza, ya sea un XModel preexistente o un modelo en formato PB. En cualquier caso, ambos compiladores requieren de un archivo adicional crucial: `arch.json`.

El archivo `arch.json` define las características de la unidad **DPU** implementada en el sistema. Este archivo se genera junto con la aplicación creada en Vitis y contiene la "huella" de la DPU, especificando su tamaño y configuración.

En el caso del proyecto, la configuración corresponde a un **MPSoC** con dimensiones **B1600**, lo que genera una DPU específica cuya configuración ya fue revisada previamente en Vitis, la huella es "`0x101000056010404`".

Para compilar los modelos obtenidos en formato XModel, se utilizó el siguiente comando de ejemplo:

```
vai_c_xir -a arch.json -x modelo.xmodel -o modelos_compilados/ -n
modelo_compilado
```

Este comando realiza lo siguiente:

- Define el archivo de configuración `arch.json` para la DPU.
- Especifica el modelo XModel de entrada con el parámetro `-x`.
- Indica la carpeta de salida con `-o` (en este caso, `modelos_compilados/`).

- Establece el nombre del modelo compilado con `-n` (en este caso, `modelo_compilado`).

El resultado es un archivo llamado `modelo_compilado.xmodel` que está optimizado para la DPU definida en `arch.json`.

Tras explicar el proceso general, se procedió a compilar los modelos obtenidos durante las fases previas del proyecto.

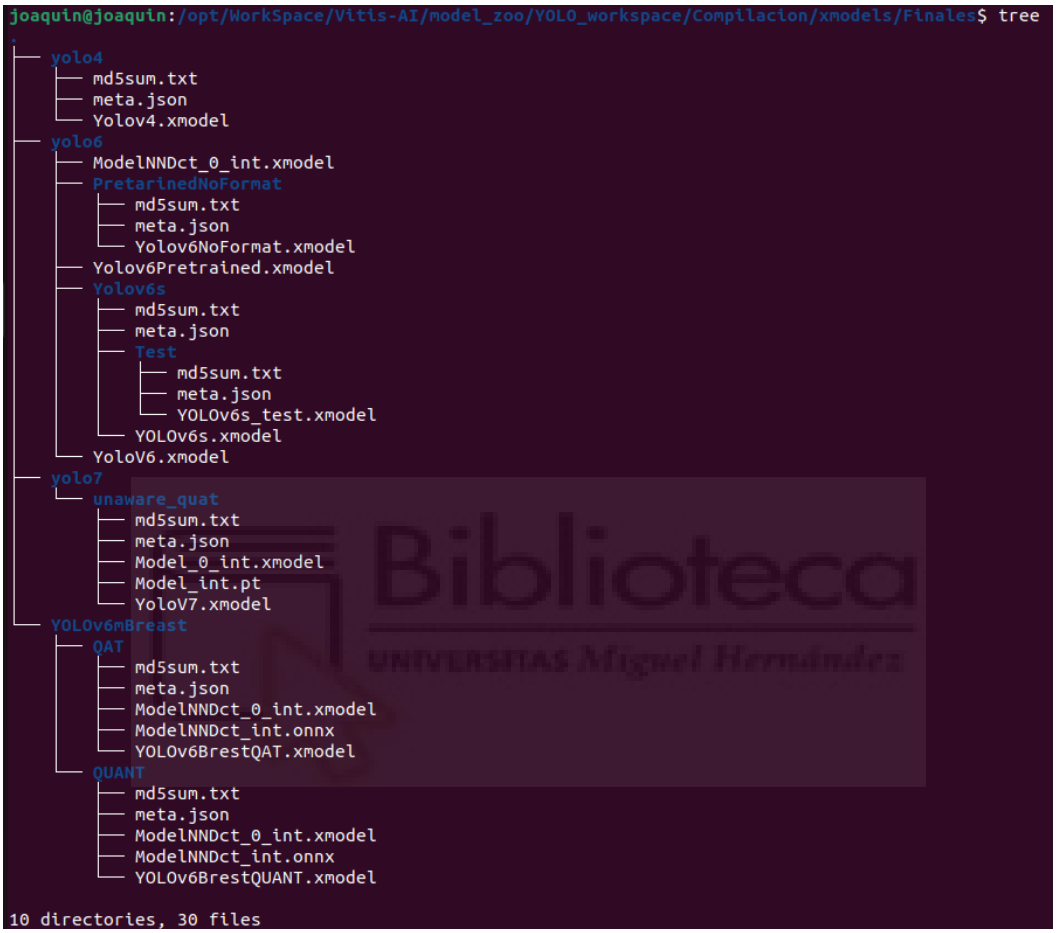

Figura 58. Modelos en formato XModel compilados.

Además de los modelos entrenados, también se compilaron modelos proporcionados por el **model_zoo** de Vitis-AI, como el **ResNet50_pt**. La guía oficial de Vitis-AI recomienda descargar y compilar estos modelos de ejemplo como referencia.

Para ello, Vitis-AI ofrece un script llamado `downloader.py`, que permite descargar los XModels disponibles en el model_zoo. Sin embargo, este script solo realiza la descarga y no incluye el proceso de compilación.

Como alternativa, se desarrolló un script basado en `downloader.py` que añadía funcionalidad para compilar los modelos descargados. Este script toma como entrada el archivo `arch.json`, garantizando que los modelos estén adaptados a la configuración del sistema.

En nuestro caso, se utilizó un script basado en la versión 3.0 de Vitis-AI, al cual se le realizaron las actualizaciones necesarias para incluir la compilación automática. Anexo 13. Compile_modelzoo_pyt.sh y Anexo 14. Compile_model_zoo_tf2.sh.

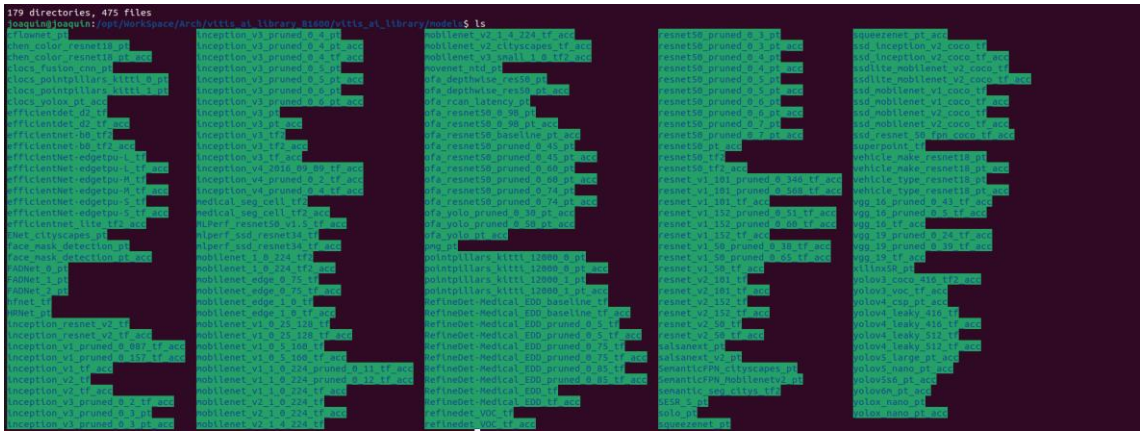

Figura 59. Resultados del script compilador de los modelos del model_zoo.

4.2.3. COMPILACIÓN DE MODELOS ONNX

En lo referente a la preparación de los modelos ONNX para funcionar dentro de la DPU hay muy poca información y tampoco hay tutoriales muy completos en la materia. Por lo tanto, para poder hacer uso de esto se preparó un script en Python para poder condensar y generalizar el proceso de cuantización del modelo para su uso en la DPU. Los modelos ONNX que funcionan en la DPU tienen dos formas de obtenerse, la mira es cuantizando un modelo de la misma manera que se hace en el copyleft y exportarlo a ONNX por medio de la API de PyTorch. La otra forma de hacerlo, en teoría, sería cuantizando todos los elementos del modelo usando una herramienta facilitada por Vitis-AI en la nueva versión. Esta herramienta interactuará con todos los nodos de la red neuronal y por medio de distintas herramientas la cuantizará usando unos parámetros para que pierda la menor precisión posible.

Debido a los resultados de los entrenamientos del Copyleft, solo pudimos obtener a través del primer método el ONNX de YOLOv6m. Por otra parte, ahora explicaremos como hemos obtenido los modelos ONNX usando la herramienta proporcionada por Vitis para procesar directamente cualquier modelo ONNX.

Lo primero sería la preparación del entorno para usarlo. Para ello debemos abrir cualquiera de los dockers de Vitis e instalarle las herramientas de ONNX. Tras abrir el docker nos dirigiremos a los archivos de fuente de las herramientas de Vitis-AI e instalar las herramientas.

```
cd src/vai_quantizer/vai_q_onnx
sh build.sh
pip install pkgs/*.whl
```

Dado que para el aprendizaje del uso de esta herramienta también hicimos uso de un ejemplo que ofrecen para ver cómo funciona, decidí instalar también los requerimientos del ejemplo

```
cd example/resnet_ptq_cpu/
```

```
python -m pip install -r requirements.txt
```

Después de instalar todas las librerías necesarias me desplacé al espacio de trabajo donde tengo el script y los modelos ONNX. Aquí es donde se encuentra el código que desarrollé, ONNX_quantizer.py Anexo 9. ONNX_quantizer.py. Este código recibirá indicaciones a través de los argumentos de llamado al script y ejecutará los pasos necesarios para la cuantización, que son:

1. Inicialización del modelo:
2. Preparación del paquete de datos, con imágenes y etiquetas
3. Cuantización del modelo

Para cuantizar los modelos deberemos ejecutar el siguiente script:

```
ONNX_quantizer.py modelo.onnx modelo_cuantizado.onnx
datasets/coco128/images/ --calib_method NonOverflow --
quant_format QDQ
```

Nota: Los apartados en negrita si no se indican tendrán una configuración por defecto

Descripción de los argumentos:

- `input_model`: El primer argumento siempre será reconocido como el modelo a cuantizar, y este deberá estar en formato ONNX.
- `output_model`: El segundo argumento será el modelo de salida. Aquí podremos especificar la ruta relativa y el nombre final del archivo, pero debemos añadir `.onnx` al final siempre.
- `--calib_method`: Define el método de calibración, en concreto, ajusta el rango de los valores de salida de las neuronas. Las opciones disponibles son:
    - **NonOverflow**: Ajusta el valor de forma que el máximo no desborde, en el proceso de cuantización puede provocar una pérdida en la precisión si los valores de activación no están homogéneamente distribuidos.
    - **MinMSE**: Ajusta el valor de forma que minimice el error cuadrático entre los valores cuantizados y el original. Puede probar desbordamiento en algunas activaciones en los extremos, pero asegura una mayor precisión para distribuciones no homogéneas.
- `--quant_format`: Define la forma de cuantización, que en ONNX se refiere a donde y como se hubicaran los operadores de cuantización/descuatización. Las opciones disponibles son:
    - **FixNeuron**: Cuantización tradicional, convierte los valores de los tensores en números enteros en la salida de cada capa. Este sería el más eficiente en hardware.
    - **QO**: Delante de las neuronas agrega un nodo de cuantización, pero no añade ninguno de descuantización, por lo que si el modelo se ha obtenido con algún framework que no lo permita, puede fallar. Este formato requiere que todas las funciones puedan trabajar con números enteros, está optimizado para las DPUs y las TPUs.

- **QDQ**: Como el anterior solo que añade adicionalmente el nodo de decuantización después de la neurona. Esto mantiene la compatibilidad con más frameworks, pero aumenta las demandas de hardware.
- **VitisQDQ**: Es una versión de QDQ especialmente optimizada para su despliegue en la DPU.

Como resultado final de este proceso, cuantizamos diferentes modelos pero no pudimos comprobar si estos habían sido correctamente cuantizados, más allá del mensaje del cuantizador de VitisAI. Estos fueron los resultados cuantizados:

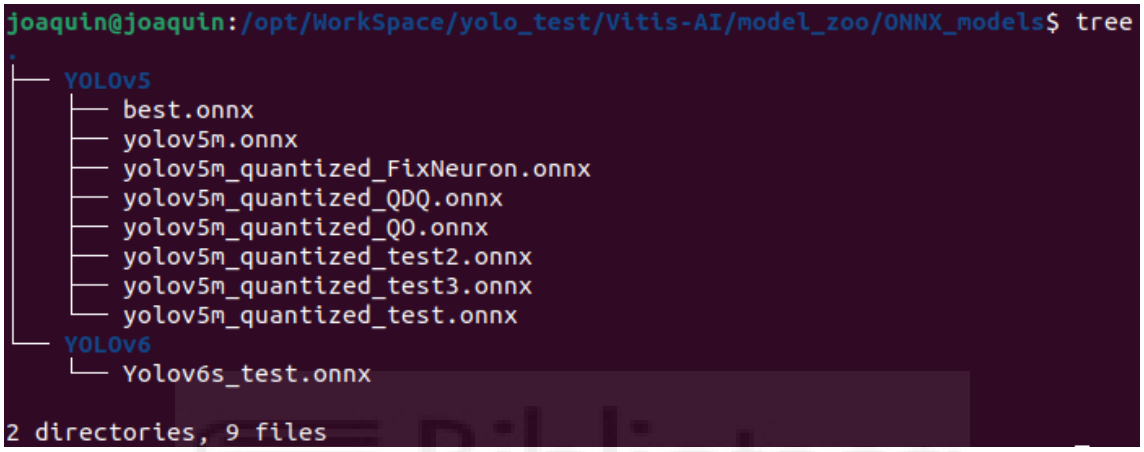

Figura 60. Modelos ONNX cuantizados.

CAPÍTULO 5: PUESTA EN MARCHA

5.1. PREPARACIÓN DE LA TARJETA

Este apartado tiene como objetivo detallar los pasos necesarios para preparar la conexión con la tarjeta y configurarla para su arranque mediante la tarjeta SD.

En primer lugar, es fundamental configurar la tarjeta para que cargue su configuración desde la tarjeta SD, dado que nuestro chip, una FPGA, pierde su configuración al interrumpirse el suministro eléctrico. Esto se debe a la naturaleza volátil de las FPGAs, que requieren cargar su configuración desde un dispositivo maestro cada vez que se reinician. En nuestro caso, dicha configuración se encuentra almacenada en la tarjeta SD, la cual debe ser replicada por la FPGA durante el proceso de arranque.

Para establecer el comportamiento maestro-esclavo necesario para esta configuración, es necesario ajustar un conjunto de interruptores ubicados en la esquina de las conexiones UART. Estos interruptores deben ser configurados físicamente en el modo de arranque adecuado: en **SW3**, **switch 1=OFF** y **switch 2=ON**.

Una vez configurado el modo de arranque, se debe insertar la tarjeta SD en su ranura correspondiente, asegurándose de que quede completamente fijada mediante el mecanismo de rebote al final. A continuación, se conecta la tarjeta al ordenador utilizando el adaptador previamente mencionado en la sección de materiales. Si la conexión se realiza correctamente, se encenderá una luz amarilla que indica que el microUSB y el cable están correctamente conectados al ordenador.

Por último, se conecta la fuente de alimentación al puerto **J10** y se activa el botón de encendido **SW4**, lo que iniciará el arranque de la tarjeta. Al principio del arranque habrá unos LEDs encendidos de color verde, tras pulsar el botón de arranque se encenderán unos de color rojo y cuando este finalice quedarán encendido unos de color azul y otros de color verde.

5.2. CONEXIÓN A LA TARJETA

Para establecer una conexión con la tarjeta, se utilizó **MobaXterm**, configurando inicialmente una conexión en serie. Los pasos realizados son los siguientes:

1. **Conexión en Serie**
    - Se inició una conexión serial especificando los parámetros de transferencia establecidos previamente durante la configuración del sistema. Desde MobaXTerm será una conexión COM USB Serial con una velocidad de 115200 bits por segundo.
    - Se introdujo el usuario y contraseña predeterminados de **Petalinux** (usuario: `root`, contraseña: `root`).

2. **Configuración de la Red WiFi**
    - Una vez conectados en serie, se utilizó el editor **nano** para modificar el archivo `wpa_supplicant.conf`. Este archivo contiene la configuración de la red WiFi, incluyendo:
        - El nombre de la red (SSID).
        - La clave de acceso (contraseña).
3. **Establecimiento de la Conexión WiFi**
    - Tras guardar los cambios en el archivo, se ejecutó el script `wifi.sh`, diseñado para intentar establecer la conexión con la red configurada.
    - Si la conexión es exitosa, el sistema mostrará la dirección IPv4 asignada al chip.
4. **Conexión SSH**
    - Con la dirección IPv4 obtenida, se inició una nueva conexión SSH desde MobaXterm. Para ello:
        - Se especificó la dirección IPv4 como destino.
        - Se introdujo el usuario y contraseña predeterminados de **Petalinux**.
    - Una vez autenticados, la conexión con la tarjeta quedó establecida, permitiendo acceso completo al sistema.

Este proceso asegura una conexión estable y funcional con la tarjeta, facilitando la comunicación y el control remoto del dispositivo.

5.3. PRUEBA DE LA PLATAFORMA HARDWARE

Con la conexión realizada correctamente, lo siguiente será comprobar si la unidad DPU está instalada correctamente y es de las dimensiones esperadas. Para ello ejecutaremos el siguiente comando y comprobaremos de la siguiente manera los distintos resultados.

Lo primero será llamar a la herramienta que comprueba el estado de la DPU que viene con el software instalado en Petalinux `xdputil query`.

El resultado que nos tiene que salir es el siguiente:

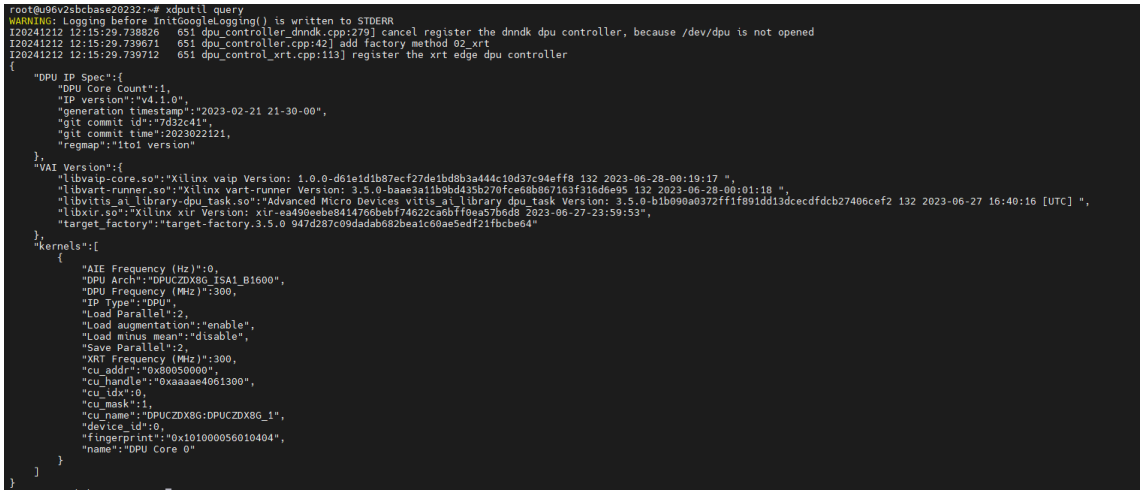

Figura 61. Resultado si la DPU está correctamente integrada.

La primera vez que llegamos a este punto se nos olvidó activar los controladores de la DPU en el proyecto de Petalinux y el resultado que nos salía era el siguiente:

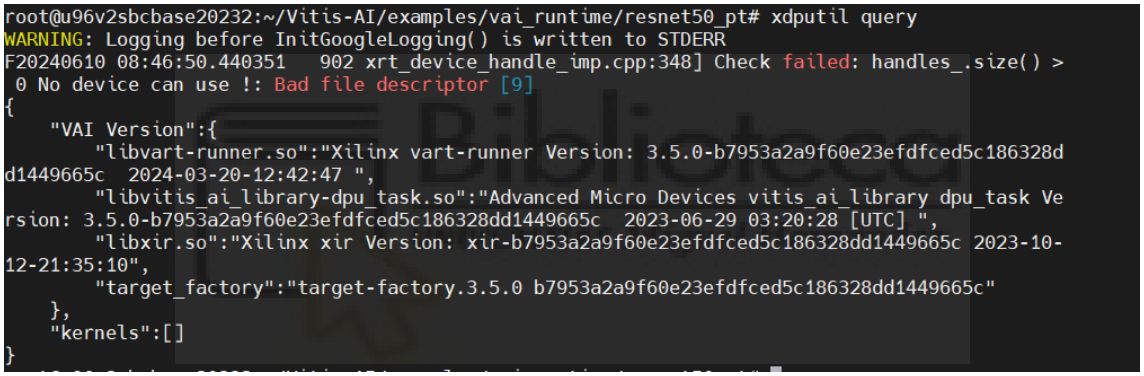

Figura 62. Resultados si a la DPU le faltan los controladores.

Como se puede comprobar al ver las diferencias, en primer lugar, no nos indica las especificaciones de la generación de la DPU, y en segundo lugar y más importante, no nos indica el núcleo ni sus especificaciones. Otro detalle que podemos observar es en VAI Version, que podemos ver que hay una librería más instalada, libvaip_core-so.

5.4. PRUEBA DE LOS MODELOS DEL MODEL ZOO DE VITIS AI

Para probar la **DPU** (Deep Processing Unit) con los modelos del **Vitis AI Model Zoo**, se realizaron los siguientes pasos:

1. **Descarga del Repositorio**
    - Se descargó el repositorio de GitHub de **Vitis AI**, específicamente el apartado de **examples**, que contiene los modelos y scripts necesarios para las pruebas.
    - Usando **MobaXterm**, se transfirió la carpeta descargada a la tarjeta, ubicándola en la ruta `/home/root`.

2. **Preparación de los Modelos**

    - Se descargaron y compilaron todos los modelos del Model Zoo utilizando un script adaptado y el archivo `arch.json`, que define las características de la DPU instalada.
    - El script generó una carpeta llamada **vitis-ai-library**, que se copió a la ruta `/usr/share/vitis-ai-library` en la tarjeta. Aunque no es estrictamente necesario seguir esta convención, se mantuvo por coherencia con las guías oficiales.
3. **Compilación y Ejecución del Código**

    - Dentro de la carpeta de ejemplos ubicada en `/home/root/VitisAI-examples/vai_runtime/`, se seleccionó el modelo **ResNet50** para las pruebas. Esta elección se basó en su potencia y capacidad para probar adecuadamente el rendimiento de la DPU.
    - Dentro de la subcarpeta **resnet50_pt/** se localizaron los archivos necesarios:
        - `build.sh`: Script para compilar el código.
        - `resnet50_pt.cpp`: Código fuente para la ejecución del modelo.
        - `word_list.inc`: Lista de palabras correspondientes a las clases.
    - Para compilar el modelo, se ejecutó el siguiente comando:

```
bash -x build.sh
```

4. **Prueba del Modelo Compilado**

    - Según las instrucciones del archivo **README.md** ubicado en **/vai_runtime/**, se ejecutó el modelo proporcionándole una imagen de entrada.
    - Para ello, se descargó previamente un conjunto de imágenes a la tarjeta, utilizado como dataset de pruebas.
    - El comando para ejecutar el modelo fue:

```
./resnet50_pt
/usr/share/vitis_ai_library/models/resnet50_pruned_0_3_pt/resnet50_pruned_0_3_pt.xmodel /home/root/dataset_tests/flor.jpg
```

    - Al ejecutar el modelo, se verificaron los resultados en pantalla, confirmando el correcto funcionamiento de la DPU con las capacidades del modelo ResNet50.

```
root@u96v2sbcbase20232:~/Vitis-AI-examples/vai_runtime/resnet50_pt# ./resnet50_pt
 /usr/share/vitis_ai_library/models/resnet50_pruned_0_3_pt/resnet50_pruned_0_3_pt
.xmodel /home/root/dataset_tests/flor.jpg
El valor de input_scale es: 32
Shape del tensor: 1 1000
score[738]  =  0.325601     text: pot, flowerpot,
score[883]  =  0.325601     text: vase,
score[94]   =  0.0565809    text: hummingbird,
score[985]  =  0.0267269    text: daisy,
score[716]  =  0.0267269    text: picket fence, paling,
```

Figura 63. Resultados de ResNet50_PT para la figura 64.

Figura 64. Imagen de prueba para la detección de objetos.

Esta prueba nos sirvió como caso representativo para evaluar el funcionamiento del hardware con la DPU, así como la correcta integración entre los modelos y la DPU.

5.5. IMPLEMENTACIÓN DE LOS MODELOS YOLO EN FORMATO XMODEL

5.5.1. Introducción, estructura y flujo de trabajo

En este apartado, utilizaremos los modelos YOLOvX obtenidos durante nuestro desarrollo. Para implementarlos, será necesario crear un código específico que se adapte a cada base de datos utilizada en los modelos. En el caso de aquellos modelos que hemos descargado (por no haber podido entrenarlos), se asumirá que han sido entrenados con el mismo conjunto de datos y, por lo tanto, presentan sesgos similares. Sin embargo, para nuestro modelo YOLOv6, el posprocesamiento tendrá que ser diseñado de manera más flexible.

5.5.1.1. Explicación de los Segmentos del Código y Fuentes de Inspiración

La implementación de los modelos YOLOvX requiere la creación de un código que se adapte tanto a los tensores de entrada como de salida, así como al posprocesamiento, que dependerá de los outputs generados por el modelo.

El proceso seguido para desarrollar las aplicaciones necesarias para implementar los modelos incluyó las siguientes etapas:

1. **Análisis de los códigos de ejemplo**
2. **Planteamiento de una estructura base**
3. **Implementación por bloques y depuración**
4. **Integración de los bloques y obtención de resultados finales**

5.5.1.2. Análisis de Códigos Existentes

El primer paso fue analizar el código de ejemplo probado previamente en nuestra tarjeta, el **ResNet50_pt**. Este código presentaba una estructura claramente definida en etapas:

- Arranque del modelo con comprobaciones iniciales
- Creación de los tensores
- Preprocesamiento
- Sincronización de inputs y outputs
- Posprocesamiento
- Visualización de resultados

La estructura del código era sencilla, y en general, su diseño resultaba limpio y manejable gracias al uso de pocas funciones adicionales, lo que ayudaba a evitar que la estructura principal resultara engorrosa.

Posteriormente, revisamos el código de **adas_detection**, diseñado para YOLOv3. Aunque no pudimos probar este código directamente, era relevante porque compartía similitudes con los modelos que íbamos a utilizar. Sin embargo, detectamos que este código estaba excesivamente complicado debido a la intención de integrarlo con video en lugar de imágenes estáticas. Para ello, implementaba un sistema de hilos (`threads`) y exclusiones mutuas (`mutex`) con el objetivo de procesar el análisis del video de manera semiparalela, lo cual buscaba mantener la fluidez del video.

El principal aporte de este código fue conceptual: su enfoque hacia el preprocesamiento y el posprocesamiento. No obstante, decidimos no reutilizarlo directamente, ya que su diseño no era eficiente ni claro.

5.5.1.3. Estructura Propuesta para Nuestro Código

Tras analizar los códigos anteriores, diseñamos una estructura para nuestra implementación. Dado que este sería nuestro primer código, optamos por seguir un esquema similar al del **ResNet50_pt**, ya que las pruebas iniciales y la curva de aprendizaje serían más manejables trabajando primero con imágenes, en lugar de video.

5.5.2. Preparación del código

5.5.2.1. Primer Bloque: Lectura de argumentos y deserialización del modelo

El primer paso en la implementación del código es manejar los argumentos de entrada, deserializar el modelo YOLO, y verificar la disponibilidad de unidades DPU. En esta etapa, se realiza la creación de un **runner**, que permitirá acceder y gestionar los tensores de entrada y salida necesarios para ejecutar la unidad DPU.

```
int main(int argc, char* argv[]) {

 if (argc < 3) {
  std::cout << "usage: " << argv[0]
        << " <yolovx.xmodel> sample_image [sample_image ...] \n";
  return 0;
 }

 auto xmodel_file = std::string(argv[1]);
 std::vector<cv::Mat> input_images;

 for (auto i = 2; i < argc; i++) {
  cv::Mat img = cv::imread(argv[i]);
  if (img.empty()) {
   std::cout << "Cannot load image : " << argv[i] << std::endl;
   continue;
  }
  input_images.push_back(img);
 }

 if (input_images.empty()) {
  std::cerr << "No image load success!" << std::endl;
  abort();
 }

 // Crear runner para DPU
 auto graph = xir::Graph::deserialize(xmodel_file);
 auto root = graph->get_root_subgraph();
 xir::Subgraph* subgraph = nullptr;
 for (auto c : root->children_topological_sort()) {
  CHECK(c->has_attr("device"));
  if (c->get_attr<std::string>("device") == "DPU") {
   subgraph = c;
   break;
  }
 }
 if (subgraph == nullptr) {
 std::cerr << "Error: No se encontró un subgrafo DPU." << std::endl;
 return -1;
 }

 auto attrs = xir::Attrs::create();
 std::unique_ptr<vart::RunnerExt> runner =
   vart::RunnerExt::create_runner(subgraph, attrs.get());
```

```
if (!runner) {
std::cerr << "Error: No se pudo crear el runner." << std::endl;
return -1;
}
```

5.5.2.2. Segundo Bloque: Creación y configuración de tensores

El siguiente paso consiste en crear los tensores necesarios para el funcionamiento del modelo. Esto incluye tanto los tensores de entrada como los de salida. Primero se debe crear el tensor correspondiente a los de entrada, ya que solo debería haber uno, y posteriormente para los tensores de salida. Durante el proceso de depuración de esta sección del código, se requirió un considerable esfuerzo de prueba y error hasta identificar la causa de la estructura de los tensores y comprender las diferencias entre estos para cada modelo. Además, cada tensor tendrá asociada una **escala**, dado que todos los valores han sido transformados de formato de coma flotante a enteros con una precisión de 8 bits. Será necesario obtener la escala de cada tensor para poder operar adecuadamente con los valores cuando se envíen o extraigan.

Las conclusiones obtenidas después del proceso de prueba y error fueron las siguientes: Los tensores de entrada, en todos los modelos, son iguales, con una forma de 1 640 640 3, donde la primera dimensión corresponde al batch. Sin embargo, dado que los modelos cuantizados solo pueden trabajar con una imagen a la vez, no es necesario verificar que siempre habrá un valor de 0 en esa dimensión. Las segunda y tercera dimensiones definen las filas y columnas de las imágenes de entrada, y, dado que todos los modelos proporcionados por los códigos de Vitis están formateados para trabajar con imágenes cuadradas de 640 píxeles, estas dimensiones indican las filas y columnas a las que se hace referencia. Finalmente, la tercera dimensión corresponde al número de canales, siendo 3 en este caso, ya que YOLO trabaja con imágenes en formato BGR.

En cuanto a los tensores de salida, la diversidad es mayor, pero hay un aspecto claro: todos los modelos preparados por Vitis trabajan con 3 capas o "layers". Por lo tanto, para una imagen de 640 píxeles cuadrada, siempre se elegirían rejillas de 80, 40 y 20. De este modo, los tensores de salida tendrán alguna de las siguientes formas: 1 80 80 X, 1 40 40 X o 1 20 20 X. En estas formas, el primer valor corresponde a la imagen batch, el segundo y tercero corresponden a la celda de la rejilla, y lo restante corresponde a los resultados de las predicciones.

El problema surge con las diferencias entre YOLOv6 y las versiones anteriores, como YOLOv7 y YOLOv4. En el caso de YOLOv6, las transformaciones de los tensores de salida están sobrecargadas con operadores de transmutación u operaciones no válidas, lo que, al cuantizarlo, segmenta el proceso hacia la CPU, dejando los tensores rotos y no en un formato final de salidas. Algo similar ocurre con YOLOv4 y YOLOv7, y parece ser que en la versión del modelo que hemos obtenido se ha actualizado el API de PyTorch responsable de gestionar los encabezados de transmutación, sin considerar que, en algunos casos, debe desviarse el proceso hacia la CPU.

Los tensores de salida finales para cada modelo son los siguientes:

- **YOLOv4 y YOLOv7**: 1 80 80 255, 1 40 40 255 y 1 20 20 255. El valor 255 se compone de 3 anclas de 85, donde 85 a su vez se descompone en 4+1+80. Los primeros 4 valores definen las dimensiones y posición de la caja, el quinto valor corresponde a la confianza de que haya un objeto en esa caja, y los 80 valores restantes corresponden a la confianza de cada clase.
- **YOLOv6**: 1 80 80 80, 1 80 80 4, 1 40 40 80, 1 40 40 4, 1 20 20 80 y 1 20 20 4. En este caso, se pierden las anclas y las cajas se colocan en un tensor diferente. Además, no existe la puntuación de confianza de la detección de un objeto en una celda, sino solo la confianza por clase, lo que complica considerablemente el posprocesamiento, aunque este aspecto se abordará en bloques posteriores.

```
//ENTRADAS
auto inputTensors = runner->get_input_tensors();
auto inputTensorBuffers = runner->get_inputs();

//SALIDAS
auto outputTensors = runner->get_output_tensors();
auto outputTensorBuffers = runner->get_outputs();
```

5.5.2.3. Tercer Bloque: Preprocesamiento de Imágenes

El preprocesamiento de imágenes es una etapa fundamental para preparar los datos de entrada antes de enviarlos al modelo. En esta parte del código, es necesario decidir si se procesarán varias imágenes en serie o no. En mi caso, inicialmente no implementé este paso en el código de prueba, pero después de concluir el proceso de depuración, decidí que sería útil procesarlas de manera secuencial, por lo que se añadió dicha funcionalidad.

Este bloque consiste en la creación de un objeto que replicará las propiedades del tensor de entrada, permitiendo la transferencia de datos mediante un puntero. Posteriormente, se procederá a pasar la información al tensor de entrada. Para lograr esto, se utilizarán tuplas, que deben sincronizarse para que apunten correctamente al interior del tensor. Además, en esta sección del código se implementarán dos funciones clave:

1. **Redimensionamiento de la Imagen**: Esta función se encargará de ajustar las dimensiones de la imagen, ya sea mediante reducción o ampliación, y rellenará los bordes con un color gris para asegurar que la imagen cumpla con los requisitos del modelo.
2. **Normalización de los Valores RGB**: La segunda función se encargará de procesar la imagen píxel por píxel. Para cada píxel, se convertirá el valor RGB extraído de la imagen a un valor normalizado según la escala de entrada del tensor, ordenando los valores en formato BGR, que es el estándar requerido por el modelo.

Cuando los valores se envíen al tensor, se representarán como objetos del tipo 'signed int', con el número de bits determinado por la escala o por la información obtenida al analizar el modelo en herramientas como Netron. En los casos trabajados en C++, se utilizó un tamaño de 8 bits (int8_t).

```
auto itb = inputTensorBuffers[0];
```

```
 int input_tensor_size = inputTensors[0]->get_element_num();
 if (input_tensor_size <= 0) {
    std::cerr << "Error: Tensor 0 está vacío!" << std::endl;
 }

 // Revisamos para el tensor de entrada el componente fixed_point para la función setImageRGB
 auto it = itb->get_tensor();
 auto input_scale = get_input_scale(it);

 auto batch = inputTensors[0]->get_shape().at(0);
 auto height = inputTensors[0]->get_shape().at(1);
 auto width = inputTensors[0]->get_shape().at(2);

 for (auto i = 0; i < input_images.size(); i += batch) {
        auto run_batch = std::min(((int)input_images.size() - i), batch);
        auto images = std::vector<cv::Mat>(run_batch);

        // Preprocesamiento
        std::vector<uint64_t> data_in(run_batch);
        std::vector<size_t> size_in(run_batch);
        //Preparar los colores de la imagen
        for (auto batch_idx = 0; batch_idx < run_batch; ++batch_idx) {
                images[batch_idx] = preprocess_image(input_images[i + batch_idx], cv::Size(width, height));
                std::tie(data_in[batch_idx], size_in[batch_idx]) = inputTensorBuffers[0]->data({batch_idx, 0, 0, 0});
                CHECK_NE(size_in[batch_idx], 0u);
                setImageRGB(images[batch_idx], (void*)data_in[batch_idx], input_scale);
        }
```

5.5.2.4. Cuarto Bloque: Sincronización y Ejecución en el Runner

En este bloque, se sincronizan los objetos tensor y se envían al modelo para su ejecución utilizando el *runner*. Además, se realiza una comprobación del estado del *runner* para asegurarse de que la ejecución se haya realizado correctamente. Posteriormente, se sincronizan los tensores de salida para evitar la pérdida de datos.

El flujo de trabajo en este bloque es el siguiente:

1. **Sincronización de Tensores de Entrada**: Se sincronizan los tensores de entrada con la CPU/DPU para asegurarse de que los datos estén listos para ser procesados.
2. **Ejecución del Modelo**: El modelo se ejecuta de manera asincrónica mediante el uso del *runner*. Se verifica que la ejecución haya sido exitosa mediante la comprobación del estado del *runner*.
3. **Sincronización de Tensores de Salida**: Después de la ejecución, se sincronizan los tensores de salida para asegurar que los resultados se puedan leer correctamente sin pérdida de información.

```
//Sincronizamos el input con la CPU/DPU para que lo reciba
```

```
        for (auto& input : inputTensorBuffers) {
                input->sync_for_write(0,    input->get_tensor()->get_data_size()    /
input->get_tensor()->get_shape()[0]);
        }

        auto v = runner->execute_async(inputTensorBuffers, outputTensorBuffers);
        auto status = runner->wait(v.first, -1);
                CHECK_EQ(status, 0) << "fallo en la ejecución de DPU";

        for (auto& output : outputTensorBuffers) {
                output->sync_for_read(0, output->get_tensor()->get_data_size() / output-
>get_tensor()->get_shape()[0]);
        }
```

5.5.2.5. Quinto Bloque: Formateo de tensores de salida

El quinto bloque tiene como objetivo darle formato a los tensores de salida de modo que el posprocesamiento sea más cómodo y eficiente. En esencia, este bloque debe tomar la salida del runner, formatear los valores de salida aplicando la escala y la función sigmoide si es necesario. No obstante, debido a la fragmentación de los tensores en el código de YOLOv6, fue necesario desarrollar una estrategia para organizar los datos y emparejar los tensores de clases y los de cajas antes de enviarlos al posprocesamiento.

Al principio del bloque se crea una tupla con los objetos necesarios para apuntar a los valores de un tensor. A continuación, los valores se extraen y se almacenan en una estructura de datos que facilita su manejo. Para ello, se creó una estructura denominada `TensorData`, en la cual se almacenan las dimensiones del tensor y una matriz con la misma estructura del tensor que contiene los valores.

```
std::vector<TensorData> class_probabilities_vec;
std::vector<TensorData> boxes_vec;

        for (size_t idx = 0; idx < outputTensorBuffers.size(); ++idx) {
                auto tb = outputTensorBuffers[idx];
                int tensor_size = outputTensors[idx]->get_element_num();

                if (tensor_size <= 0) {
                        std::cerr << "Error: Tensor " << idx << " está vacío!" << std::endl;
                        continue;
                }

                std::uint64_t data_addr;
                std::tie(data_addr, std::ignore) = tb->data({0, 0, 0, 0});

                if (data_addr == 0) {
                        std::cerr << "Error: Dirección de datos vacía para el tensor " << idx
<< std::endl;
                        continue;
                }
```

```
                int8_t* data_ptr = (int8_t*)(data_addr);

                // Obtener las dimensiones del tensor
                auto shape = outputTensors[idx]->get_shape();
                auto ot = outputTensors[idx];
                int first_dim = shape.front();
                int last_dim = shape.back();
                int second_dim = shape[shape.size() - 2];
                int third_dim = shape[shape.size() - 3];

                // Estructura para almacenar los datos con su forma original
                TensorData tensor_data;
                tensor_data.shape = shape;
                tensor_data.data.resize(first_dim);
                for (int ii = 0; ii < first_dim; ++ii) {
                        tensor_data.data[ii].resize(second_dim);
                                for (int j = 0; j < second_dim; ++j) {
                                        tensor_data.data[ii][j].resize(third_dim);
                                        for (int k = 0; k < third_dim; ++k) {
                                                tensor_data.data[ii][j][k].resize(last_dim);
                                        }
                                }
                }
                // Copiar los datos del tensor en la estructura manteniendo la forma original
                int index = 0;
                for (int ii = 0; ii < first_dim; ++ii) {
                        for (int j = 0; j < second_dim; ++j) {
                                for (int k = 0; k < third_dim; ++k) {
                                        for (int l = 0; l < last_dim; ++l) {

                                                //Usando la escala convertimos el valor int8_t a float
                                                // Aplicar Sigmoid a cada valor del tensor y multiplicar por la escala
                                                float output_scale = get_output_scale(ot);
                                                float value = data_ptr[index++] * output_scale;

                                                value = sigmoid(value);
                                                tensor_data.data[ii][j][k][l] = value;
                                        }
                                }
                        }
                }

                // Clasificar los tensores
                if (last_dim == 80) {
                    class_probabilities_vec.push_back(tensor_data);
                } else if (last_dim == 4 || last_dim == 5) {
```

```
                        boxes_vec.push_back(tensor_data);
                }
        }
//EMPAREJAMIENTO PARA YOLOv6
        std::vector<std::tuple<TensorData, TensorData>> tensor_pairs;

        for (size_t i = 0; i < class_probabilities_vec.size(); ++i) {
                auto& class_probs = class_probabilities_vec[i];
                auto& boxes = boxes_vec[i];

                // Nos aseguramos de que las dimensiones coincidan
                if (class_probs.shape[1] == boxes.shape[1]  &&  class_probs.shape[2]
== boxes.shape[2]) {
                        tensor_pairs.push_back(std::make_tuple(class_probs, boxes));
                } else {
                        std::cerr  <<  "Error:  Dimensiones  no  coinciden  entre
probabilidad de clases y boxes." << std::endl;
                }
        }
```

*En negrita estarán marcadas las parte para YOLOv6

5.5.2.6. Sexto Bloque: Posprocesamiento

El último bloque de código implementa el proceso de **posprocesamiento**, que está diseñado para ser genérico y flexible. Por ello, se optó por encapsular este proceso en una función independiente. A continuación, se describe cómo se lleva a cabo este proceso:

1. **Extracción de Cajas:** En primer lugar, se realiza un barrido por todas las celdas de la rejilla para cada tensor. Para cada celda, se evalúa si la clase detectada supera el umbral de confianza establecido. En caso afirmativo, se extraen las coordenadas de la caja delimitadora en formato `y_center`, `x_center`, `width` y `height`. Este formato es transformado posteriormente en las dimensiones adecuadas de la caja.
    - Para determinar las dimensiones de las cajas, se consultan los archivos de entrenamiento de YOLO en busca de sesgos por defecto. Si están definidos, se incorporan al proceso de posprocesamiento. En caso contrario, se crea una operación para generar dimensiones aproximadas que se ajusten a las características deseadas. Esta operación será evaluada mediante pruebas con diferentes imágenes para garantizar que ofrece resultados consistentes y adecuados.
2. **Redimensionamiento de Cajas:** Una vez extraídas las coordenadas, se verifica que las dimensiones de las cajas sean válidas (es decir, que el ancho y la altura sean mayores que 1). Las cajas válidas se almacenan en un vector junto con la clase correspondiente y la confianza asociada.
3. **Redimensionamiento dentro de los límites de la imagen:** Después de obtener las cajas, se aplican correcciones para asegurarse de que todas las cajas estén dentro de los límites de la imagen. Si alguna de las cajas se extiende fuera de los límites (por ejemplo, si su coordenada superior es negativa o su coordenada inferior

excede el tamaño de la imagen), se ajustan las posiciones y tamaños para que se ajusten a los límites de la imagen.

4. **Aplicación de Non-Maximum Suppression (NMS):** Con el conjunto de cajas válidas obtenidas, se procede a la aplicación de un filtro de **supresión de no máximos** (NMS, por sus siglas en inglés). Este algoritmo elimina las cajas redundantes que se solapan excesivamente, conservando únicamente la caja con la mayor confianza. El umbral de superposición (IoU) para la supresión se ajusta según las necesidades del modelo.

5. **Código para YOLOv6, YOLOv4 y YOLOv7:** El código está adaptado para trabajar con distintas versiones de YOLO, considerando las diferencias en la forma de procesar las salidas de los tensores. Cada versión maneja sus propios valores de *stride*, *biases* y el formato de las coordenadas de la caja. El proceso general de detección sigue siendo el mismo, con la inclusión de ajustes específicos para cada versión de YOLO.

6. **Resultados Finales:** Al finalizar el proceso, se obtiene un conjunto de cajas delimitadoras que representan las ubicaciones de los objetos detectados. Este conjunto de resultados se puede usar para tareas posteriores, como la visualización de las detecciones o su procesamiento adicional en aplicaciones específicas.

YOLOv6

```
std::vector<cv::Rect> boxes;
   std::vector<int> class_ids;
   std::vector<float> confidences;
   std::vector<std::string> class_names = {"PERSON", "BICYCLE", "CAR", "MOTORCYCLE",
"AIRPLANE", "BUS", "TRAIN", "TRUCK", "BOAT", "TRAFFIC LIGHT", "FIRE HYDRANT", "STOP
SIGN", "PARKING METER", "BENCH", "BIRD", "CAT", "DOG", "HORSE", "SHEEP", "COW",
"ELEPHANT", "BEAR", "ZEBRA", "GIRAFFE", "BACKPACK", "UMBRELLA", "HANDBAG", "TIE",
"SUITCASE", "FRISBEE", "SKIS", "SNOWBOARD", "SPORTS BALL", "KITE", "BASEBALL BAT",
"BASEBALL GLOVE", "SKATEBOARD", "SURFBOARD", "TENNIS RACKET", "BOTTLE", "WINE
GLASS", "CUP", "FORK", "KNIFE", "SPOON", "BOWL", "BANANA", "APPLE", "SANDWICH",
"ORANGE", "BROCCOLI", "CARROT", "HOT DOG", "PIZZA", "DONUT", "CAKE", "CHAIR",
"COUCH", "POTTED PLANT", "BED", "DINING TABLE", "TOILET", "TV", "LAPTOP", "MOUSE",
"REMOTE", "KEYBOARD", "CELL PHONE", "MICROWAVE", "OVEN", "TOASTER", "SINK",
"REFRIGERATOR", "BOOK", "CLOCK", "VASE", "SCISSORS", "TEDDY BEAR", "HAIR DRIER",
"TOOTHBRUSH"};
   int counter=0;
   std::vector<int> biases = {10, 13, 36, 75, 142, 110};

   for (const auto& pair : tensor_pairs) {
      const TensorData& class_probs = std::get<0>(pair);
      const TensorData& boxes_tensor = std::get<1>(pair);

      // Determinar el stride según las dimensiones del tensor de cajas
           int stride = (boxes_tensor.shape[1] == 80) ? 8 : (boxes_tensor.shape[1] == 40) ? 16 :
(boxes_tensor.shape[1] == 20) ? 32 : 1;
    int anch = (boxes_tensor.shape[1] == 80) ? 0 : (boxes_tensor.shape[1] == 40) ? 1 :
```

```
(boxes_tensor.shape[1] == 20) ? 2 : -1;

    for (int i = 0; i < class_probs.shape[0]; ++i) {
      for (int j = 0; j < class_probs.shape[1]; ++j) {
        for (int k = 0; k < class_probs.shape[2]; ++k) {
          for (int c = 0; c < nc; ++c) {
            float confidence = class_probs.data[i][j][k][c];
            if (confidence > confidence_threshold) {
              if (confidence == 1) counter++;

              // Extraer las coordenadas de la caja que están en un formato diferente del
habitual:
              //y_center, x_center, width y height

              float y = (sigmoid(boxes_tensor.data[i][j][k][0]) + j) * stride;
              float x = (sigmoid(boxes_tensor.data[i][j][k][1]) + k) * stride;

              //float w = (boxes_tensor.data[i][j][k][2]) * stride * std::exp(1);
              //float h = (boxes_tensor.data[i][j][k][3]) * stride * std::exp(1);
              //Test
              //float w = std::exp(sigmoid(boxes_tensor.data[i][j][k][2]))*biases[anch];
              float w = std::pow(sigmoid(boxes_tensor.data[i][j][k][2])*(std::exp(1)-
anch/4),2)*stride;
              //float h = std::exp(sigmoid(boxes_tensor.data[i][j][k][3]))*biases[anch+1];
              float h = std::pow(sigmoid(boxes_tensor.data[i][j][k][3])*(std::exp(1)-
anch/4),2)*stride;
              //



              float left = (x - w / 2);
              float top = (y - h / 2);


              // Comprobar si las dimensiones son válidas (ancho y alto mayores que 1)
              if (w >= 1 || h >= 1) {

                boxes.emplace_back(cv::Rect(left, top, w, h));
                float max_class_score = class_probs.data[i][j][k][c];
                int class_id = c;

                // Encontrar la clase con el mayor puntaje
                for (int jj = 0; jj < nc; ++jj) {
                  if (class_probs.data[i][j][k][jj] > max_class_score) {
                    max_class_score = class_probs.data[i][j][k][jj];
                    class_id = jj;  // Ajustar índice según el formato de salida de YOLO
                  }
                }

                class_ids.push_back(class_id);
                confidences.push_back(confidence);
```

```
                }
              }
            }
          }
        }
      }
    }

    //filtro de cajas
    for(auto& box : boxes){

      float top = box.y;
      float left = box.x;
      float bottom = box.y + box.height;
      float right = box.x + box.width;

      if (top < 0) top = 0;
      if (left < 0) left = 0;
      if (bottom > 640) bottom = 640;
      if (right > 640) right = 640;

      box.y = top;
      box.x = left;
      box.height = bottom - top;
      box.width = right - left;

    }

    // Aplicar NMS (supresión de no máximos) para eliminar detecciones redundantes
    std::vector<int> indices;
    NMSBoxesPerClass(boxes, confidences, class_ids, confidence_threshold, 0.2, indices,
nc);
```

YOLOv4 y YOLOv7

```
std::vector<cv::Rect> boxes;
    std::vector<int> class_ids;
    std::vector<float> confidences;
    std::vector<std::string> class_names = {"PERSON", "BICYCLE", "CAR", "MOTORCYCLE",
"AIRPLANE", "BUS", "TRAIN", "TRUCK", "BOAT", "TRAFFIC LIGHT", "FIRE HYDRANT", "STOP
SIGN", "PARKING METER", "BENCH", "BIRD", "CAT", "DOG", "HORSE", "SHEEP", "COW",
"ELEPHANT", "BEAR", "ZEBRA", "GIRAFFE", "BACKPACK", "UMBRELLA", "HANDBAG", "TIE",
"SUITCASE", "FRISBEE", "SKIS", "SNOWBOARD", "SPORTS BALL", "KITE", "BASEBALL BAT",
"BASEBALL GLOVE", "SKATEBOARD", "SURFBOARD", "TENNIS RACKET", "BOTTLE", "WINE
GLASS", "CUP", "FORK", "KNIFE", "SPOON", "BOWL", "BANANA", "APPLE", "SANDWICH",
"ORANGE", "BROCCOLI", "CARROT", "HOT DOG", "PIZZA", "DONUT", "CAKE", "CHAIR",
"COUCH", "POTTED PLANT", "BED", "DINING TABLE", "TOILET", "TV", "LAPTOP", "MOUSE",
"REMOTE", "KEYBOARD", "CELL PHONE", "MICROWAVE", "OVEN", "TOASTER", "SINK",
"REFRIGERATOR", "BOOK", "CLOCK", "VASE", "SCISSORS", "TEDDY BEAR", "HAIR DRIER",
"TOOTHBRUSH"};
    int counter=0;
```

```
  for (const auto& tensor : outputTensors) {

  // Determinar el stride según las dimensiones del tensor de cajas (En principio no hay)
  int stride = (tensor.shape[1] == 80) ? 8 : (tensor.shape[1] == 40) ? 16 : (tensor.shape[1] ==
20) ? 32 : 1;
   int aux = (tensor.shape[1] == 80) ? 0 : (tensor.shape[1] == 40) ? 1 : (tensor.shape[1] ==
20) ? 2 : -1;
  std::vector<float> biases{12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192,
243, 459, 401};

    for (int i = 0; i < tensor.shape[0]; ++i) {
      for (int j = 0; j < tensor.shape[1]; ++j) {
        for (int k = 0; k < tensor.shape[2]; ++k) {
          for (int anch = 0; anch < tensor.shape[3]/(nc+5); ++anch){

            float confidence = sigmoid(tensor.data[i][j][k][4+(nc+5)*anch]); //Por lo general
en YOLO va despues de las dimensiones de las cajas
            if (confidence > confidence_threshold) {
              // Extraer las coordenadas de la caja que están en formato
              //x_center, y_center, width y height
              float y1 = (sigmoid(tensor.data[i][j][k][0+(nc+5)*anch]) + j) * stride;
              float x1 = (sigmoid(tensor.data[i][j][k][1+(nc+5)*anch]) + k) * stride;
              float w1 = std::exp(sigmoid(tensor.data[i][j][k][2+(nc+5)*anch])) *
biases[aux*6+anch*2];
              float h1 = std::exp(sigmoid(tensor.data[i][j][k][3+(nc+5)*anch])) *
biases[aux*6+anch*2+1];
              float left1 = (x1 - w1 / 2);
              float top1 = (y1 - h1 / 2);

              // Comprobar si las dimensiones son válidas (ancho y alto mayores que 4)
              //if ((w1 <= 4 || h1 <= 4) || (w1/h1>4 || h1/w1>4)) continue;
              float max_score = 0;
              int class_id = -1;
              for (int c = 4+(nc+5)*anch; c < 84+(nc+5)*anch; ++c) {
                float class_confidence = sigmoid(tensor.data[i][j][k][c]);
                if (class_confidence > max_score){
                  max_score = class_confidence;
                  class_id = c - 5 - (nc+5)*anch;
                }
              }

              if(class_id >= 0) {

                //Guardamos la caja como objecto Rect
                boxes.emplace_back(cv::Rect(left1, top1, w1, h1));
                //Guardamos la id de la clase y su confianza para el print
                class_ids.push_back(class_id);
                confidences.push_back(confidence*max_score);
              }
            }
```

```
            }
          }
        }
      }
    }

    //filtro de cajas
    for(auto& box : boxes){

      float top = box.y;
      float left = box.x;
      float bottom = box.y + box.height;
      float right = box.x + box.width;

      if (top < 0) top = 0;
      if (left < 0) left = 0;
      if (bottom > 640) bottom = 640;
      if (right > 640) right = 640;

      box.y = top;
      box.x = left;
      box.height = bottom - top;
      box.width = right - left;

    }

    // Aplicar NMS (supresión de no máximos) para eliminar detecciones redundantes
    std::vector<int> indices;
    NMSBoxesPerClass(boxes, confidences, class_ids, confidence_threshold, 0.05,
indices, nc);
```

Al final del posprocesamiento se formatearán los resultados finales para adaptarlos a los formatos de salida disponibles. Aquí el código detectará si el dispositivo de salida puede imprimir imágenes o no, e imprimirá un texto con las detecciones, y si puede, una imagen con cajas de distintos colores para cada clase.

```
std::vector<cv::Scalar> colors;
    for (int i = 0; i < nc; ++i) {
      int r = (i * 37) % 255;  // Cálculo cíclico para rojo
      int g = (i * 73) % 255;  // Cálculo cíclico para verde
      int b = (i * 7) % 255;  // Cálculo cíclico para azul
      colors.emplace_back(cv::Scalar(b, g, r));
    }


    if (canDisplayImages()) {
      std::cout << "Ha detectado que puede imprimir imágenes" << std::endl;
      for (int idx : indices) {
        cv::rectangle(image, boxes[idx], colors[class_ids[idx]] , 2);
        std::string label = class_names[class_ids[idx]]+ " " +
cv::format("%.1f",confidences[idx]);
```

```
        int baseline = 0;
        cv::Size textSize = cv::getTextSize(label, cv::FONT_HERSHEY_SIMPLEX, 0.5, 1,
&baseline);
        int rect_top = std::max(boxes[idx].y - textSize.height - 5, 0);
        int rect_left = boxes[idx].x;
        int rect_width = textSize.width + 10;
        int rect_height = textSize.height + 5;
        cv::rectangle(image, cv::Point(rect_left, rect_top), cv::Point(rect_left + rect_width,
rect_top + rect_height), colors[class_ids[idx]], cv::FILLED);
        cv::putText(image, label, cv::Point(rect_left + 5, rect_top + textSize.height),
cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255, 255, 255), 1);
      }
      printDetections(boxes, confidences, indices, class_ids);
      cv::imshow("Detecciones", image);
      cv::waitKey(0);
    } else {
      std::cout << "Ha detectado que no puede imprimir imágenes" << std::endl;
      printDetections(boxes, confidences, indices, class_ids);
    }
```

5.5.3. Funciones adicionales

Entre las funciones adicionales nos encontraremos funciones básicas como extraer el atributo de 'fix-point' para emplearlo como escala para transformar los valores enteros en valores de coma flotante, a funciones más detalladas como NMSCustom y NMSPerClass, que son variantes del filtro NMS responsable de elegir las cajas delimitadoras de los objetos con un criterio u otro.

Las funciones adicionales se desarrollaron con la finalidad de tener un código más limpio y estructurado. Esas funciones son las siguientes.

5.5.3.1. get_input_scale y get_output_scale

Son funciones que son esencialmente iguales, ya que extraen el valor 'fix-point', sin embargo, a la hora de devolver el valor, el valor se convierte en una potencia de 2. La diferencia aquí es si el valor se obtiene de elevar 2 al valor positivo o al negativo, quedándonos resultados inversos. Este se debe a la naturaleza que van a recibir, pues un valor entero que quiero volver coma flotante se multiplicará por la escala, mientras que un valor flotante que quiero volver un número entero con las restricciones de escala, tendré que dividirlo. Por esta razón, con intención de simplificar el código y simplemente ver cómo se multiplica con la escala y quedará el resultado deseado.

```
// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor) {
  return tensor->has_attr("fix_point") ? std::exp2f(1.0f * tensor->get_attr<int>("fix_point"))
: 1.0f;
}

static float get_output_scale(const xir::Tensor* tensor) {
      return tensor->has_attr("fix_point") ? std::exp2f(-1.0f * tensor-
```

```
>get_attr<int>("fix_point")) : 1.0f;
}
```

5.5.3.2. setImageRGB

Esta función fue una de las más problemáticas de desarrollar, pues los códigos de ejemplo que se usaron tenían funciones muy diferentes y aunque se intentaran aplicar, estas eran deficientes.

```
static void setImageRGB(const cv::Mat& image, void* data, float fix_scale) {
  int8_t* data1 = (int8_t*)data;
  int c = 0;
  for (int row = 0; row < image.rows; row++) {
    for (int col = 0; col < image.cols; col++) {
      auto v = image.at<cv::Vec3b>(row, col);
      auto B = (float)v[0];
      auto G = (float)v[1];
      auto R = (float)v[2];

      // Normalización
      auto nB = B / 255.0f * fix_scale;
      auto nG = G / 255.0f * fix_scale;
      auto nR = R / 255.0f * fix_scale;

      // Clamping
      nB = std::max(std::min(nB, fix_scale), 0.0f);
      nG = std::max(std::min(nG, fix_scale), 0.0f);
      nR = std::max(std::min(nR, fix_scale), 0.0f);

      data1[c++] = (int8_t)(nR);
      data1[c++] = (int8_t)(nG);
      data1[c++] = (int8_t)(nB);
    }
  }

  //print_input_image(data1);

}
```

5.5.3.3. NMSPerClass y NMSCustom

Estas funciones se encargarán de hacer un barrido por las distintas detecciones en base a los mismos criterios, pero en diferente orden. Los criterios en común, es que, si dos cajas apuntan a la misma clase de objeto, se comprobará si estás cajas tienen cierta superposición entre ellas, ese grado de superposición se contrastará con un umbral preestablecido y si se supera, se quedará solo la caja con mayor confianza. La diferencia radica en cómo se ordenan las cajas para su filtrado, en NMSPerClass, las cajas de ordenan por clases y luego se aplica un bucle para revisarlas, en NMSCustom se ordenan por

confianza y luego se comparará la primera de la lista con todas las demás buscando primero las que sean de la misma clase y luego el grado de superposición.

```
void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& class_ids, float confidence_threshold, float iou_threshold, std::vector<int>& indices, int nc) {

  // Iterar por cada clase
  for (int c = 0; c < nc; ++c) {
    std::vector<int> class_indices;

    // Recoger índices de la clase actual
    for (size_t i = 0; i < class_ids.size(); ++i) {
      if (class_ids[i] == c && confidences[i] > confidence_threshold) {
        class_indices.push_back(i);
      }
    }

    // Ordenar por confianza
    std::sort(class_indices.begin(), class_indices.end(),
          [&](int i, int j) { return confidences[i] > confidences[j]; });

    while (!class_indices.empty()) {
      int best_index = class_indices[0];
      indices.push_back(best_index);
      class_indices.erase(class_indices.begin());

      for (auto it = class_indices.begin(); it != class_indices.end();) {
        int idx = *it;
        float iou = IoU(boxes[best_index], boxes[idx]);

        if (iou > iou_threshold) {
          it = class_indices.erase(it);
        } else {
          ++it;
        }
      }
    }
  }
}

void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences,
            float confidence_threshold, float iou_threshold, std::vector<int>& indices) {

  // Filtrar las cajas basadas en el umbral de confianza
  std::vector<int> initial_indices;
  for (size_t i = 0; i < confidences.size(); ++i) {
    if (confidences[i] >= confidence_threshold) {
      initial_indices.push_back(i);
    }
  }
```

```
  // Ordenar las cajas filtradas por puntuación de confianza descendente
  std::sort(initial_indices.begin(), initial_indices.end(),
        [&](int i, int j) { return confidences[i] > confidences[j]; });

  // NMS
  while (!initial_indices.empty()) {
    int current_idx = initial_indices.front();
    indices.push_back(current_idx);
    initial_indices.erase(initial_indices.begin());

    auto it = initial_indices.begin();
    while (it != initial_indices.end()) {
      int idx = *it;
      float iou = IoU(boxes[current_idx], boxes[idx]);
      if (iou > iou_threshold) {
        it = initial_indices.erase(it);
      } else {
        ++it;
      }
    }
  }
}
```

5.5.3.4. IoU

Está función será la responsable de calcular el grado de superposición, que se calculará como intersección de las áreas sobre la unión de las áreas. Definiendo las cajas, se buscarán los puntos que definen el área de intersección y luego se calculará la suma de las áreas de las detecciones y se le restará la intersección para obtener el área de unión.

```
float IoU(const cv::Rect& box1, const cv::Rect& box2) {
  // Suponiendo que Box tiene las propiedades x1, y1, x2, y2
  float x1_intersection = std::max(box1.x, box2.x);
  float y1_intersection = std::max(box1.y, box2.y);
  float x2_intersection = std::min(box1.x + box1.width, box2.x + box2.width);
  float y2_intersection = std::min(box1.y + box1.height, box2.y + box2.height);

  // Calcular el área de intersección
  float intersection_area = std::max(0.0f, x2_intersection - x1_intersection) *
                std::max(0.0f, y2_intersection - y1_intersection);

  // Calcular el área de las cajas
  float area1 = box1.width * box1.height;
  float area2 = box2.width * box2.height;

  // Calcular el área de unión
  float union_area = area1 + area2 - intersection_area;
```

```
    // Evitar división por cero
    if (union_area <= 0) return 0.0f;

    // Retornar IoU
    return intersection_area / union_area;
}
```

5.5.3.5. CanDisplayImages

CanDisplayImages será la responsable de comprobar si el sistema donde se ejecute tiene la capacidad de representar una imagen con el formato de salida que estaremos utilizando, en este caso OpenCV. Lo hará por medio de intentar generar una plantilla donde imprimir imágenes y luego comprobar si su existencia no es nula, en caso de serlo devolverá un valor booleano negativo.

```
bool canDisplayImages() {
    //std::cout << "Esta comprobando si puede usar el display" << std::endl;
    try {
        cv::namedWindow("test", cv::WINDOW_AUTOSIZE);
        cv::destroyWindow("test");
        //return false;
        return true;
    } catch (const cv::Exception& e) {
        std::cout << "No se pueden mostrar imágenes en este entorno." << std::endl;
        return false;
    }
}
```

5.5.3.6. printDetections

Esta función deberá representar las detecciones en un formato de texto comprensible por el lector, dado que se trata de detecciones que no incluyen detecciones pose, o segmentaciones, esto es particularmente fácil ya que solo se deberá incluir la clase detectada, la confianza y la caja delimitadora, también se incluirá un número que indique el total de detecciones realizadas

```
void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& indices, const std::vector<int>& class_ids) {
    std::vector<std::string> class_names = {"person","bicycle","car","motorcycle","airplane","bus","train","truck","boat","traffic light","fire hydrant","stop sign","parking meter","bench","bird","cat","dog","horse","sheep","cow","elephant","bear","zebra","giraffe","backpack","umbrella","handbag","tie","suitcase","frisbee","skis","snowboard","sports ball","kite","baseball bat","baseball glove","skateboard","surfboard","tennis racket","bottle","wine glass","cup","fork","knife","spoon","bowl","banana","apple","sandwich","orange","broccoli","carrot","hot dog","pizza","donut","cake","chair","couch","potted plant","bed","dining table","toilet","tv","laptop","mouse","remote","keyboard","cell phone","microwave","oven","toaster","sink","refrigerator","book","clock","vase","scissors","teddy bear","hair drier","toothbrush"};
    std::cout << indices.size() << " objetos detectados:\n";
    for (auto& indx : indices) {
```

```
      std::cout << "Clase: " << class_names[class_ids[indx]]
            << ", Confianza: " << confidences[indx]
            << ", Caja: [" << boxes[indx].x << ", " << boxes[indx].y << ", "
            << boxes[indx].x + boxes[indx].width << ", " << boxes[indx].y + boxes[indx].height
<< "]\n";
   }
}
```

5.5.3.7. Sigmoid

```
float sigmoid(float x) {
   return 1.0f / (1.0f + std::exp(-x));
}
```

5.5.4. Mejoras del código

En cuanto a modificar el código para trabajar con vídeo, hay problemas con la gestión de memoria, el error se debe a la falta de capacidad de display por imagen, pues se comunica a través de SSH para poder representar los resultados. Esto provoca una latencia entre la salida del modelo y la entrada del siguiente fotograma que satura la gestión de memoria del dispositivo. Sin embargo, para evitar la saturación del vídeo se puede activar la webcam, tomar un solo fotograma y apagar la webcam, de modo que esa imagen sea la procesada y nos muestre sus detecciones. Con ese método podemos obtener una velocidad máxima de 1 fotograma por cada 8 segundos, que es difícil considerarlo un vídeo, viendo la imagen con las detecciones, pero si quitamos la posibilidad de emitir imágenes y solo emite texto, podremos obtener una velocidad de 1 fotograma por segundo. También se puede grabar un vídeo y ejecutar el modelo sobre él y que este vaya capturando fotogramas, pero la velocidad de muestreo es bastante inestable, sin embargo, ronda un fotograma por cada 4 segundos. Por último, también sería posible dejar la capturadora siempre encendida y anular la salida de imagen.

Los fragmentos de código necesarios giran en torno a la herramienta de openCV y te permiten hacer uso de distintos protocolos para abrir la capturadora. Un ejemplo sería la función captureSingleFrame desarrollada para enviar un dispositivo y devolver un fotograma capturado y el tiempo de latencia estimado para el dispositivo.

```
std::tuple<cv::Mat, int> captureSingleFrame(std::string videoFile) {
   // Abre la capturadora
   cv::VideoCapture video(videoFile); //cv::VideoCapture video(videoFile, cv::CAP_V4L2);
   if (!video.isOpened()) {
      std::cerr << "Error: No se pudo abrir la cámara." << std::endl;
      return {cv::Mat(), 0}; // Devuelve un fotograma vacío
   }

   // Captura un fotograma
   cv::Mat frame;
   if (!video.read(frame)) {
      std::cerr << "Error: No se pudo capturar un fotograma." << std::endl;
      return {cv::Mat(),0}; // Devuelve un fotograma vacío
   }
```

```
    // Obtener delay
    int delay = static_cast<int>(10 / video.get(cv::CAP_PROP_FPS));

    // Cierra la capturadora
    video.release();
    return {frame, delay}; // Devuelve el fotograma capturado
}
```

Y otra manera de gestionarlo sería llamar a la capturadora al principio del código donde antes leíamos las imágenes pasadas como argumento, y crear un bucle que englobe todo desde el preprocesamiento hasta el posprocesamiento donde en cada iteración capture un fotograma y lo mande a la secuencia principal. Esto queda reflejado en el Anexo 12. YOLO_camera_demo.cpp donde la versión comentada corresponde a mantener la capturadora activa permanentemente.

5.6. IMPLEMENTACIÓN DE LOS MODELOS YOLO EN FORMATO ONNX

En otros frentes, como la integración de los modelos usando ONNX Runtime en la tarjeta, aún no se han conseguido obtener resultados a falta de un modelo cuantizado en formato ONNX para contrastar. Para preparar el entorno como indica la guía de usuario de VitisAI (UG1414), te pide que instales en el dispositivo objetivo los paquetes facilitados por ellos. Para hacerlo de forma directa, recurrí a una guía publicada en hackster.io (Misoji, 2024) cuyo objetivo es instalar todo correctamente además de generar unos archivos de ejemplo para distintos modelos.

Los pasos por seguir serían los siguientes:

1. Preparar el entorno de instalación:

```
cd /home/root/
mkdir VOE_Install
cd VOE_Install/
```

2. Descargar el **VOE** (Vitis AI ONNXRuntime Engine) y descomprimir:

```
wget https://www.xilinx.com/bin/public/openDownload?filename=vitis_ai_2023.1-r3.5.0.tar.gz
sudo tar -xzvf openDownload\?filename\=vitis_ai_2023.1-r3.5.0.tar.gz -C /
ls
```

3. Descarga e instalación de los instaladores:

```
wget https://www.xilinx.com/bin/public/openDownload?filename=voe-0.1.0-py3-none-any.whl -O voe-0.1.0-py3-none-any.whl
pip3 install voe*.whl
wget https://www.xilinx.com/bin/public/openDownload?filename=onnxruntime_vitisai-1.16.0-py3-none-any.whl -O onnxruntime_vitisai-1.16.0-py3-none-any.whl
```

```
pip3 install onnxruntime_vitisai*.whl
```

Tras completar los pasos la instalación ya estaría realizada y sería visible en `/usr/share/vitis-ai-library/` donde habrán aparecido carpetas.

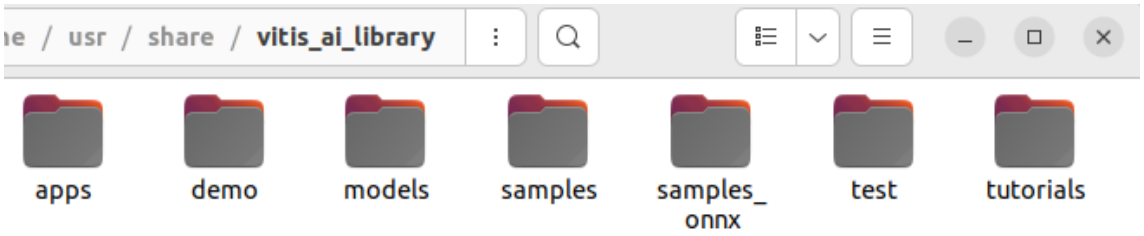

Figura 65. Vitis-ai-library actualizada.

En este punto, si dispusiéramos de más tiempo hubiésemos podido explorar y desarrollar una aplicación para probar los modelos ONNX en la Ultra96v2. Sin embargo, he optado por añadir este apartado porque me parece un camino interesante a retomar en los trabajos futuros.

CAPÍTULO 6: ENTRENAMIENTO PARA LA DETECCIÓN DE ARTEFACTOS MÉDICOS

En este capítulo se desarrolla el proceso seguido para obtener el modelo de YOLO entrenado con imágenes médicas para la detección de artefactos en ecografías de seno siguiendo el criterio de BI-RADS para definir la peligrosidad del tumor detectado. Para poder llevar a cabo el entrenamiento se necesitaría una base de datos de alta fidelidad que nos indique los artefactos con la mayor precisión posible. A continuación, se llevará a cabo el entrenamiento del modelo YOLOv6m usando el repositorio de Xilinx para obtener el modelo final en el formato XModel para introducirlo al dispositivo. Finalmente, en el dispositivo se pondrá a prueba la precisión del modelo y de las cajas de detección.

6.1. PREPARACIÓN DE LA BASE DE DATOS

Para obtener un conjunto de fotos médicas desarrollado para el entrenamiento de inteligencia artificial se revisaron numerosas fuentes oficiales de distintos países y diversos estudios o proyectos cuya finalidad era crear una base de datos con este propósito. En este último grupo era raro tener acceso a esa base de datos de forma remota, sin embargo, se encontró publicada dentro de la web estadounidense National Institute of Health (NIH) en el apartado dedicado a la investigación al cancer National Cancer Institute (NCI) en uno de estudios dentro del programa Cancer Imaging Program (CIP) publicado en 2024 (Anna Pawłowska, 2024).

Esta base de datos te ofrecía las imágenes clasificadas usando el criterio de BI-RADS, ofrecía 256 casos que habían seguido el criterio registrado en un Excel anexo con las máscaras, que son archivos que indican por segmentación la posición y forma exacta de los tumores detectados. Adicionalmente, hay agregado el seguimiento de los resultados finales de la exploración y biopsias de las masas dando la posibilidad de entrenar el modelo sin un criterio médico a expensas de que el propio proceso de iteraciones del modelo genere su propio criterio.

En la base de datos descargada te encuentras una carpeta donde te encontrarás los archivos tal que:

- **Imagen médica**: estará indicada como `caseXXX.png`.
- **Objetos detectados**: se indicará de dos maneras diferentes, una para el tumor de mayor riesgo y otra para las otras detecciones que se encuentren en la imagen:
    - **Objetos de riesgo**: `caseXXX_tumor.png`.
    - **Objetos adicionales**: `caseXXX_otherX.png`.

A continuación, el trabajo consistiría en convertir las imágenes y los resultados del Excel en un formato con el repositorio de Xilinx pueda trabajar. Para ello sería necesario extraer de las máscaras las cajas y del Excel la clase equivalente en BI-RADS. Para esto se emplearon diversos scripts de Python, las funciones que cumplieron fueron:

1. Clasificar las imágenes: repartir las imágenes en los directorios `images/` y `masks/` en función al nombre.
2. Crear las etiquetas con el formato COCO: revisando la carpeta de `masks/` usando OpenCV convertirá los objetos de segmentación en cajas de detección gracias a que se tratan de imágenes negras con la segmentación marcado en blanco.
3. Modificar las etiquetas para que indiquen correctamente el objeto al que apuntan: para esto hace falta extraer del Excel una lista en un archivo `.txt` donde los objetos a los que apunta cada máscara para cada imagen médica tengan un BI-RADS asignado. También hizo falta revisar algunos objetos de la lista de objetos adicionales debido a que no se veían reflejados en la lista de objetos (en su mayoría se trataban de objetos comunes como las costillas, el pulmón o un implante estético), Luego con un script revisar las etiquetas y cambiar las clases.

Con todo eso terminado tendríamos el conjunto de datos en formato COCO completado. Para terminar solo hizo falta crear un YALM para la interpretación de datos por el modelo para su aprendizaje.

6.2. PREPARACIÓN, ENTRENAMIENTO Y PRUEBA DEL MODELO

Con la intención de hacer más breve el entrenamiento se desarrollaron los archivos de configuración del modelo para que este se llevara a cabo en el formato pequeño. Para hacer esto tenemos que fijarnos en los archivos de configuración utilizados para la versión mediana, especialmente para completar las distintas etapas del entrenamiento. Para la primera etapa estaba listo entonces fue fácil coger una referencia. Usando los archivos para YOLOv6m solo tuve que ver a que partes del modelo llamaban al entrenamiento y con que herramienta lo hacían y mantener la configuración de los pesos indicada en el primer archivo de configuración que te ofrecían. También estaba la opción usando la función de activación LeakyReLU y también se desarrollaron sus archivos de configuración.

Antes de entrenar el modelo con las ecografías se le entrenó con COCO128 para valorar la calidad de las detecciones cuando se usaba la red de menor tamaño. Estos resultados son visiblemente peores que el del formato de tamaño medio. Fue por este motivo por el que decidí entrenar el modelo con el tamaño medio.

Finalmente, en cuanto a los resultados del entrenamiento, indican una precisión muy elevada, con un **mAP de 98% en PTQ y 96% en QAT**, me temo que se debe al alto contraste que hay en las detecciones además del hecho que hay pocas imágenes para el entrenamiento y son las mismas empleadas para la validación. De todas formas, como se puede ver en el siguiente capítulo, las detecciones apuntan correctamente pero las cajas no parecen tener los tamaños deseados. Sin embargo, si se hubiese conseguido la implementación de ONNX se hubiese intentado implementar un entrenamiento con el módulo de segmentación para que detectase los objetos completos a la perfección. Los resultados obtenidos, en general, son satisfactorios dejan una buena base para un desarrollo posterior.

CAPÍTULO 7: RESULTADOS

En este capítulo demostraremos los resultados finales del proyecto, como lo son los modelos que hemos conseguido cuantizar, las aplicaciones desarrolladas para estos modelos y las detecciones de cada modelo en imágenes. En cuanto a las detecciones de vídeo con la webcam u otras comprobaciones lo mejor sería demostrar su funcionamiento en directo.

7.1. MODELOS

En este apartado explicaremos que modelos se han conseguido entrenar, que modelos se podrían haber conseguido con más tiempo o capacidad de hardware y que modelos podríamos haber obtenido con ayuda de un profesional. Finalmente habrá una categoría donde estarán los modelos en formato ONNX, dado que estos no se han conseguido comprobar como cuantizados.

Con modelos que hemos conseguido entrenar nos referiremos a aquellos modelos que hemos conseguido completar su entrenamiento en el Copyleft y que además hemos desarrollado un conjunto de datos específico para su aplicación médica. En este apartado solamente se encuentra el modelo YOLOv6, pero en este modelo se ha trabajado de diversas maneras y conseguido distintos resultados gracias al desempeño del estudiante. Los modelos son los siguientes:

1. **YOLOv6m entrenado con COCO128**: este modelo es el primero que obtuvimos usando el entrenamiento del Copyleft y el subconjunto de datos preparado por Ultralytics. Este modelo tendrá un desempeño significativamente peor que el de YOLOv6m preentrenado debido a la reducida cantidad de fotos con las que fue entrenado, sin embargo, con una mayor capacidad de hardware hubiéramos sido capaces de entrenarlo perfectamente con otros conjuntos de datos.
2. **YOLOv6s entrenado con COCO128**: este modelo sería un modelo obtenido creando unos archivos de configuración diferentes a los disponibles dentro del Copyleft con la intención de tener un modelo de menor tamaño y comprobar el desempeño del modelo en la tarjeta de Avnet. Este modelo requirió muchísimo menos tiempo de entrenamiento pese a usar el mismo conjunto de datos debido a la baja carga computacional que suponen los modelos ‘s’.
3. **YOLOv6m entrenado con ecografías de mamas**: este modelo sería un acercamiento directo al resultado final que se buscaba de este proyecto. Pese a que el proyecto está orientado a usar una nueva tecnología no disponíamos de nadie capaz de clasificar los distintos artefactos en las imágenes diagnósticas, sin embargo, optamos por hacer uso de ecografías ya que tienen unos criterios muy claros en cuanto a la diferenciación de los objetos. Este conjunto de datos fue desarrollado por el estudiante con la ayuda de un conjunto de datos publicado (Anna Pawłowska, 2024) y con los conocimientos propios del estudiante en el área médica de diagnóstico por imagen.

Dentro de la categoría de modelos que podrían haber sido obtenidos con más tiempo y/o capacidad de hardware nos encontramos con los modelos restantes del Copyleft. Estos

modelos los obtuvimos gracias a que el propio repositorio te daba acceso a una versión de estos entrenada por COCO ya en formato XModel. Estos modelos serían:

1. YOLOv4-csp
2. YOLOv6m
3. YOLOv7m

En los modelos que podríamos haber conseguido si hubiéramos tenido acceso a los archivos de configuración internos del modelo y conseguido configurar la capa 24. Estos modelos hubieran sido cuantizados a través de PTQ gracias a los scripts adaptados del tutorial de Mario Bergeron, pero dado la naturaleza de las capas fue difícil modificar suficiente los modelos como para obtener una única unidad de DPU o una unidad que recogiera, aunque sea hasta las capas finales de trasmutaciones para adaptar el código como se hizo con YOLOv6m. Aquí estarían:

1. YOLOv3
2. **YOLOv5**: Este modelo, pese a haber conseguido cuantizarlo no quedó en un formato útil o aplicable a nuestra plataforma dado que disponíamos solo de una unidad DPU.

Y finalmente, los modelos ONNX, aquí encontraremos dos categorías, los exportados junto al XModel que se ha conseguido entrenar con el Copyleft, y los modelos exportados en crudo del repositorio de Ultralytics y después han sido cuantizados con el script de cuantización para los modelos ONNX.

1. YOLOv6m con COCO128
2. YOLOv6s con COCO128
3. YOLOv6m con ecografías de seno
4. YOLOv5m con COCO128
5. YOLOv8m con COCO128
6. YOLOv11m con COCO128

7.2. APLICACIÓN

Las aplicaciones finales desarrolladas están divididas en subcategorías, así como lo están por el archivo del modelo también lo estarán por el tipo de entrada que tienen. Las aplicaciones que se han conseguido desarrollar son las siguientes:

1. **YOLOv6_APP**: opera con modelos en formato XModel y con imágenes. Anexo 10. YOLOv6_APP.cpp
2. **YOLOv4y7_APP**: opera con los modelos en formato XModel y con imágenes. Anexo 11. YOLOv4y7_APP.cpp
3. **YOLO_camera_demo**: opera con modelos en formato XModel y con imágenes obtenidas de la webcam. También dispone comentado como operar con vídeo, pero si se indica que la salida sea solo textual o si el dispositivo tiene conectada una salida de vídeo nativa y no por SSH. Anexo 12. YOLO_camera_demo.cpp

Estas aplicaciones nos permitieron comunicarnos con la DPU además de adaptar los archivos de entrada a la estructura de los tensores de entrada y la transformación de los datos de los tensores de salida en información interpretable por el usuario. En el desarrollo de estos códigos se le dio prioridad a la legibilidad e interpretación del código sobre a la eficiencia de este para facilitar la formación de nuevos desarrolladores.

7.3. DETECCIONES

Aquí podré una muestra de las imágenes obtenidas al comprobar las detecciones de los modelos, imágenes con detecciones de distintos tamaños, imágenes con variedad de objetos detectados e imágenes de las detecciones de los artefactos médicos.

Comparación de los modelos YOLOv6s, YOLOv6m con COCO, YOLOv4 y YOLOv7 en la detección de objetos de distintos tamaños con alto contraste

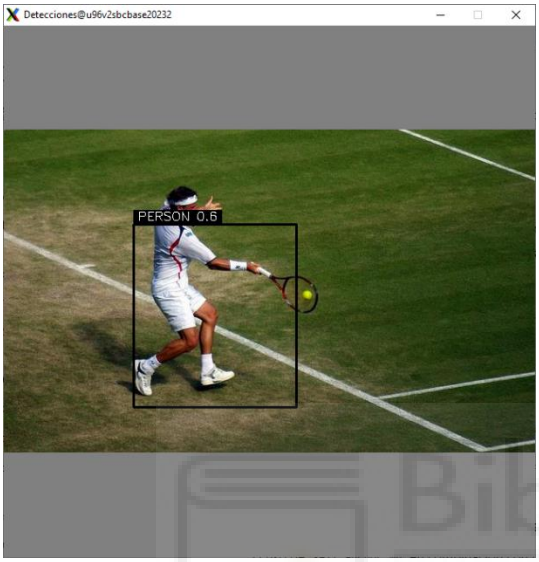

Figura 66. Tennist por YOLOv6s.

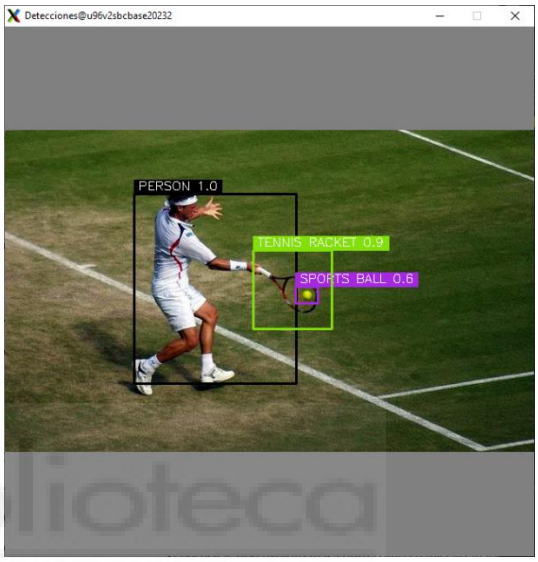

Figura 67. Tennist por YOLOv6m.

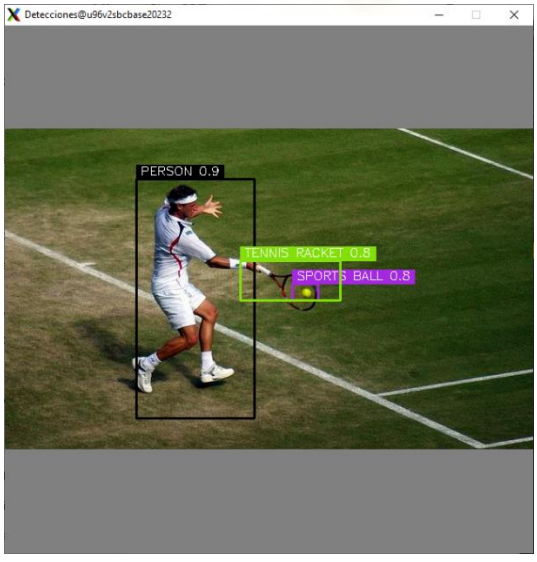

Figura 68. Tennis por YOLOv4.

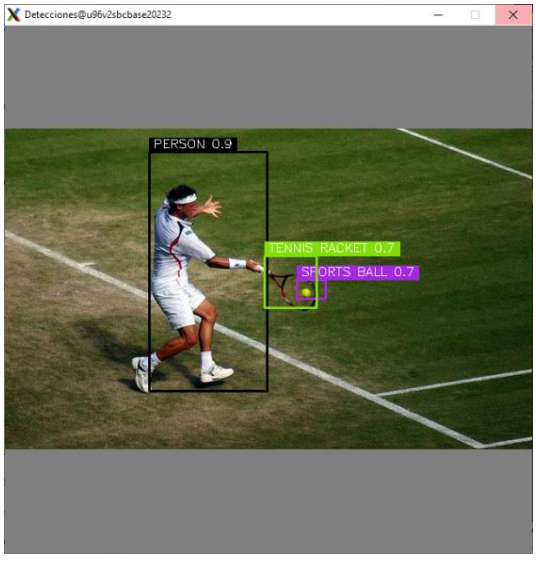

Figura 69. Tennist por YOLOv7.

En esta comparación somos capaces de observar la diferencia clara de capacidad entre nuestro modelo y los modelos preentrenados. Tanto en la calidad de la caja como en la

confianza en la detección, así como en la detección de los distintos objetos en una imagen con alto contraste

Comparación de los modelos YOLOv6s, YOLOv6m con COCO, YOLOv4 y YOLOv7 en la detección de múltiples objetos de diferentes clases y tamaños en una imagen mosaico.

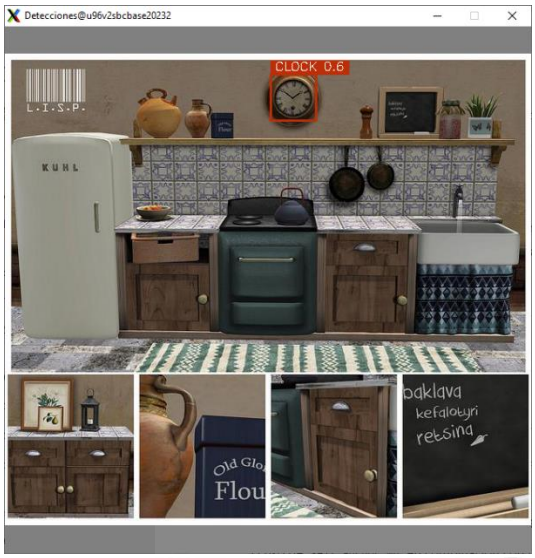

Figura 70. Kitchen por YOLOv6s.

Figura 71. Kitchen por YOLOv6m.

Figura 72. Kitchen por YOLOv4.

Figura 73. Kitchen por YOLOv7.

En estas imágenes podemos comprobar la eficiencia de los modelos preentrenados frente a nuestro modelo YOLOv6s para la detección de múltiples objetos de distintas clases, así como YOLOv6m parece tener una mayor capacidad de detectar más objetos aunque con una precisión de las cajas delimitadoras más baja que YOLOv7.

Comparación entre los modelos YOLOv6m QUANT y QAT en la detección de artefactos médicos, con una o dos clases.

Figura 74. BI-RADS 4c por QUANT.

Figura 75. BI-RADS 4c por QAT.

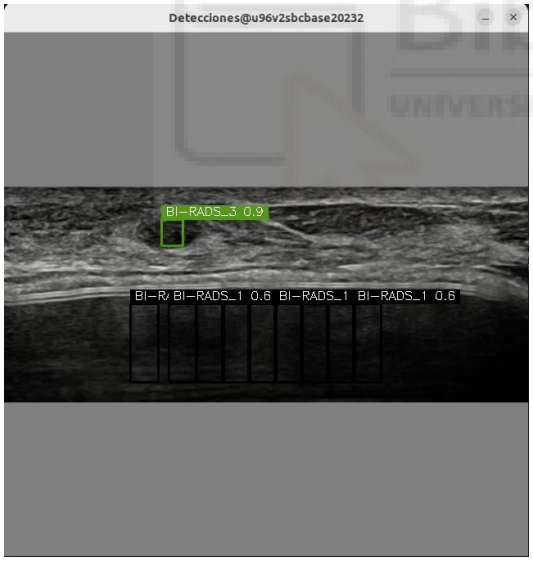

Figura 76. BI-RADS 1 y 3 por QUANT.

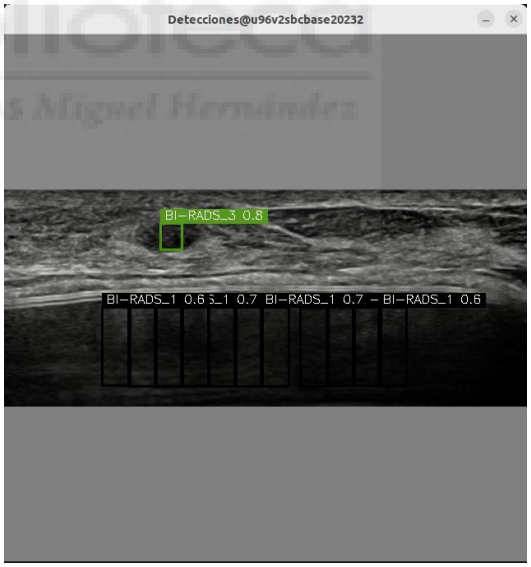

Figura 77. BI-RADS 1 y 3 por QAT.

Como se puede comprobar, los resultados son esencialmente idénticos a diferencia de la confianza en los objetos detectados, en los que QUANT tiene una mayor confianza. También parece haber un problema con los objetos más anchos, aunque esto es probable que se deba a la segmentación de las últimas capas de YOLOv6m.

CAPÍTULO 8: CONCLUSIONES

Los modelos que fueron entrenados con el hardware limitado y por ende usando COCO128 demuestran una pérdida significativa en la capacidad de hacer detecciones tras ser cuantizados y ejecutados en el dispositivo. Por otra parte, los modelos ofrecidos por Vitis-AI demuestran una capacidad de detección superior en lo referente a imágenes, concretamente YOLOv6m reconoce una mayor variedad de objetos y es capaz de reconocer personas en una mayor variedad de entornos. Por otra parte, en detección por vídeo, YOLOv4 y YOLOv7 tienen un mejor rendimiento a la hora de coger los objetos completos, aunque estos tengan menos confianza que las detecciones de YOLOv6. En cuanto a la comprobación de los resultados dentro y fuera del chip, solo fue posible con YOLOv6m ya que disponíamos del modelo ONNX generado durante el entrenamiento. En esta comprobación el funcionamiento dentro del chip salía perdiendo por una amplia diferencia, sobre todo en la cantidad y calidad de las detecciones. Esto se puede deber a que los modelos ONNX tengan un mejor rendimiento cuantizados que los modelos XModel, pues tras revisar las capas, éstas contienen la misma información pese a que la cuantización de ONNX sea mucho más superficial y pierde menos su estructura.

Estoy bastante satisfecho con lo que se ha conseguido de este proyecto, he aprendido mucho sobre el desarrollo de plataformas hardware, así como de diseño de sistemas operativos y sus distintos elementos. También he desarrollado mis conocimientos acerca del funcionamiento de la detección por imagen con YOLO, así como de los distintos métodos de entrenamiento y cuantización. Finalmente, donde más me he desarrollado a lo largo del proyecto ha sido en las habilidades informáticas, como la generación de un código como la exploración de distintas librerías y lenguajes, así como la investigación de los modelos cuantizados que en un principio eran como una caja negra que al finalizar el proyecto he conseguido desentrañar. Y, por último, las habilidades de depuración y compatibilización de hardware para poder llevar a cabo este proyecto en mi ordenador personal. Me doy por satisfecho con las habilidades de desarrollo de código que he conseguido, he avanzado en mis conocimientos con Python y PyTorch, y a su vez con los códigos en C++ con las librerías de VART y XIR.

Veo necesario para dar una conclusión a este proyecto mencionar los diferentes problemas que han surgido durante el desarrollo de este y que se han debido a la idiosincrasia de mi ordenador personal, ya que esto ha supuesto un gran impedimento en lo referente al desarrollo completo del trabajo y sobre todo al final alcanzado, pues con el mismo tiempo y un ordenador como el del laboratorio, hubiese podido avanzar mucho más rápido y quizá haber conseguido más avances. Uno de los problemas más complejos que tuve que abarcar fue un problema de compatibilidad entre la definición de la BIOS de los ordenadores ASUS con el Sistema Operativo Linux.

Esta descripción impedía el acceso a los distintos componentes del ordenador desde el arranque y además implicaba limitaciones en el consumo de energía o uso de la CPU, lo que hizo imposible el uso de todos sus núcleos en paralelo, debido a que esto causaba la muerte del proceso sin un mensaje de la aplicación o Linux. Esto hizo muy complicado el

uso de herramientas como Petalinux o Vitis donde no es fácil la configuración de los trabajos en paralelo que se ejecutan y tienen configurado por defecto hacer uso de todos los núcleos al 100%.

Otro de los problemas que generó este conflicto es que afectó a la instalación de Ubuntu 22.04.4 LTS ya que la instalación de paquetes esenciales para el funcionamiento de las herramientas, incluso de algunas nativas de Linux como 'Camera', fuese defectuoso desde el principio. Esto requirió el proceso de revisión manual de todas las herramientas utilizadas, así como la desinstalación de paquetes corrompidos para su reinstalación.

Por otra parte, también tuvimos complicaciones con Docker y NVIDIA, tan Roberto como yo tuvimos que hacer configuraciones adicionales para que reconociera la GPU desde dentro del contenedor. Estos pasos indicados en CAPÍTULO 4: FLUJO DE TRABAJO → OBTENCIÓN DE LAS REDES NEURONALES → Herramientas complementarias fue una guía completa siguiendo pasos estrictos para evitar todas las posibles complicaciones adicionales que experimentamos.

Adicionalmente, me gustaría hablar de las distintas facilidades proporcionadas por AMD, como Vitis 2023.2, que era imposible usarlo en su versión normal, y que posterior al lanzamiento, mediante un anexo al final de la guía de usuario, tuvieron que especificar que había quedado defectuosa la aplicación y que recomendaban el uso de versiones anteriores así como el uso responsable de la versión classic ya que esta también podía tener errores importantes, como pudimos comprobar durante el desarrollo de nuestro proyecto. Otra de las facilidades, aunque de forma indirecta sería la parte de la guía de Logitronix, que aunque revises todo el proyecto, tanto su repositorio como el repositorio publicado por Ultralytics del momento en el que desarrollaron la guía que usamos como referencia, queda claro que es imposible completar el proceso que sugieren y que además no tiene mucho fundamento algunas de las decisiones que tomaron como comentar la función que define la primera capa y es revisada en cada iteración después. De no ser por ayuda externa y esfuerzo personal hubiese sido imposible llegar al punto donde se cuantiza el modelo.

Finalmente, me gustaría concluir el apartado de dificultades encontradas con el trabajo que llevó todo el proceso de identificar las carencias del modelo YOLOv6m debido a la fragmentación de su última capa. Esto supuso que en el posprocesamiento no fuera capaz de identificar los distintos elementos, y de nuevo, de no ser por ayuda externa y esfuerzo personal, haber descubierto la presencia de los tensores fragmentados y luego interpretar que información nos aportaba cada uno hubiese sido imposible terminar su aplicación. Así como fue también revisar todos los proyectos publicados en el copyleft de forma minuciosa en búsqueda del correcto proceso de preprocesamiento de las imágenes para su interpretación por parte del modelo o la búsqueda de las anclas, pues YOLOv6m fue configurado con anclas variables y no predefinidas como YOLOv4 y YOLOv7, lo que agregó una nueva capa de complejidad.

Sin embargo, todos estos contratiempos fueron tomados más como desafíos que como injusticias y todos se afrontaron con la mentalidad de aprender todo lo posible, aunque no se consiguiera el objetivo deseado. Y fue gracias a esta mentalidad y a la paciencia por

obtener resultados que se consiguió completar toda esta guía y adquirir todos los conocimientos técnicos adicionales que no se reflejan en este proyecto.

8.1. TRABAJOS FUTUROS

El proyecto ha concluido con una guía que detalla paso a paso lo que hay que seguir para conseguir ejecutar en la tarjeta Avnet Ultra 96V2 un modelo YOLO entrenado con un dataset personalizado. Este proyecto es útil a fecha de hoy, pues los modelos YOLO se especializan en dar un resultado rápido por encima de la precisión, pero en el ámbito de la salud no es tanto la rapidez de un resultado sino la precisión de dicho resultado, y con la nueva normativa de la UE que se piensa implementar pronto, esto descartaría el uso de modelos como YOLO en ámbitos de la salud. Así pues, propondría el entrenamiento y cuantización de modelos basados en análisis cíclicos, aunque supongan una mayor carga computacional y un entrenamiento más complicado, pero dan resultados mucho más precisos. Además, considero que su integración en la DPU podría solventar su mayor problema, la eficiencia de los cálculos, que desemboca en una mayor carga computacional. Sin embargo, soy consciente de que las dimensiones habituales de estos modelos superan a las características de YOLO, como un modelo ligero y rápido para aplicaciones diversas, pero también soy conocedor de diversos proyectos que se están desarrollando con esta misma finalidad, de ser ligeros, que se espera que se publiquen pronto. Por tanto, me gustaría incentivar a desarrolladores a preparar una guía para modificar estos modelos nuevos para hacerlos cuantizables y preparar una guía de entrenamiento, cuantización e integración de éstos en un dispositivo MPSoC.

CAPÍTULO 9: BIBLIOGRAFÍA

AMD. (2022). *Vitis AI 3.0 Documentation*. Obtenido de Github.io: https://xilinx.github.io/Vitis-AI/3.0/html/index.html

AMD. (2023). *Vitis AI 3.5 Documentation*. Obtenido de Github.io: https://xilinx.github.io/Vitis-AI/3.5/html/index.html

Anna Pawłowska, A. Ć.-P. (2024). *Curated benchmark dataset for ultrasound based breast lesion analysis*. Obtenido de Nature- Scientific data: https://www.cancerimagingarchive.net/collection/breast-lesions-usg/

Bailén, J. M. (Julio de 2023). Plataforma de aceleración de inteligencia artificial en dispositivos MPSoC de Xilinx.

Bergeron, M. (21 de Marzo de 2023). *Hackster.io*. Obtenido de https://www.hackster.io/AlbertaBeef/ultra96-v2-building-the-foundational-designs-e4315f

Docker. (s.f.). *DockerDocs*. Obtenido de DockerDocs: https://docs.docker.com/engine/install/ubuntu/

El-Abed, N. A. (2022). *Microwave Imaging for Early Breast Cancer Detection: Current State, Challenges, and Future Directions*. Obtenido de MDPI: https://www.mdpi.com/2313-433X/8/5/123

Hartfiel, J. (2023). *Trenz Electronic Documentation*. Obtenido de Trenz Electronic Documentation: https://wiki.trenz-electronic.de/display/PD/PetaLinux

Hussain, M. (2024). *In-Depth Review of YOLOv1 to YOLOv10 Variants for Enhanced Photovoltaic Defect Detection*. Obtenido de MDPI: https://www.mdpi.com/2673-9941/4/3/16

Hussain, R. K. (Octubre de 2024). *YOLOv11: An Overview of the Key Architectural Enhancements.* Obtenido de Cornell University: https://arxiv.org/abs/2410.17725

LigicTronix. (15 de Diciembre de 2023). *Hackster.io*. Obtenido de Hackster.io: https://www.hackster.io/LogicTronix/yolov5-quantization-compilation-with-vitis-ai-3-0-for-kria-7b005d

Misoji. (Julio de 2024). *Vitis AI ONNX Runtime Engine (VOE) with KR260 + Python*. Obtenido de Hackster.io: https://www.hackster.io/iotengineer22/vitis-ai-onnx-runtime-engine-voe-with-kr260-python-0d02c3

Nazish Khalid, M. Z. (Junio de 2024). *Emerging paradigms in microwave imaging technology for biomedical applications: unleashing the power of artificial intelligence*. Obtenido de Nature- NPJ imaging: https://www.nature.com/articles/s44303-024-00012-8

Salcedo, P. M. (Diciembre de 2012). *SENSORES DE MICROONDAS PARA LA DETECCION* . Obtenido de Universidad de Cantabria en academia.edu: https://www.academia.edu/101877334/Sensores_de_microondas_para_la_detecci%C3%B3n_de_materiales_de_alta_constante_diel%C3%A9ctrica

Wang, L. (2023). *Microwave Imaging and Sensing Techniques for Breast Cancer Detection*. Obtenido de MDPI: https://www.mdpi.com/2072-666X/14/7/1462

Xilinx. (2023). *GitHub*. Obtenido de GitHub: https://github.com/Xilinx/Vitis-Tutorials/tree/2023.2

```
#Note: Mention Each package in individual line
#These packages will get added into rootfs menu entry

CONFIG_gpio-demo
CONFIG_peekpoke
CONFIG_xrt
CONFIG_dnf
CONFIG_gpio-utils
CONFIG_json-c
CONFIG_libpython3
CONFIG_lmsensors-sensorsdetect
CONFIG_coreutils
CONFIG_ethtool
CONFIG_gpio-utils
CONFIG_hdparm
CONFIG_i2c-tools
CONFIG_i2c-tools-misc
CONFIG_iperf3
CONFIG_iw
CONFIG_kernel-modules
CONFIG_nano
CONFIG_e2fsprogs-resize2fs
CONFIG_parted
CONFIG_resize-part
CONFIG_cmake
CONFIG_pciutils
CONFIG_python3
CONFIG_python3-pip
CONFIG_xrt-dev
CONFIG_opencl-clhpp-dev
CONFIG_opencl-headers-dev
CONFIG_mesa-megadriver
CONFIG_packagegroup-petalinux-gstreamer
CONFIG_packagegroup-petalinux-vitisai-dev
CONFIG_packagegroup-petalinux-opencv
CONFIG_packagegroup-petalinux-opencv-dev
CONFIG_packagegroup-petalinux-x11
CONFIG_packagegroup-petalinux-v4lutils
CONFIG_packagegroup-petalinux-matchbox
CONFIG_packagegroup-petalinux-vitisai
CONFIG_packagegroup-petalinux-self-hosted
CONFIG_vitis-ai-library
CONFIG_vitis-ai-library-dev
CONFIG_vitis-ai-library-dbg
CONFIG_wpa-supplicant
```

Anexo1. user-rootfsconfig.

```
/*
* Copyright 2019 Xilinx Inc.
```

```
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

//Setting the arch of DPU, For more details, Please read the PG338


/*====== Architecture Options ======*/
// |-----------------------------------------------------|
// | Support 8 DPU size
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | `define B512
// +-----------------------------------------------------+
// | `define B800
// +-----------------------------------------------------+
// | `define B1024
// +-----------------------------------------------------+
// | `define B1152
// +-----------------------------------------------------+
// | `define B1600
// +-----------------------------------------------------+
// | `define B2304
// +-----------------------------------------------------+
// | `define B3136
// +-----------------------------------------------------+
// | `define B4096
// |-----------------------------------------------------|

`define B1600

// |-----------------------------------------------------|
// | If the FPGA has Uram. You can define URAM_EN parameter
// | if change, Don't need update model
// +-----------------------------------------------------+
// | for zcu104 : `define URAM_ENABLE
// +-----------------------------------------------------+
// | for zcu102 : `define URAM_DISABLE
// |-----------------------------------------------------|

`define URAM_DISABLE
```

```
//config URAM
`ifdef URAM_ENABLE
  `define def_UBANK_IMG_N       5
  `define def_UBANK_WGT_N       17
  `define def_UBANK_BIAS        1
`elsif URAM_DISABLE
  `define def_UBANK_IMG_N       0
  `define def_UBANK_WGT_N       0
  `define def_UBANK_BIAS        0
`endif

// |-----------------------------------------------------|
// | You can use DRAM if FPGA has extra LUTs
// | if change, Don't need update model
// +-----------------------------------------------------+
// | Enable DRAM : `define DRAM_ENABLE
// +-----------------------------------------------------+
// | Disable DRAM : `define DRAM_DISABLE
// |-----------------------------------------------------|

`define DRAM_DISABLE

//config DRAM
`ifdef DRAM_ENABLE
  `define def_DBANK_IMG_N       1
  `define def_DBANK_WGT_N       1
  `define def_DBANK_BIAS        1
`elsif DRAM_DISABLE
  `define def_DBANK_IMG_N       0
  `define def_DBANK_WGT_N       0
  `define def_DBANK_BIAS        0
`endif

// |-----------------------------------------------------|
// | RAM Usage Configuration
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | RAM Usage High : `define RAM_USAGE_HIGH
// +-----------------------------------------------------+
// | RAM Usage Low : `define RAM_USAGE_LOW
// |-----------------------------------------------------|

`define RAM_USAGE_LOW

// |-----------------------------------------------------|
// | Channel Augmentation Configuration
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | Enable : `define CHANNEL_AUGMENTATION_ENABLE
// +-----------------------------------------------------+
// | Disable : `define CHANNEL_AUGMENTATION_DISABLE
// |-----------------------------------------------------|
```

```
`define CHANNEL_AUGMENTATION_ENABLE

// |-----------------------------------------------------|
// | ALU parallel Configuration
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | setting 0  : `define ALU_PARALLEL_DEFAULT
// +-----------------------------------------------------+
// | setting 1  : `define ALU_PARALLEL_1
// |-----------------------------------------------------|
// | setting 2  : `define ALU_PARALLEL_2
// |-----------------------------------------------------|
// | setting 3  : `define ALU_PARALLEL_4
// |-----------------------------------------------------|
// | setting 4  : `define ALU_PARALLEL_8
// |-----------------------------------------------------|

`define ALU_PARALLEL_DEFAULT

// +-----------------------------------------------------+
// | CONV RELU Type Configuration
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | `define CONV_RELU_RELU6
// +-----------------------------------------------------+
// | `define CONV_RELU_LEAKYRELU_RELU6
// |-----------------------------------------------------|

`define CONV_RELU_LEAKYRELU_RELU6

// +-----------------------------------------------------+
// | ALU RELU Type Configuration
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | `define ALU_RELU_RELU6
// +-----------------------------------------------------+
// | `define ALU_RELU_LEAKYRELU_RELU6
// |-----------------------------------------------------|

`define ALU_RELU_RELU6

// |-----------------------------------------------------|
// | argmax or max Configuration
// | It relates to model. if change, must update model
// +-----------------------------------------------------+
// | enable  : `define SAVE_ARGMAX_ENABLE
// +-----------------------------------------------------+
// | disable : `define SAVE_ARGMAX_DISABLE
// |-----------------------------------------------------|

`define SAVE_ARGMAX_ENABLE
```

```
// |-----------------------------------------------------|
// | DSP48 Usage Configuration
// | Use dsp replace of lut in conv operate
// | if change, Don't need update model
// +-----------------------------------------------------+
// | `define DSP48_USAGE_HIGH
// +-----------------------------------------------------+
// | `define DSP48_USAGE_LOW
// |-----------------------------------------------------|

`define DSP48_USAGE_LOW

// |-----------------------------------------------------|
// | Power Configuration
// | if change, Don't need update model
// +-----------------------------------------------------+
// | `define LOWPOWER_ENABLE
// +-----------------------------------------------------+
// | `define LOWPOWER_DISABLE
// |-----------------------------------------------------|

`define LOWPOWER_DISABLE

// |-----------------------------------------------------|
// | DEVICE Configuration
// | if change, Don't need update model
// +-----------------------------------------------------+
// | `define MPSOC
// +-----------------------------------------------------+
// | `define ZYNQ7000
// |-----------------------------------------------------|

`define MPSOC
```

Anexo 2. dpu_conf.vh

```
# /*
# * Copyright 2019 Xilinx Inc
# *
# * Licensed under the Apache License, Version 2,0 (the "License");
# * you may not use this file except in compliance with the License
# * You may obtain a copy of the License at
# *    http://www,apache,org/licenses/LICENSE-2,0
# * Unless required by applicable law or agreed to in writing, software
# * distributed under the License is distributed on an "AS IS" BASIS,
# * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
```

```
# * See the License for the specific language governing permissions and
# * limitations under the License
# */

[clock]

freqHz=200000000:DPUCZDX8G_1.aclk
freqHz=400000000:DPUCZDX8G_1.ap_clk_2

[connectivity]

sp=DPUCZDX8G_1.M_AXI_GP0:HPC0
sp=DPUCZDX8G_1.M_AXI_HP0:HP0
sp=DPUCZDX8G_1.M_AXI_HP2:HP1

[advanced]
misc=:solution_name=link

param=compiler.addOutputTypes=sd_card
#param=compiler.skipTimingCheckAndFrequencyScaling=1

[vivado]
prop=run.impl_1.strategy=Performance_Explore
#param=place.runPartPlacer=0
```

Anexo 3. dpu-link.cfg

```
# Add Docker's official GPG key:
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

Anexo 4. docker_apt_repo_installer.sh

```
#accept-nvidia-visible-devices-as-volume-mounts = false
#accept-nvidia-visible-devices-envvar-when-unprivileged = true
disable-require = false
supported-driver-capabilities = "compat32,compute,display,graphics,ngx,utility,video"
```

```
#swarm-resource = "DOCKER_RESOURCE_GPU"

[nvidia-container-cli]
#debug = "/var/log/nvidia-container-toolkit.log"
environment = []
#ldcache = "/etc/ld.so.cache"
ldconfig = "@/sbin/ldconfig.real"
load-kmods = true
#no-cgroups = false
#path = "/usr/bin/nvidia-container-cli"
#root = "/run/nvidia/driver"
#user = "root:video"

[nvidia-container-runtime]
#debug = "/var/log/nvidia-container-runtime.log"
log-level = "info"
mode = "auto"
runtimes = ["docker-runc", "runc", "crun"]

[nvidia-container-runtime.modes]

[nvidia-container-runtime.modes.cdi]
annotation-prefixes = ["cdi.k8s.io/"]
default-kind = "nvidia.com/gpu"
spec-dirs = ["/etc/cdi", "/var/run/cdi"]

[nvidia-container-runtime.modes.csv]
mount-spec-path = "/etc/nvidia-container-runtime/host-files-for-container.d"

[nvidia-container-runtime-hook]
path = "nvidia-container-runtime-hook"
skip-mode-detection = false

[nvidia-ctk]
path = "nvidia-ctk"
```

Anexo 5. config.tolm

```
train: datasets/coco128/images/images
val: datasets/coco128/images/images
test: datasets/coco128/images/images  # 128 images
label: datasets/coco128/labels
# number of classes
nc: 80

# class names
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic
light',
    'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
      'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase',
'frisbee',
      'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',
```

```
'surfboard',
      'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
      'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
     'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell
phone',
      'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy
bear',
      'hair drier', 'toothbrush']
```

Anexo 6. coco128.yalm

```
# pip install -r requirements.txt

# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5,<=1.21.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch==1.12.1
# torch==1.8.0
torchvision==0.13.1
# torchvision==0.9.0
tqdm>=4.41.0

# Logging -------------------------------------
tensorboard>=2.4.1
 wandb

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Export --------------------------------------
 coremltools>=4.1  # CoreML export
 onnx>=1.9.0  # ONNX export
 onnx-simplifier>=0.3.6  # ONNX simplifier
 scikit-learn==0.19.2  # CoreML quantization
# tensorflow>=2.4.1  # TFLite export
# tensorflowjs>=3.9.0  # TF.js export

# Extras --------------------------------------
 albumentations>=1.0.3
 Cython  # for pycocotools https://github.com/cocodataset/cocoapi/issues/172
 pycocotools>=2.0  # COCO mAP
 roboflow
thop  # FLOPs computation
 ninja
```

Anexo 7. requeriments.txt

```
# Copyright 2019 Xilinx Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

"""Statistics utilities of model complexity."""

import torch
from torch import nn

from nndct_shared.utils import common

import logging

class MetricName(object):
 MACs = 'MACs'
 Flops = 'Flops'
 TrainableParams = 'trainable'
 NonTrainableParams = 'non-trainable'

def _accumulate_metric_value(module, metric_name, value):
 setattr(module, metric_name, getattr(module, metric_name) + value)

def count_convNd(module, input, output):
 # kw x kh
 num_kernels = (module.weight.size()[2:]).numel()
 bias = 1 if module.bias is not None else 0

 # c_out x w x h  x (c_in x kw x kh + bias)
 single_batch_output = output[0]
  MACs = single_batch_output.numel() * (module.in_channels // module.groups *
num_kernels)
 Flops = 2 * MACs + bias * single_batch_output.numel()
 _accumulate_metric_value(module, MetricName.MACs, MACs)
 _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_linear(module, input, output):
 # (N, *, Hin) x (Hin, Hout) = (N, *, Hout)
 bias = 1 if module.bias is not None else 0

 MACs = module.in_features * output[0].numel()
 Flops = 2 * MACs + bias * output.shape[-1]
```

```
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def zero_ops(module, input, output):
  MACs = 0
  Flops = 2 * MACs
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_normalization(module, input, output):
  # y = (x - mean) / sqrt(eps + var) * weight + bias
  MACs = 2*input[0].numel()
  Flops = 2 * MACs
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_relu(module, input, output):
  MACs = 0
  Flops = input[0].numel()
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_prelu(module, input, output):
  MACs = input[0].numel()
  Flops = 2 * MACs
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_softmax(module, input, output):
  nfeatures = input[0].size()[module.dim]
  total_exp = nfeatures
  total_add = nfeatures - 1
  total_div = nfeatures
  total_ops = total_exp + total_div
  MACs = total_ops
  Flops = MACs + total_add
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_sigmoid(module, input, output):
  MACs = 0
  Flops = 4 * input[0].numel()
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_pool(module, input, output):
  MACs = 0
  Flops = output.numel()
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_adap_pool(module, input, output):
```

```
  kernel = torch.div(
    torch.DoubleTensor([*(input[0].shape[2:])]),
    torch.DoubleTensor([*(output.shape[2:])])
  )
  total_add = torch.prod(kernel)
  num_elements = output.numel()
  total_div = 1
  kernel_op = total_add + total_div
  total_ops = int(kernel_op * num_elements)
  MACs = 0
  Flops = total_ops
  _accumulate_metric_value(module, MetricName.MACs, MACs)
  _accumulate_metric_value(module, MetricName.Flops, Flops)

def count_upsample(module, input, output):
  if module.mode not in (
     "nearest",
     "linear",
     "bilinear",
     "bicubic",
  ):  # "trilinear"
    logging.warning("mode %s is not implemented yet, take it a zero op" % m.mode)
    MACs = 0
    Flops = 0
    _accumulate_metric_value(module, MetricName.MACs, MACs)
    _accumulate_metric_value(module, MetricName.Flops, Flops)
  else:
    total_ops = output.nelement()
    if module.mode == "linear":
      total_ops *= 5
    elif module.mode == "bilinear":
      total_ops *= 11
    elif module.mode == "bicubic":
      ops_solve_A = 224  # 128 muls + 96 adds
      ops_solve_p = 35  # 16 muls + 12 adds + 4 muls + 3 adds
      total_ops *= ops_solve_A + ops_solve_p
    elif module.mode == "trilinear":
      total_ops *= 13 * 2 + 5
    MACs = 0
    Flops = total_ops
    _accumulate_metric_value(module, MetricName.MACs, MACs)
    _accumulate_metric_value(module, MetricName.Flops, Flops)

macs_counters = {
    nn.ZeroPad2d: zero_ops,  # padding does not involve any multiplication.
    nn.Conv1d: count_convNd,
    nn.Conv2d: count_convNd,
    nn.Conv3d: count_convNd,
    nn.ConvTranspose1d: count_convNd,
    nn.ConvTranspose2d: count_convNd,
    nn.ConvTranspose3d: count_convNd,
    nn.BatchNorm1d: count_normalization,
```

```
    nn.BatchNorm2d: count_normalization,
    nn.BatchNorm3d: count_normalization,
    nn.LayerNorm: count_normalization,
    nn.InstanceNorm1d: count_normalization,
    nn.InstanceNorm2d: count_normalization,
    nn.InstanceNorm3d: count_normalization,
    nn.PReLU: count_prelu,
    nn.Softmax: count_softmax,
    nn.Sigmoid: count_sigmoid,
    nn.ReLU: count_relu,
    nn.ReLU6: count_relu,
    nn.LeakyReLU: count_relu,
    nn.MaxPool1d: count_pool,
    nn.MaxPool2d: count_pool,
    nn.MaxPool3d: count_pool,
    nn.AdaptiveMaxPool1d: count_adap_pool,
    nn.AdaptiveMaxPool2d: count_adap_pool,
    nn.AdaptiveMaxPool3d: count_adap_pool,
    nn.AvgPool1d: count_pool,
    nn.AvgPool2d: count_pool,
    nn.AvgPool3d: count_pool,
    nn.AdaptiveAvgPool1d: count_adap_pool,
    nn.AdaptiveAvgPool2d: count_adap_pool,
    nn.AdaptiveAvgPool3d: count_adap_pool,
    nn.Linear: count_linear,
    nn.Dropout: zero_ops,
    nn.Upsample: count_upsample,
    nn.UpsamplingBilinear2d: count_upsample,
    nn.UpsamplingNearest2d: count_upsample,
    nn.Sequential: zero_ops,
    nn.PixelShuffle: zero_ops,
}

class MetricHook(object):

  def __init__(self):
    self._module_hooks = []

  def register(self, module):
    raise NotImplementedError('Not implemented')

  def clear(self, module):
    pass

  @property
  def module_hooks(self):
    return self._module_hooks

  @property
  def value(self):
    raise NotImplementedError('Not implemented')
```

```
class macsMetric(MetricHook):

 def __init__(self):
  super(macsMetric, self).__init__()

 def register(self, module):
  handle = None
  fn = macs_counters.get(type(module), None)
  if fn:
   module.register_buffer(MetricName.MACs, torch.zeros(1, dtype=torch.int64))
   module.register_buffer(MetricName.Flops, torch.zeros(1, dtype=torch.int64))
   handle = module.register_forward_hook(fn)
   self._module_hooks.append((module, handle))
  return handle

 def clear(self, module):
  if type(module) in macs_counters:
   module._buffers.pop(MetricName.MACs)
   module._buffers.pop(MetricName.Flops)

 def value(self, module):
  if type(module) in macs_counters:
                                return          [module._buffers.get(MetricName.MACs).item(),
module._buffers.get(MetricName.Flops).ite>
  return None

class HookedStat(object):

 def __init__(self):
  self._metrics = []
  self._module_hooks = {}

 def _register_metric(self, module):
  for metric in self._metrics:
   hook = metric.register(module)
   if not hook:
    continue
   if module not in self._module_hooks:
    self._module_hooks[module] = []
   self._module_hooks[module].append(hook)

 def clear_metrics(self):
  for module, hooks in self._module_hooks.items():
   for hook in hooks:
    hook.remove()

   for metric in self._metrics:
    metric.clear(module)
  self._metrics = []

 def _aggregate_metric_values(self):
  values = {}
```

```
  for idx, module in enumerate(self._module_hooks.keys()):
    class_name = str(module.__class__).split(".")[-1].split("'")[0]
    module_key = "%s-%i" % (class_name, idx + 1)
    metric_values = {
       'MACs and Flops': metric.value(module) for metric in self._metrics
    }
    values[module_key] = metric_values

  return values

 def add_metric(self, metric):
  self._metrics.append(metric)

 def run(self, model, inputs):
  model.apply(self._register_metric)

  if torch.cuda.is_available():
    model.cuda()
    if isinstance(inputs, (tuple, list)):
      inputs = [input.cuda() for input in inputs]
    else:
      inputs = [inputs.cuda()]
  model.eval()
  model(*inputs)

  values = self._aggregate_metric_values()

  self.clear_metrics()
  return values

def stat_macs(model, inputs):
 stat = HookedStat()
 stat.add_metric(macsMetric())
 metrics = stat.run(model, inputs)

 model_analysis_info = {}

 for module, metric in metrics.items():
  class_name = module.split("-")[0]
  if class_name not in model_analysis_info:
    model_analysis_info[class_name] = {'number':1 , 'MACs':metric['MACs and Flops'][0],
'Flops':me>
  else:
    model_analysis_info[class_name]['number'] += 1
    model_analysis_info[class_name]['MACs'] += metric['MACs and Flops'][0]
    model_analysis_info[class_name]['Flops'] += metric['MACs and Flops'][1]

 sorted(model_analysis_info.items(), key = lambda x:x[0][2] ,reverse = True)
 return metrics, model_analysis_info

def stat_params(model):
```

```
  params = {}

  for name, module in model.named_modules():
    if len(list(module.children())) > 0:
      continue

    params[name] = {}
    for p in module.parameters():
      key = (
        MetricName.TrainableParams
        if p.requires_grad else MetricName.NonTrainableParams)
      if key not in params[name]:
       params[name][key] = 0
      params[name][key] += p.numel()

  return params

def model_complexity(model, inputs, return_flops=False, readable=False,
print_model_analysis=False):
  """Stat the complexity of the given model. Currently includes macs and params.
  MACs: multiply–accumulate operations that performs a += b x c
  Flops = 2*MACs + BiasAdd
  Params: total number of parameters of a model.

  Args:
    model: An `nn.Module` object.
    inputs: A list or tuple of inputs used to run forward passes on the model.
    readable: Whether to return readable numbers.

  Returns:
    Statistically obtained macs and params by given inputs.
  """
  macs, model_analysis_info = stat_macs(model, inputs)
  total_macs = 0
  total_flops = 0

  for module, value in macs.items():
    total_macs += value['MACs and Flops'][0]
    total_flops += value['MACs and Flops'][1]

  if print_model_analysis:
    header_fields = ['Module Name', 'Number', 'MACs', 'Flops']
    rows_fields = []
    for module, value in model_analysis_info.items():
      rows_fields.append([module, value['number'], value['MACs'], value['Flops']])
    #spu.print_summary(header_fields, rows_fields)
    logging.info("Total multiply–accumulate operations (MACs) : {0}".format(total_macs))
    logging.info("Total floating point operations (Flops) : {0}".format(total_flops))

  if return_flops:
    total_macs = total_flops
```

```
  total_params = 0
  params = stat_params(model)
  for name in params:
    if MetricName.TrainableParams in params[name]:
      total_params += params[name][MetricName.TrainableParams]

  if readable:
    total_macs = common.readable_num(total_macs)
    total_params = common.readable_num(total_params)
  return total_macs, total_params
```

Anexo 8. summary.py

```
#DEMO Use for the script
#python ONNX_quantizer.py ONNX_models/yolo.onnx ONNX_models/QUANT_PTQ/yolo_quantized.onnx datasets/coco128/images/ --calib_method *** --quant_format FixNeuron



import os
import cv2
import numpy as np
from pathlib import Path
import onnxruntime as ort
from onnxruntime.quantization import CalibrationDataReader, QuantType, QuantFormat, quantize_static
import vai_q_onnx

class YOLOv5CalibrationDataReader(CalibrationDataReader):
    def __init__(self, images_folder, input_name, input_size=(640, 640), batch_size=1):
        """
        Calibration data reader for YOLOv5 model quantization.

        Args:
            images_folder (str): Path to the folder containing the images.
            input_size (tuple): Input size for the model (default: 640x640).
            batch_size (int): Number of images per batch.
        """
        self.input_size = input_size
        self.batch_size = batch_size
        self.input_name = input_name

        # Get all image files
        self.image_files = sorted(list(Path(images_folder).glob("*.jpg")) + list(Path(images_folder).glob("*.png")))
        self.iterator = iter(self._generate_batches())

    def _generate_batches(self):
        """
```

```
    Generate batches of images preprocessed for calibration.
    """
    for i in range(0, len(self.image_files), self.batch_size):
      batch_images = []
      for image_file in self.image_files[i:i + self.batch_size]:
        # Load and preprocess the image
        # Viejo
        """
        img = cv2.imread(str(image_file))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, self.input_size)
        img = img / 255.0  # Normalize
        img = np.transpose(img, (2, 0, 1))  # HWC to CHW
        """
        # Nuevo con letterbox
        img = Image.open(image_file).convert("RGB")
        img = letterbox_image(img)
        img = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
       img = cv2.dnn.blobFromImage(img, 1/255, self.input_size, [0, 0, 0], 1, crop=False)
        """
        data = img.squeeze(0).transpose(1,2,0)
        data = (data * 255).astype(np.uint8)
        data = Image.fromarray(data)
        data.save("letterboxed_img.png")

        """
        #import ipdb;ipdb.set_trace()
        batch_images.append(img[0])

      yield {self.input_name: np.array(batch_images, dtype=np.float32)}

  def get_next(self):
    """
    Get the next batch of data for calibration.
    """
    try:
      return next(self.iterator)
    except StopIteration:
      return None

##############################################################################
############################
import torch
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from torchvision.datasets import CIFAR10
class CIFAR10DataSet:
  def __init__(
    self,
    data_dir,
    **kwargs,
  ):
```

```
    super().__init__()
    self.train_path = data_dir
    self.vld_path = data_dir
    self.setup("fit")

  def setup(self, stage: str):
    transform = transforms.Compose(
                                [transforms.Pad(4),      transforms.RandomHorizontalFlip(),
transforms.RandomCrop(32), transforms.ToTensor()]
    )
     self.train_dataset = CIFAR10(root=self.train_path, train=True, transform=transform,
download=False)
      self.val_dataset = CIFAR10(root=self.vld_path, train=True, transform=transform,
download=False)

class PytorchResNetDataset(Dataset):
  def __init__(self, dataset):
    self.dataset = dataset

  def __len__(self):
    return len(self.dataset)

  def __getitem__(self, index):
    sample = self.dataset[index]
    input_data = sample[0]
    label = sample[1]
    return input_data, label

def create_dataloader(data_dir, batch_size):
  cifar10_dataset = CIFAR10DataSet(data_dir)
  _, val_set = torch.utils.data.random_split(cifar10_dataset.val_dataset, [49000, 1000])
        benchmark_dataloader      =      DataLoader(PytorchResNetDataset(val_set),
batch_size=batch_size, drop_last=True)
  return benchmark_dataloader

class ResnetCalibrationDataReader(CalibrationDataReader):
  def __init__(self, data_dir: str, batch_size: int = 16):
    super().__init__()
    self.iterator = iter(create_dataloader(data_dir, batch_size))

  def get_next(self) -> dict:
    try:
      images, labels = next(self.iterator)
      return {"images": images.numpy()}
    except Exception:
      return None

def resnet_calibration_reader(data_dir, batch_size=16):
  return ResnetCalibrationDataReader(data_dir, batch_size=batch_size)
##########################################################################
########################
```

```
def quantize_yolov5(input_model_path, output_model_path, images_folder, quant_format, calib_method):
  """
  Quantize the YOLOv5 ONNX model.

  Args:
    input_model_path (str): Path to the original ONNX model.
    output_model_path (str): Path where the quantized model will be saved.
    images_folder (str): Path to the folder containing the calibration images.
  """

  ort_session = ort.InferenceSession(input_model_path)
  nombre_entrada_real = ort_session.get_inputs()[0].name
  nombre_salida_real = ort_session.get_outputs()[0].name
  #import ipdb;ipdb.set_trace()
  # Create the calibration data reader
                                                calibration_reader                =
YOLOv5CalibrationDataReader(images_folder,nombre_entrada_real)
   ##########################################################################
###############
  # `calibration_dataset_path` is the path to the dataset used for calibration during quantization.
  calibration_dataset_path = "datasets/CIFAR10/"

  # `dr` (Data Reader) is an instance of ResNet50DataReader, which is a utility class that
  # reads the calibration dataset and prepares it for the quantization process.
  dr = resnet_calibration_reader(calibration_dataset_path)
   ##########################################################################
###############

  # Perform quantization
  """
  vai_q_onnx.quantize_static(
    model_input=input_model_path,
    model_output=output_model_path,
    calibration_data_reader=calibration_reader,
    quant_format=vai_q_onnx.VitisQuantFormat.FixNeuron,
    calibrate_method=vai_q_onnx.PowerOfTwoMethod.MinMSE,
    activation_type=QuantType.QUInt8,
    weight_type=QuantType.QInt8,
  )
  """
  vai_q_onnx.quantize_static(
    model_input=input_model_path,
    model_output=output_model_path,
    calibration_data_reader=dr,
    quant_format=quant_format,
    calibrate_method=calib_method,
    #input_nodes=[nombre_entrada_real],
    #output_nodes=[nombre_salida_real],
    op_types_to_quantize=None,
    per_channel=True,
```

```
        reduce_range=False,
        activation_type=QuantType.QInt8,
        weight_type=QuantType.QInt8,
        nodes_to_quantize=None,
        nodes_to_exclude=None,
        optimize_model=True,
        use_external_data_format=False,
        #extra_options=None,
    )

    print(f"Quantized model saved at: {output_model_path}")

from PIL import Image, ImageOps
def letterbox_image(image, target_size=(640, 640)):
    original_width, original_height = image.size
    target_width, target_height = target_size

    # Calcular la escala para que la imagen quepa en la caja de destino
    scale = min(target_width / original_width, target_height / original_height)
    new_width = int(original_width * scale)
    new_height = int(original_height * scale)

    # Redimensionar la imagen manteniendo la relación de aspecto
    resized_image = image.resize((new_width, new_height), Image.BILINEAR)

    # Calcular el relleno (asegurando que el relleno sea simétrico)
    pad_width = (target_width - new_width) // 2
    pad_height = (target_height - new_height) // 2

    # Si hay un pixel impar de diferencia, ajustar el padding
    pad_width_extra = (target_width - new_width) % 2
    pad_height_extra = (target_height - new_height) % 2

    # Agregar los márgenes con el relleno gris (128, 128, 128)
    padded_image = ImageOps.expand(resized_image,
                                   border=(pad_width, pad_height, pad_width + pad_width_extra,
pad_height + pad_height_extra),
                                   fill=(128, 128, 128))

    return padded_image

if __name__ == "__main__":
    import argparse

    # Map string arguments to Vitis AI quantization formats and calibration methods
    quant_format_map = {
        "FixNeuron": vai_q_onnx.VitisQuantFormat.FixNeuron,
        "QO": QuantFormat.QOperator,
        "QDQ": QuantFormat.QDQ,
        "VitisQDQ": vai_q_onnx.VitisQuantFormat.QDQ,
    }
```

```
  calib_method_map = {
    "NonOverflow": vai_q_onnx.PowerOfTwoMethod.NonOverflow,
    "MinMSE": vai_q_onnx.PowerOfTwoMethod.MinMSE,
  }

  parser = argparse.ArgumentParser(description="Quantize YOLOv5 ONNX model using Vitis AI.")
    parser.add_argument("input_model", type=str, help="Path to the original YOLOv5 ONNX model.")
    parser.add_argument("output_model", type=str, help="Path to save the quantized YOLOv5 model.")
   parser.add_argument("images_folder", type=str, help="Path to the folder containing calibration images.")
  parser.add_argument(
    "--quant_format",
    type=str,
    choices=quant_format_map.keys(),
    default="QDQ",
    help="Quantization format: FixNeuron, QO, QDQ, or VitisQDQ (default: QDQ).",
  )
  parser.add_argument(
    "--calib_method",
    type=str,
    choices=calib_method_map.keys(),
    default="NonOverflow",
    help="Calibration method: NonOverflow or MinMSE (default: NonOverflow).",
  )
  args = parser.parse_args()

  # Resolve the quant_format and calib_method to their actual objects
  quant_format = quant_format_map[args.quant_format]
  calib_method = calib_method_map[args.calib_method]

        quantize_yolov5(args.input_model, args.output_model, args.images_folder, quant_format, calib_method)
```

Anexo 9. ONNX_quantizer.py

```
#include <glog/logging.h>
#include <algorithm>
#include <cmath>
#include <functional>
#include <iomanip>
#include <iostream>
#include <memory>
#include <numeric>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xir/graph/graph.hpp>
```

```
#include "vart/runner.hpp"
#include "vart/runner_ext.hpp"
#include "vitis/ai/collection_helper.hpp"

//RAROS
#include <vector>
#include <tuple>
#include <vart/tensor_buffer.hpp>
#include <xir/tensor/tensor.hpp>
#include "common.h"
//#include <opencv2/dnn.hpp>

struct TensorData {
  std::vector<std::vector<std::vector<std::vector<float>>>> data;
  std::vector<int> shape;
};

// Funciones de preprocesamiento
static cv::Mat preprocess_image(const cv::Mat& image, cv::Size size);
static void setImageRGB(const cv::Mat& image, void* data, float fix_scale);

// Funciones de postprocesamiento

static void process_detections(const std::vector<std::tuple<TensorData, TensorData>>& tensor_pairs, float confidence_threshold, cv::Mat& image, int nc);
void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, float confidence_threshold, float iou_threshold, std::vector<int>& indices);
float IoU(const cv::Rect& box1, const cv::Rect& box2);
void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& class_ids, float confidence_threshold, float iou_threshold, std::vector<int>& indices, int nc);

// Funciones adicionales
// Imprimir la imagen a partir del puntero
static void print_input_image(const int8_t* data);
// Ver los vectores
static void print_tensor(vart::TensorBuffer* tensor_buffer);
// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor);
static float get_output_scale(const xir::Tensor* tensor);
// Comprobar si se pueden generar imagenes
bool canDisplayImages();
void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences,  const std::vector<int>& indices, const std::vector<int>& class_ids);
//Sigmoid
float sigmoid(float x);

//PREPROCESAMIENTO

// Preprocesamiento de imágenes

static cv::Mat preprocess_image(const cv::Mat& image, cv::Size size) {
```

```
 // Crear un fondo gris de tamaño `size` (640x640)
 cv::Mat output_image(size, CV_8UC3, cv::Scalar(128, 128, 128));

 // Calcular la escala manteniendo la proporción
    double scale = std::min(static_cast<double>(size.width) / image.cols,
static_cast<double>(size.height) / image.rows);

 // Calcular el nuevo tamaño de la imagen
  cv::Size new_size(static_cast<int>(image.cols * scale), static_cast<int>(image.rows *
scale));

 // Redimensionar la imagen manteniendo la proporción
 cv::Mat resized_image;
 cv::resize(image, resized_image, new_size);

  // Calcular las coordenadas para centrar la imagen redimensionada dentro del fondo
gris
 int top_left_x = (size.width - new_size.width) / 2;
 int top_left_y = (size.height - new_size.height) / 2;

 // Pegar la imagen redimensionada dentro del fondo gris
   resized_image.copyTo(output_image(cv::Rect(top_left_x, top_left_y, new_size.width,
new_size.height)));;

 return output_image;
}

static void setImageRGB(const cv::Mat& image, void* data, float fix_scale) {
   int8_t* data1 = (int8_t*)data;
   int c = 0;
   for (int row = 0; row < image.rows; row++) {
      for (int col = 0; col < image.cols; col++) {
         auto v = image.at<cv::Vec3b>(row, col);
         auto B = (float)v[0];
         auto G = (float)v[1];
         auto R = (float)v[2];

         // Normalización
         auto nB = B / 255.0f * fix_scale;
         auto nG = G / 255.0f * fix_scale;
         auto nR = R / 255.0f * fix_scale;

         // Clamping
         nB = std::max(std::min(nB, fix_scale), 0.0f);
         nG = std::max(std::min(nG, fix_scale), 0.0f);
         nR = std::max(std::min(nR, fix_scale), 0.0f);

         data1[c++] = (int8_t)(nR);
         data1[c++] = (int8_t)(nG);
         data1[c++] = (int8_t)(nB);
      }
   }
```

```
  //print_input_image(data1);

}

//POSTPROCESAMIENTO

// Filtrar detecciones basadas en el umbral de confianza y aplicar NMS
static void process_detections(const std::vector<std::tuple<TensorData, TensorData>>& tensor_pairs, float confidence_threshold, cv::Mat& image, int nc) {
  std::vector<cv::Rect> boxes;
  std::vector<int> class_ids;
  std::vector<float> confidences;
  std::vector<std::string> class_names = {"PERSON", "BICYCLE", "CAR", "MOTORCYCLE", "AIRPLANE", "BUS", "TRAIN", "TRUCK", "BOAT", "TRAFFIC LIGHT", "FIRE HYDRANT", "STOP SIGN", "PARKING METER", "BENCH", "BIRD", "CAT", "DOG", "HORSE", "SHEEP", "COW", "ELEPHANT", "BEAR", "ZEBRA", "GIRAFFE", "BACKPACK", "UMBRELLA", "HANDBAG", "TIE", "SUITCASE", "FRISBEE", "SKIS", "SNOWBOARD", "SPORTS BALL", "KITE", "BASEBALL BAT", "BASEBALL GLOVE", "SKATEBOARD", "SURFBOARD", "TENNIS RACKET", "BOTTLE", "WINE GLASS", "CUP", "FORK", "KNIFE", "SPOON", "BOWL", "BANANA", "APPLE", "SANDWICH", "ORANGE", "BROCCOLI", "CARROT", "HOT DOG", "PIZZA", "DONUT", "CAKE", "CHAIR", "COUCH", "POTTED PLANT", "BED", "DINING TABLE", "TOILET", "TV", "LAPTOP", "MOUSE", "REMOTE", "KEYBOARD", "CELL PHONE", "MICROWAVE", "OVEN", "TOASTER", "SINK", "REFRIGERATOR", "BOOK", "CLOCK", "VASE", "SCISSORS", "TEDDY BEAR", "HAIR DRIER", "TOOTHBRUSH"};
  int counter=0;
  std::vector<int> biases = {10, 13, 36, 75, 142, 110};

  for (const auto& pair : tensor_pairs) {
    const TensorData& class_probs = std::get<0>(pair);
    const TensorData& boxes_tensor = std::get<1>(pair);

    // Determinar el stride según las dimensiones del tensor de cajas
     int stride = (boxes_tensor.shape[1] == 80) ? 8 : (boxes_tensor.shape[1] == 40) ? 16 : (boxes_tensor.shape[1] == 20) ? 32 : 1;
    int anch = (boxes_tensor.shape[1] == 80) ? 0 : (boxes_tensor.shape[1] == 40) ? 1 : (boxes_tensor.shape[1] == 20) ? 2 : -1;

    for (int i = 0; i < class_probs.shape[0]; ++i) {
      for (int j = 0; j < class_probs.shape[1]; ++j) {
        for (int k = 0; k < class_probs.shape[2]; ++k) {
          for (int c = 0; c < nc; ++c) {
            float confidence = class_probs.data[i][j][k][c];
            if (confidence > confidence_threshold) {
              if (confidence == 1) counter++;

              // Extraer las coordenadas de la caja que están en un formato diferente del habitual:
              //y_center, x_center, width y height

              float y = (sigmoid(boxes_tensor.data[i][j][k][0]) + j) * stride;
              float x = (sigmoid(boxes_tensor.data[i][j][k][1]) + k) * stride;
```

```
                //float w = (boxes_tensor.data[i][j][k][2]) * stride * std::exp(1);
                //float h = (boxes_tensor.data[i][j][k][3]) * stride * std::exp(1);
                //Test
                //float w = std::exp(sigmoid(boxes_tensor.data[i][j][k][2]))*biases[anch];
                      float w = std::pow(sigmoid(boxes_tensor.data[i][j][k][2])*(std::exp(1)-
anch/4),2)*stride;
                //float h = std::exp(sigmoid(boxes_tensor.data[i][j][k][3]))*biases[anch+1];
                       float h = std::pow(sigmoid(boxes_tensor.data[i][j][k][3])*(std::exp(1)-
anch/4),2)*stride;
                //



                float left = (x - w / 2);
                float top = (y - h / 2);


                // Comprobar si las dimensiones son válidas (ancho y alto mayores que 1)
                if (w >= 1 || h >= 1) {

                  boxes.emplace_back(cv::Rect(left, top, w, h));
                  float max_class_score = class_probs.data[i][j][k][c];
                  int class_id = c;

                  // Encontrar la clase con el mayor puntaje
                  for (int jj = 0; jj < nc; ++jj) {
                    if (class_probs.data[i][j][k][jj] > max_class_score) {
                      max_class_score = class_probs.data[i][j][k][jj];
                      class_id = jj;  // Ajustar índice según el formato de salida de YOLO
                    }
                  }

                  class_ids.push_back(class_id);
                  confidences.push_back(confidence);
                }
              }
            }
          }
        }
      }

  //filtro de cajas
  for(auto& box : boxes){

    float top = box.y;
    float left = box.x;
    float bottom = box.y + box.height;
    float right = box.x + box.width;

    if (top < 0) top = 0;
```

```
    if (left < 0) left = 0;
    if (bottom > 640) bottom = 640;
    if (right > 640) right = 640;

    box.y = top;
    box.x = left;
    box.height = bottom - top;
    box.width = right - left;

  }

  // Aplicar NMS (supresión de no máximos) para eliminar detecciones redundantes
  std::vector<int> indices;
  //for (int a=0; a<confidences.size(); ++a) indices.push_back(a);
     NMSBoxesPerClass(boxes, confidences, class_ids, confidence_threshold, 0.2,
indices, nc);
  //NMSBoxesCustom(boxes, confidences, confidence_threshold, 0.4, indices);


  // Dibujar las cajas de detección en la imagen
  // se puede poner dentro del condicional esto también, pero igual se queda pillado
  //getenv("DISPLAY") != nullptr
  std::vector<cv::Scalar> colors;
  for (int i = 0; i < nc; ++i) {
    int r = (i * 37) % 255;  // Cálculo cíclico para rojo
    int g = (i * 73) % 255;  // Cálculo cíclico para verde
    int b = (i * 7) % 255;  // Cálculo cíclico para azul
    colors.emplace_back(cv::Scalar(b, g, r));
  }


  if (canDisplayImages()) {
    std::cout << "Ha detectado que puede imprimir imágenes" << std::endl;
    for (int idx : indices) {
      cv::rectangle(image, boxes[idx], colors[class_ids[idx]] , 2);
                    std::string label = class_names[class_ids[idx]]+ " " +
cv::format("%.1f",confidences[idx]);
      int baseline = 0;
        cv::Size textSize = cv::getTextSize(label, cv::FONT_HERSHEY_SIMPLEX, 0.5, 1,
&baseline);
      int rect_top = std::max(boxes[idx].y - textSize.height - 5, 0);
      int rect_left = boxes[idx].x;
      int rect_width = textSize.width + 10;
      int rect_height = textSize.height + 5;
       cv::rectangle(image, cv::Point(rect_left, rect_top), cv::Point(rect_left + rect_width,
rect_top + rect_height), colors[class_ids[idx]], cv::FILLED);
         cv::putText(image, label, cv::Point(rect_left + 5, rect_top + textSize.height),
cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255, 255, 255), 1);
    }
    printDetections(boxes, confidences, indices, class_ids);
    cv::imshow("Detecciones", image);
    cv::waitKey(0);
```

```
    } else {
        std::cout << "Ha detectado que no puede imprimir imágenes" << std::endl;
        printDetections(boxes, confidences, indices, class_ids);
    }

}

float IoU(const cv::Rect& box1, const cv::Rect& box2) {
    // Suponiendo que Box tiene las propiedades x1, y1, x2, y2
    float x1_intersection = std::max(box1.x, box2.x);
    float y1_intersection = std::max(box1.y, box2.y);
    float x2_intersection = std::min(box1.x + box1.width, box2.x + box2.width);
    float y2_intersection = std::min(box1.y + box1.height, box2.y + box2.height);

    // Calcular el área de intersección
    float intersection_area = std::max(0.0f, x2_intersection - x1_intersection) *
                              std::max(0.0f, y2_intersection - y1_intersection);

    // Calcular el área de las cajas
    float area1 = box1.width * box1.height;
    float area2 = box2.width * box2.height;

    // Calcular el área de unión
    float union_area = area1 + area2 - intersection_area;

    // Evitar división por cero
    if (union_area <= 0) return 0.0f;

    // Retornar IoU
    return intersection_area / union_area;
}

void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences, const std::vector<int>& class_ids, float confidence_threshold, float
iou_threshold, std::vector<int>& indices, int nc) {

    // Iterar por cada clase
    for (int c = 0; c < nc; ++c) {
        std::vector<int> class_indices;

        // Recoger índices de la clase actual
        for (size_t i = 0; i < class_ids.size(); ++i) {
            if (class_ids[i] == c && confidences[i] > confidence_threshold) {
                class_indices.push_back(i);
            }
        }

        // Ordenar por confianza
        std::sort(class_indices.begin(), class_indices.end(),
                  [&](int i, int j) { return confidences[i] > confidences[j]; });

        while (!class_indices.empty()) {
```

```
        int best_index = class_indices[0];
        indices.push_back(best_index);
        class_indices.erase(class_indices.begin());

        for (auto it = class_indices.begin(); it != class_indices.end();) {
          int idx = *it;
          float iou = IoU(boxes[best_index], boxes[idx]);

          if (iou > iou_threshold) {
            it = class_indices.erase(it);
          } else {
            ++it;
          }
        }
      }
    }
}

void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences,
              float confidence_threshold, float iou_threshold, std::vector<int>& indices) {

  // Filtrar las cajas basadas en el umbral de confianza
  std::vector<int> initial_indices;
  for (size_t i = 0; i < confidences.size(); ++i) {
    if (confidences[i] >= confidence_threshold) {
      initial_indices.push_back(i);
    }
  }

  // Ordenar las cajas filtradas por puntuación de confianza descendente
  std::sort(initial_indices.begin(), initial_indices.end(),
        [&](int i, int j) { return confidences[i] > confidences[j]; });

  // NMS
  while (!initial_indices.empty()) {
    int current_idx = initial_indices.front();
    indices.push_back(current_idx);
    initial_indices.erase(initial_indices.begin());

    auto it = initial_indices.begin();
    while (it != initial_indices.end()) {
      int idx = *it;
      float iou = IoU(boxes[current_idx], boxes[idx]);
      if (iou > iou_threshold) {
        it = initial_indices.erase(it);
      } else {
        ++it;
      }
    }
  }
}
```

```
//FUNCIONES ADICIONALES

// Imprimir la imagen de entrada a la DPU
static void print_input_image(const int8_t* data) {
  // Dimensiones de la imagen
  const int height = 640;
  const int width = 640;
  const int channels = 3;  // RGB

  // Crear un objeto cv::Mat para almacenar la imagen resultante (8-bit, 3 canales)
  cv::Mat image(height, width, CV_8UC3);

  // Recorremos el buffer de datos y convertimos a valores de imagen
  int index = 0;
  for (int row = 0; row < height; ++row) {
    for (int col = 0; col < width; ++col) {
      // Cada píxel tiene 3 canales (RGB)
      int8_t r = data[index++];
      int8_t g = data[index++];
      int8_t b = data[index++];

      // Reescalado de [0, 127] a [0, 255] para que OpenCV lo muestre correctamente
      uint8_t R = static_cast<uint8_t>((r / 64.0) * 255.0);
      uint8_t G = static_cast<uint8_t>((g / 64.0) * 255.0);
      uint8_t B = static_cast<uint8_t>((b / 64.0) * 255.0);

      // Asignamos los valores al píxel en la imagen
      image.at<cv::Vec3b>(row, col) = cv::Vec3b(B, G, R);  // OpenCV usa BGR por defecto
    }
  }

  // Mostrar la imagen usando OpenCV
  cv::imshow("Imagen Resultante", image);
  cv::waitKey(0);  // Espera hasta que se presione una tecla
}


// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor) {
 return tensor->has_attr("fix_point") ? std::exp2f(1.0f * tensor->get_attr<int>("fix_point"))
: 1.0f;
}

static float get_output_scale(const xir::Tensor* tensor) {
      return tensor->has_attr("fix_point") ? std::exp2f(-1.0f * tensor-
>get_attr<int>("fix_point")) : 1.0f;
}

bool canDisplayImages() {
  //std::cout << "Esta comprobando si puede usar el display" << std::endl;
  try {
```

```
      cv::namedWindow("test", cv::WINDOW_AUTOSIZE);
      cv::destroyWindow("test");
      //return false;
      return true;
    } catch (const cv::Exception& e) {
      std::cout << "No se pueden mostrar imágenes en este entorno." << std::endl;
      return false;
    }
}

void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences,  const std::vector<int>& indices, const std::vector<int>& class_ids) {
                                    std::vector<std::string>                class_names                =
{"person","bicycle","car","motorcycle","airplane","bus","train","truck","boat","traffic
light","fire                hydrant","stop                sign","parking
meter","bench","bird","cat","dog","horse","sheep","cow","elephant","bear","zebra","giraffe",
"backpack","umbrella","handbag","tie","suitcase","frisbee","skis","snowboard","sports
ball","kite","baseball          bat","baseball          glove","skateboard","surfboard","tennis
racket","bottle","wine
glass","cup","fork","knife","spoon","bowl","banana","apple","sandwich","orange","broccoli",
"carrot","hot      dog","pizza","donut","cake","chair","couch","potted      plant","bed","dining
table","toilet","tv","laptop","mouse","remote","keyboard","cell
phone","microwave","oven","toaster","sink","refrigerator","book","clock","vase","scissors","t
eddy bear","hair drier","toothbrush"};
  std::cout << indices.size() << " objetos detectados:\n";
  for (auto& indx : indices) {
    std::cout << "Clase: " << class_names[class_ids[indx]]
          << ", Confianza: " << confidences[indx]
          << ", Caja: [" << boxes[indx].x << ", " << boxes[indx].y << ", "
          << boxes[indx].x + boxes[indx].width << ", " << boxes[indx].y + boxes[indx].height
<< "]\n";
  }
}

//DEBUGGING

// Función para imprimir el contenido de un tensor

static void print_tensor(vart::TensorBuffer* tensor_buffer) {

  // Obtener dimensiones del tensor para reconocer el tipo de tensor
  auto tensor = tensor_buffer->get_tensor();
  auto shape = tensor->get_shape();

  std::cout << "Estas son las dimensiones del tensor: " << std::endl;
  for( auto& dim : shape){
    std::cout << " " << dim;
  }
  std::cout << std::endl;

  // Obtener la escala con la función adecuada dependiendo de si es entrada o salida
  float scale = 1.0f;
```

```
  if (shape[3] == 80) scale = get_output_scale(tensor);
  if (shape[3] == 4) scale = get_output_scale(tensor);
  if (shape[3] == 3) scale = get_input_scale(tensor);

  // Obtener los datos del tensor de forma segura mediante la función data()
  uint64_t data_addr;
  size_t data_size;

  // Obtener el puntero y tamaño de los datos usando un índice válido (ej: {0, 0, 0, 0})
  std::tie(data_addr, data_size) = tensor_buffer->data(std::vector<int>{0, 0, 0, 0});

  // Convertir el puntero a int8_t*
  int8_t* tensor_data = (int8_t*)(data_addr);

  // Obtener el tamaño del tensor y comprobar que no está vacío.
  size_t tensor_size = tensor->get_element_num();
  CHECK_NE(tensor_size, 0u);

  // Comprobación de las dimensiones del tensor
  if (shape[3] == 80) std::cout << "Es vector de clases y su escala es: 1/" << 1.0f/scale <<
std::endl;
   if (shape[3] == 4) std::cout << "Es vector de cajas y su escala es: 1/" << 1.0f/scale <<
std::endl;
    if (shape[3] == 3) std::cout << "Es vector de entrada y su escala es: " << scale <<
std::endl;

  // Recorrer algunos valores del tensor y convertirlos a float
  for (size_t i = 0; i < 9; ++i) {
   if (shape[3] == 80) {
      float valor_float = 1.0f / (1.0f + std::exp(-(tensor_data[i] * scale)));
      std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
   } else if (shape[3] == 4) {
      float valor_float = tensor_data[i] * scale;
      std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
   } else if (shape[3] == 3) {
      float valor_float = tensor_data[i] / scale * 255;
      std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
   } else {
      std::cout << "No es un vector conocido" << std::endl;
   }
  }
  for (size_t i = 0; i < tensor_size; ++i) {
   if (shape[3] == 80) {
      float valor_float = 1.0f / (1.0f + std::exp(-(tensor_data[i] * scale)));
        if (valor_float >= 0.35) std::cout << "Valor [" << i/80.0f << "]: " << valor_float <<
std::endl;
   } else {
      std::cout << "No es un vector conocido" << std::endl;
      break;
   }
  }
}
```

```
// Sigmoid

float sigmoid(float x) {
   return 1.0f / (1.0f + std::exp(-x));
}

//MAIN

int main(int argc, char* argv[]) {
 //DECLARACIÓN DE PARAMETROS
 int NUMBER_OF_CLASSES = 80;
 float CONFIDENCE_THRESHOLD = 0.45;


 if (argc < 3) {
  std::cout << "usage: " << argv[0]
        << " <yolov6.xmodel> sample_image [sample_image ...] \n";
  return 0;
 }

 auto xmodel_file = std::string(argv[1]);
 std::vector<cv::Mat> input_images;

 for (auto i = 2; i < argc; i++) {
  cv::Mat img = cv::imread(argv[i]);
  if (img.empty()) {
   std::cout << "Cannot load image : " << argv[i] << std::endl;
   continue;
  }
  input_images.push_back(img);
 }

 if (input_images.empty()) {
  std::cerr << "No image load success!" << std::endl;
  abort();
 }


 // Crear runner para DPU
 auto graph = xir::Graph::deserialize(xmodel_file);
 auto root = graph->get_root_subgraph();
 xir::Subgraph* subgraph = nullptr;
 for (auto c : root->children_topological_sort()) {
  CHECK(c->has_attr("device"));
  if (c->get_attr<std::string>("device") == "DPU") {
   subgraph = c;
   break;
  }
 }
 if (subgraph == nullptr) {
 std::cerr << "Error: No se encontró un subgrafo DPU." << std::endl;
```

```
 return -1;
 }

 auto attrs = xir::Attrs::create();
 std::unique_ptr<vart::RunnerExt> runner =
   vart::RunnerExt::create_runner(subgraph, attrs.get());

 if (!runner) {
 std::cerr << "Error: No se pudo crear el runner." << std::endl;
 return -1;
 }
 //ENTRADAS
 auto inputTensors = runner->get_input_tensors();
 auto inputTensorBuffers = runner->get_inputs();

 //SALIDAS
 auto outputTensors = runner->get_output_tensors();
 auto outputTensorBuffers = runner->get_outputs();

 auto itb = inputTensorBuffers[0];
 int input_tensor_size = inputTensors[0]->get_element_num();
 if (input_tensor_size <= 0) {
     std::cerr << "Error: Tensor 0 está vacío!" << std::endl;
 }

  // Revisamos para el tensor de entrada el componente fixed_point para la función
setImageRGB
 auto it = itb->get_tensor();
 auto input_scale = get_input_scale(it);

 auto batch = inputTensors[0]->get_shape().at(0);
 auto height = inputTensors[0]->get_shape().at(1);
 auto width = inputTensors[0]->get_shape().at(2);

 for (auto i = 0; i < input_images.size(); i += batch) {
  auto run_batch = std::min(((int)input_images.size() - i), batch);
  auto images = std::vector<cv::Mat>(run_batch);

  // Preprocesamiento
  std::vector<uint64_t> data_in(run_batch);
  std::vector<size_t> size_in(run_batch);
  //Preparar los colores de la imagen
  for (auto batch_idx = 0; batch_idx < run_batch; ++batch_idx) {
      images[batch_idx] = preprocess_image(input_images[i + batch_idx], cv::Size(width,
height));
              std::tie(data_in[batch_idx], size_in[batch_idx]) = inputTensorBuffers[0]-
>data({batch_idx, 0, 0, 0});
     CHECK_NE(size_in[batch_idx], 0u);
     setImageRGB(images[batch_idx], (void*)data_in[batch_idx], input_scale);
  }

  //Sincronizamos el input con la CPU/DPU para que lo reciba
```

```
  for (auto& input : inputTensorBuffers) {
      input->sync_for_write(0, input->get_tensor()->get_data_size() / input->get_tensor()->get_shape()[0]);
  }

  auto v = runner->execute_async(inputTensorBuffers, outputTensorBuffers);
  auto status = runner->wait(v.first, -1);
    CHECK_EQ(status, 0) << "fallo en la ejecución de DPU";

  for (auto& output : outputTensorBuffers) {
          output->sync_for_read(0, output->get_tensor()->get_data_size() / output->get_tensor()->get_shape()[0]);
  }


   //EMPEZAMOS A ANALIZAR LOS RESULTADOS DESPUÉS DE SINCRONIZAR LOS OUTPUTS
  // Postprocesado modificado
  std::vector<TensorData> class_probabilities_vec;
  std::vector<TensorData> boxes_vec;

  for (size_t idx = 0; idx < outputTensorBuffers.size(); ++idx) {
    auto tb = outputTensorBuffers[idx];
    int tensor_size = outputTensors[idx]->get_element_num();

    if (tensor_size <= 0) {
      std::cerr << "Error: Tensor " << idx << " está vacío!" << std::endl;
      continue;
    }

    std::uint64_t data_addr;
    std::tie(data_addr, std::ignore) = tb->data({0, 0, 0, 0});

    if (data_addr == 0) {
      std::cerr << "Error: Dirección de datos vacía para el tensor " << idx << std::endl;
      continue;
    }

    int8_t* data_ptr = (int8_t*)(data_addr);

    // Obtener las dimensiones del tensor
    auto shape = outputTensors[idx]->get_shape();
    auto ot = outputTensors[idx];
    int first_dim = shape.front();
    int last_dim = shape.back();
    int second_dim = shape[shape.size() - 2];
    int third_dim = shape[shape.size() - 3];

    // Estructura para almacenar los datos con su forma original
    TensorData tensor_data;
    tensor_data.shape = shape;
```

```
    tensor_data.data.resize(first_dim);
    for (int ii = 0; ii < first_dim; ++ii) {
      tensor_data.data[ii].resize(second_dim);
        for (int j = 0; j < second_dim; ++j) {
          tensor_data.data[ii][j].resize(third_dim);
          for (int k = 0; k < third_dim; ++k) {
            tensor_data.data[ii][j][k].resize(last_dim);
          }
        }
    }

    // Copiar los datos del tensor en la estructura manteniendo la forma original
    int index = 0;
    for (int ii = 0; ii < first_dim; ++ii) {
      for (int j = 0; j < second_dim; ++j) {
        for (int k = 0; k < third_dim; ++k) {
          for (int l = 0; l < last_dim; ++l) {

            //Usando la escala convertimos el valor int8_t a float
            // Aplicar Sigmoid a cada valor del tensor y multiplicar por la escala
            float output_scale = get_output_scale(ot);
            float value = data_ptr[index++] * output_scale;

            // Aplicar Sigmoid a cada valor del tensor de clases
            if (last_dim == 80 ) value = sigmoid(value);
            tensor_data.data[ii][j][k][l] = value;
          }
        }
      }
    }

    // Clasificar los tensores
    if (last_dim == 80) {
        class_probabilities_vec.push_back(tensor_data);
    } else if (last_dim == 4 || last_dim == 5) {
      boxes_vec.push_back(tensor_data);
    }
  }


  //EMPEZAMOS EMPAREJAMIENTO
  std::vector<std::tuple<TensorData, TensorData>> tensor_pairs;

  for (size_t i = 0; i < class_probabilities_vec.size(); ++i) {
    auto& class_probs = class_probabilities_vec[i];
    auto& boxes = boxes_vec[i];

    // Nos aseguramos de que las dimensiones coincidan
        if (class_probs.shape[1] == boxes.shape[1] && class_probs.shape[2] ==
boxes.shape[2]) {
      tensor_pairs.push_back(std::make_tuple(class_probs, boxes));
    } else {
```

```
        std::cerr << "Error: Dimensiones no coinciden entre probabilidad de clases y boxes."
<< std::endl;
      }
   }

   //MANDAMOS PAREJAS AL POSTPROCESAMIENTO Y PINTAMOS LAS DETECCIONES EN
LA IMAGEN DE ENTRADA

   // Procesar las detecciones
   for (auto& image : images) {
            process_detections(tensor_pairs , CONFIDENCE_THRESHOLD , image
,NUMBER_OF_CLASSES);
      std::cout << "Ha completado el postprocesamiento" << std::endl;
   }


 }

 return 0;
}
```

Anexo 10. YOLOv6m_APP.cpp

```
#include <glog/logging.h>
#include <algorithm>
#include <cmath>
#include <functional>
#include <iomanip>
#include <iostream>
#include <memory>
#include <numeric>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xir/graph/graph.hpp>
#include "vart/runner.hpp"
#include "vart/runner_ext.hpp"
#include "vitis/ai/collection_helper.hpp"

//RAROS
#include <vector>
#include <tuple>
#include <vart/tensor_buffer.hpp>
#include <xir/tensor/tensor.hpp>
//#include "common.h"
//#include <opencv2/dnn.hpp>

struct TensorData {
   std::vector<std::vector<std::vector<std::vector<float>>>> data;
   std::vector<int> shape;
};
```

```
// Funciones de preprocesamiento
static cv::Mat preprocess_image(const cv::Mat& image, cv::Size size);
static void setImageRGB(const cv::Mat& image, void* data, float fix_scale);

// Funciones de postprocesamiento

static void process_detections(const std::vector<TensorData>& tensor_pairs, float confidence_threshold, cv::Mat& image, int nc);
void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, float confidence_threshold, float iou_threshold, std::vector<int>& indices);
float IoU(const cv::Rect& box1, const cv::Rect& box2);
void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& class_ids, float confidence_threshold, float iou_threshold, std::vector<int>& indices, int nc);

// Funciones adicionales
// Imprimir la imagen a partir del puntero
static void print_input_image(const int8_t* data);
// Ver los vectores
static void print_tensor(vart::TensorBuffer* tensor_buffer);
// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor);
static float get_output_scale(const xir::Tensor* tensor);
// Comprobar si se pueden generar imagenes
bool canDisplayImages();
void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& indices, const std::vector<int>& class_ids);
//Sigmoid
float sigmoid(float x);

//PREPROCESAMIENTO

// Preprocesamiento de imágenes

static cv::Mat preprocess_image(const cv::Mat& image, cv::Size size) {
 // Crear un fondo gris de tamaño `size` (640x640)
 cv::Mat output_image(size, CV_8UC3, cv::Scalar(128, 128, 128));

 // Calcular la escala manteniendo la proporción
    double scale = std::min(static_cast<double>(size.width) / image.cols, static_cast<double>(size.height) / image.rows);

 // Calcular el nuevo tamaño de la imagen
  cv::Size new_size(static_cast<int>(image.cols * scale), static_cast<int>(image.rows * scale));

 // Redimensionar la imagen manteniendo la proporción
 cv::Mat resized_image;
 cv::resize(image, resized_image, new_size);

  // Calcular las coordenadas para centrar la imagen redimensionada dentro del fondo gris
```

```
 int top_left_x = (size.width - new_size.width) / 2;
 int top_left_y = (size.height - new_size.height) / 2;

 // Pegar la imagen redimensionada dentro del fondo gris
  resized_image.copyTo(output_image(cv::Rect(top_left_x,  top_left_y,  new_size.width,
new_size.height)));;

 return output_image;
}

static void setImageRGB(const cv::Mat& image, void* data, float fix_scale) {
  int8_t* data1 = (int8_t*)data;
  int c = 0;
  for (int row = 0; row < image.rows; row++) {
    for (int col = 0; col < image.cols; col++) {
      auto v = image.at<cv::Vec3b>(row, col);
      auto B = (float)v[0];
      auto G = (float)v[1];
      auto R = (float)v[2];

      // Normalización
      auto nB = B / 255.0f * fix_scale;
      auto nG = G / 255.0f * fix_scale;
      auto nR = R / 255.0f * fix_scale;

      // Clamping
      nB = std::max(std::min(nB, fix_scale), 0.0f);
      nG = std::max(std::min(nG, fix_scale), 0.0f);
      nR = std::max(std::min(nR, fix_scale), 0.0f);

      data1[c++] = (int8_t)(nR);
      data1[c++] = (int8_t)(nG);
      data1[c++] = (int8_t)(nB);
    }
  }

  //print_input_image(data1);

}

/*static void setImageRGB(const cv::Mat& image, void* data, float fix_scale) {
  int h = image.rows;
  int w = image.cols;
  int c = image.channels();

  // Acceso directo a los datos de la imagen en formato cv::Mat
  unsigned char* img_data = image.data;

  // Recorrer los píxeles y reorganizar los canales de la imagen mientras normalizamos
  std::vector<float> bb(h * w * c);
  for (int i = 0; i < h; ++i) {
    for (int j = 0; j < w; ++j) {
```

```
        for (int k = 0; k < c; ++k) {
          bb[k * w * h + i * w + j] = img_data[i * w * c + j * c + k] / 255.0f;
        }
      }
    }

    // Intercambiar canales (específicamente 1º y 3º) (RGB a BGR)
    for (int i = 0; i < w * h; ++i) {
      float swap = bb[i];
      bb[i] = bb[i + w * h * 2];
      bb[i + w * h * 2] = swap;
    }

    // Convertir los valores float a int8_t con la escala proporcionada
    float scale = pow(2, 7);
    int size = h * w * c;
    int8_t* output_data = reinterpret_cast<int8_t*>(data);
    for (int i = 0; i < size; ++i) {
      output_data[i] = static_cast<int8_t>(bb[i] * fix_scale);
      if (output_data[i] < 0) {
        output_data[i] = static_cast<int8_t>((127 / scale) * fix_scale);
      }
    }

    //print_input_image(output_data);
}*/

//POSTPROCESAMIENTO

// Filtrar detecciones basadas en el umbral de confianza y aplicar NMS
static void process_detections(const std::vector<TensorData>& outputTensors, float confidence_threshold, cv::Mat& image, int nc) {
    std::vector<cv::Rect> boxes;
    std::vector<int> class_ids;
    std::vector<float> confidences;
    std::vector<std::string> class_names = {"PERSON", "BICYCLE", "CAR", "MOTORCYCLE", "AIRPLANE", "BUS", "TRAIN", "TRUCK", "BOAT", "TRAFFIC LIGHT", "FIRE HYDRANT", "STOP SIGN", "PARKING METER", "BENCH", "BIRD", "CAT", "DOG", "HORSE", "SHEEP", "COW", "ELEPHANT", "BEAR", "ZEBRA", "GIRAFFE", "BACKPACK", "UMBRELLA", "HANDBAG", "TIE", "SUITCASE", "FRISBEE", "SKIS", "SNOWBOARD", "SPORTS BALL", "KITE", "BASEBALL BAT", "BASEBALL GLOVE", "SKATEBOARD", "SURFBOARD", "TENNIS RACKET", "BOTTLE", "WINE GLASS", "CUP", "FORK", "KNIFE", "SPOON", "BOWL", "BANANA", "APPLE", "SANDWICH", "ORANGE", "BROCCOLI", "CARROT", "HOT DOG", "PIZZA", "DONUT", "CAKE", "CHAIR", "COUCH", "POTTED PLANT", "BED", "DINING TABLE", "TOILET", "TV", "LAPTOP", "MOUSE", "REMOTE", "KEYBOARD", "CELL PHONE", "MICROWAVE", "OVEN", "TOASTER", "SINK", "REFRIGERATOR", "BOOK", "CLOCK", "VASE", "SCISSORS", "TEDDY BEAR", "HAIR DRIER", "TOOTHBRUSH"};
    int counter=0;

    for (const auto& tensor : outputTensors) {

        // Determinar el stride según las dimensiones del tensor de cajas (En principio no
```

```
hay)
     int stride = (tensor.shape[1] == 80) ? 8 : (tensor.shape[1] == 40) ? 16 : (tensor.shape[1]
== 20) ? 32 : 1;
    int aux = (tensor.shape[1] == 80) ? 0 : (tensor.shape[1] == 40) ? 1 : (tensor.shape[1] ==
20) ? 2 : -1;
    //std::vector<float> biases = {0.5, 0.75, 0.75, 1.5, 1.5, 1.25, 0.75, 1.5, 1.5, 1.25, 1.5,
2.25,  1.5, 1.25, 1.75, 2.0, 2.75, 2.75};
    std::vector<float> biases{12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192,
243, 459, 401};

      for (int i = 0; i < tensor.shape[0]; ++i) {
        for (int j = 0; j < tensor.shape[1]; ++j) {
          for (int k = 0; k < tensor.shape[2]; ++k) {
            for (int anch = 0; anch < tensor.shape[3]/(nc+5); ++anch){

              float confidence = sigmoid(tensor.data[i][j][k][4+(nc+5)*anch]); //Por lo general
en YOLO va despues de las dimensiones de las cajas
                if (confidence > confidence_threshold) {

                  if (confidence == 1) counter++;
                   //std::cout << "Ha detectado un objeto, conf= " << confidence << std::endl;

                  // Extraer las coordenadas de la caja que están en formato
                  //x_center, y_center, width y height
                  float y1 = (sigmoid(tensor.data[i][j][k][0+(nc+5)*anch]) + j) * stride;
                  float x1 = (sigmoid(tensor.data[i][j][k][1+(nc+5)*anch]) + k) * stride;
                     //float w1 = std::pow(sigmoid(tensor.data[i][j][k][2+(nc+5)*anch])*4,2) *
stride;
                      //float h1 = std::pow(sigmoid(tensor.data[i][j][k][3+(nc+5)*anch])*4,2) *
stride;
                           float w1 = std::exp(sigmoid(tensor.data[i][j][k][2+(nc+5)*anch])) *
biases[aux*6+anch*2];
                           float h1 = std::exp(sigmoid(tensor.data[i][j][k][3+(nc+5)*anch])) *
biases[aux*6+anch*2+1];
                  float left1 = (x1 - w1 / 2);
                  float top1 = (y1 - h1 / 2);

                  // Comprobar si las dimensiones son válidas (ancho y alto mayores que 4)
                  if ((w1 <= 4 || h1 <= 4) || (w1/h1>4 || h1/w1>4)) continue;
                  float max_score = 0;
                  int class_id = -1;
                  for (int c = 4+(nc+5)*anch; c < 84+(nc+5)*anch; ++c) {
                    float class_confidence = sigmoid(tensor.data[i][j][k][c]);
                    if (class_confidence > max_score){
                      max_score = class_confidence;
                      class_id = c - 5 - (nc+5)*anch;
                    }
                  }

                  if(class_id >= 0) {
                    //std::cout << "Iterando sobre clase [" << class_names[class_id] << "] con
confianza: " << std::fixed << std::setprecision(2) << max_score*confidence <<" ["<<x1<<",
```

```
"<<y1<<", "<<w1<<", "<<h1<<"]"<<std::endl;

                    //Guardamos la caja como objecto Rect
                    boxes.emplace_back(cv::Rect(left1, top1, w1, h1));
                    //Guardamos la id de la clase y su confianza para el print
                    class_ids.push_back(class_id);
                    confidences.push_back(confidence*max_score);
                }
              }
            }
          }
        }
      }
  }

  //filtro de cajas
  for(auto& box : boxes){

    float top = box.y;
    float left = box.x;
    float bottom = box.y + box.height;
    float right = box.x + box.width;

    if (top < 0) top = 0;
    if (left < 0) left = 0;
    if (bottom > 640) bottom = 640;
    if (right > 640) right = 640;

    box.y = top;
    box.x = left;
    box.height = bottom - top;
    box.width = right - left;

  }


   //Si detecta muchas clases con una confianza de 1 podemos suponer que la salida
está mal procesada
  std::cout << "Ha detectado tantas clases superiores a 1: " << counter << std::endl;

  // Aplicar NMS (supresión de no máximos) para eliminar detecciones redundantes
  std::vector<int> indices;
  //for (int a=0; a<confidences.size(); ++a) indices.push_back(a);
  //NMSBoxesCustom(boxes, confidences, confidence_threshold, 0.05, indices);
     NMSBoxesPerClass(boxes, confidences, class_ids, confidence_threshold, 0.05,
indices, nc);

  // Dibujar las cajas de detección en la imagen
  // se puede poner dentro del condicional esto también, pero igual se queda pillado
  //getenv("DISPLAY") != nullptr

  std::vector<cv::Scalar> colors;
```

```
  for (int i = 0; i < nc; ++i) {
    int r = (i * 37) % 255;  // Cálculo cíclico para rojo
    int g = (i * 73) % 255;  // Cálculo cíclico para verde
    int b = (i * 7) % 255;  // Cálculo cíclico para azul
    colors.emplace_back(cv::Scalar(b, g, r));
  }

  if (canDisplayImages()) {
    std::cout << "Ha detectado que puede imprimir imágenes" << std::endl;
    for (int idx : indices) {
      cv::rectangle(image, boxes[idx], colors[class_ids[idx]] , 2);
                        std::string label = class_names[class_ids[idx]]+ " " +
cv::format("%.1f",confidences[idx]);
      int baseline = 0;
        cv::Size textSize = cv::getTextSize(label, cv::FONT_HERSHEY_SIMPLEX, 0.5, 1,
&baseline);
      int rect_top = std::max(boxes[idx].y - textSize.height - 5, 0);
      int rect_left = boxes[idx].x;
      int rect_width = textSize.width + 10;
      int rect_height = textSize.height + 5;
       cv::rectangle(image, cv::Point(rect_left, rect_top), cv::Point(rect_left + rect_width,
rect_top + rect_height), colors[class_ids[idx]], cv::FILLED);
          cv::putText(image, label, cv::Point(rect_left + 5, rect_top + textSize.height),
cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255, 255, 255), 1);
          //cv::putText(image, label, boxes[idx].tl(), cv::FONT_HERSHEY_SIMPLEX, 0.5,
colors[class_ids[idx]], 2);
    }
    printDetections(boxes, confidences, indices, class_ids);
    cv::imshow("Detecciones", image);
    cv::waitKey(0);
  } else {
    std::cout << "Ha detectado que no puede imprimir imágenes" << std::endl;
    printDetections(boxes, confidences, indices, class_ids);
  }


}

float IoU(const cv::Rect& box1, const cv::Rect& box2) {
  // Suponiendo que Box tiene las propiedades x1, y1, x2, y2
  float x1_intersection = std::max(box1.x, box2.x);
  float y1_intersection = std::max(box1.y, box2.y);
  float x2_intersection = std::min(box1.x + box1.width, box2.x + box2.width);
  float y2_intersection = std::min(box1.y + box1.height, box2.y + box2.height);

  // Calcular el área de intersección
  float intersection_area = std::max(0.0f, x2_intersection - x1_intersection) *
                std::max(0.0f, y2_intersection - y1_intersection);

  // Calcular el área de las cajas
  float area1 = box1.width * box1.height;
  float area2 = box2.width * box2.height;
```

```
  // Calcular el área de unión
  float union_area = area1 + area2 - intersection_area;

  // Evitar división por cero
  if (union_area <= 0) return 0.0f;

  // Retornar IoU
  return intersection_area / union_area;
}

void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences, float confidence_threshold, float iou_threshold, std::vector<int>& indices)
{
  // Crear un vector de índices que clasifique las cajas por confianza (de mayor a menor)
  std::vector<int> sorted_indices(confidences.size());
  std::iota(sorted_indices.begin(), sorted_indices.end(), 0);  // Rellena con [0, 1, 2, ..., n]
  int counter = 0;

  // Ordenar los índices basados en las puntuaciones de confianza
  std::sort(sorted_indices.begin(), sorted_indices.end(),
        [&](int i, int j) { return confidences[i] > confidences[j]; });

  //De la caja con mayor confianza que quede en la lista elimina aquellas que superen la
intersección
  while (!sorted_indices.empty()) {
    int best_index = sorted_indices[0];  // Mejor caja (la de mayor confianza)
    indices.push_back(best_index);      // Añadir el índice de la mejor caja
    sorted_indices.erase(sorted_indices.begin());  // Eliminar la mejor caja de la lista

    // Comparar el resto de cajas con la mejor caja usando el umbral IoU
    for (auto it = sorted_indices.begin(); it != sorted_indices.end();) {
      int idx = *it;
      float iou = IoU(boxes[best_index], boxes[idx]);

      // Si el IoU es mayor que el umbral, eliminamos la caja actual
      if (iou > iou_threshold) {
        it = sorted_indices.erase(it);
      } else {
        ++it;
      }
    }
  }
  //std::cout << "Ha leido " << counter << " cajas" << std::endl;

}

void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences, const std::vector<int>& class_ids, float confidence_threshold, float
iou_threshold, std::vector<int>& indices, int nc) {

  // Iterar por cada clase
```

```
  for (int c = 0; c < nc; ++c) {
    std::vector<int> class_indices;

    // Recoger índices de la clase actual
    for (size_t i = 0; i < class_ids.size(); ++i) {
      if (class_ids[i] == c && confidences[i] > confidence_threshold) {
        class_indices.push_back(i);
      }
    }

    // Ordenar por confianza
    std::sort(class_indices.begin(), class_indices.end(),
          [&](int i, int j) { return confidences[i] > confidences[j]; });

    while (!class_indices.empty()) {
      int best_index = class_indices[0];
      indices.push_back(best_index);
      class_indices.erase(class_indices.begin());

      for (auto it = class_indices.begin(); it != class_indices.end();) {
        int idx = *it;
        float iou = IoU(boxes[best_index], boxes[idx]);

        if (iou > iou_threshold) {
          it = class_indices.erase(it);
        } else {
          ++it;
        }
      }
    }
  }
}

//FUNCIONES ADICIONALES

// Imprimir la imagen de entrada a la DPU
static void print_input_image(const int8_t* data) {
  // Dimensiones de la imagen
  const int height = 640;
  const int width = 640;
  const int channels = 3;  // RGB

  // Crear un objeto cv::Mat para almacenar la imagen resultante (8-bit, 3 canales)
  cv::Mat image(height, width, CV_8UC3);

  // Recorremos el buffer de datos y convertimos a valores de imagen
  int index = 0;
  for (int row = 0; row < height; ++row) {
    for (int col = 0; col < width; ++col) {
      // Cada píxel tiene 3 canales (RGB)
      int8_t r = data[index++];
      int8_t g = data[index++];
```

```
        int8_t b = data[index++];

        // Reescalado de [0, 127] a [0, 255] para que OpenCV lo muestre correctamente
        uint8_t R = static_cast<uint8_t>((r / 64.0) * 255.0);
        uint8_t G = static_cast<uint8_t>((g / 64.0) * 255.0);
        uint8_t B = static_cast<uint8_t>((b / 64.0) * 255.0);

        // Asignamos los valores al píxel en la imagen
        image.at<cv::Vec3b>(row, col) = cv::Vec3b(B, G, R);  // OpenCV usa BGR por defecto
      }
    }

    // Mostrar la imagen usando OpenCV
    cv::imshow("Imagen Resultante", image);
    cv::waitKey(0);  // Espera hasta que se presione una tecla
}


// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor) {
  return tensor->has_attr("fix_point") ? std::exp2f(1.0f * tensor->get_attr<int>("fix_point"))
: 1.0f;
}

static float get_output_scale(const xir::Tensor* tensor) {
        return tensor->has_attr("fix_point") ? std::exp2f(-1.0f * tensor-
>get_attr<int>("fix_point")) : 1.0f;
}

bool canDisplayImages() {
    //std::cout << "Esta comprobando si puede usar el display" << std::endl;
    try {
        cv::namedWindow("test", cv::WINDOW_AUTOSIZE);
        cv::destroyWindow("test");
        //return false;
        return true;
    } catch (const cv::Exception& e) {
        std::cout << "No se pueden mostrar imágenes en este entorno." << std::endl;
        return false;
    }
}

void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences,  const std::vector<int>& indices, const std::vector<int>& class_ids) {
                                std::vector<std::string> class_names =
{"person","bicycle","car","motorcycle","airplane","bus","train","truck","boat","traffic
light","fire hydrant","stop sign","parking
meter","bench","bird","cat","dog","horse","sheep","cow","elephant","bear","zebra","giraffe",
"backpack","umbrella","handbag","tie","suitcase","frisbee","skis","snowboard","sports
ball","kite","baseball bat","baseball glove","skateboard","surfboard","tennis
racket","bottle","wine
glass","cup","fork","knife","spoon","bowl","banana","apple","sandwich","orange","broccoli",
```

```
"carrot","hot dog","pizza","donut","cake","chair","couch","potted plant","bed","dining table","toilet","tv","laptop","mouse","remote","keyboard","cell phone","microwave","oven","toaster","sink","refrigerator","book","clock","vase","scissors","teddy bear","hair drier","toothbrush"};
  std::cout << indices.size() << " objetos detectados:\n";
  for (auto& indx : indices) {
    std::cout << "Clase: " << class_names[class_ids[indx]]
          << ", Confianza: " << confidences[indx]
          << ", Caja: [" << boxes[indx].x << ", " << boxes[indx].y << ", "
          << boxes[indx].x + boxes[indx].width << ", " << boxes[indx].y + boxes[indx].height << "]\n";
  }
}

//DEBUGGING

// Función para imprimir el contenido de un tensor

static void print_tensor(vart::TensorBuffer* tensor_buffer) {

  // Obtener dimensiones del tensor para reconocer el tipo de tensor
  auto tensor = tensor_buffer->get_tensor();
  auto shape = tensor->get_shape();

  //std::cout << "Estas son las dimensiones del tensor: " << std::endl;
  for( auto& dim : shape){
    std::cout << " " << dim;
  }
  std::cout << std::endl;

  // Obtener la escala con la función adecuada dependiendo de si es entrada o salida
  float scale = 1.0f;
  if (shape[3] == 255) scale = get_output_scale(tensor);
  if (shape[3] == 3) scale = get_input_scale(tensor);

  // Obtener los datos del tensor de forma segura mediante la función data()
  uint64_t data_addr;
  size_t data_size;

  // Obtener el puntero y tamaño de los datos usando un índice válido (ej: {0, 0, 0, 0})
  std::tie(data_addr, data_size) = tensor_buffer->data(std::vector<int>{0, 0, 0, 0});

  // Convertir el puntero a int8_t*
  int8_t* tensor_data = (int8_t*)(data_addr);

  // Obtener el tamaño del tensor y comprobar que no está vacío.
  size_t tensor_size = tensor->get_element_num();
  CHECK_NE(tensor_size, 0u);

  // Comprobación de las dimensiones del tensor
  if (shape[3] == 255) std::cout << "Es vector de salida y su escala es: 1/" << 1.0f/scale << std::endl;
```

```
    if (shape[3] == 3) std::cout << "Es vector de entrada y su escala es: " << scale << std::endl;

   // Recorrer algunos valores del tensor y convertirlos a float
   for (size_t i = 0; i < 9; ++i) {
    if (shape[3] == 255) {
       float valor_float = 1.0f / (1.0f + std::exp(-(tensor_data[i] * scale)));
       //std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
    } else if (shape[3] == 3) {
       float valor_float = tensor_data[i] / scale * 255;
       //std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
    } else {
       //std::cout << "No es un vector conocido" << std::endl;
    }
   }
   int counter = 0;
   for (size_t i = 0; i < tensor_size; ++i) {
    if (shape[3] == 255) {
       float valor_float = 1.0f / (1.0f + std::exp(-(tensor_data[i] * scale)));
        //if (valor_float >= 0.35) std::cout << "Valor [" << i/250.0f << "]: " << valor_float << std::endl;
       if (valor_float >= 0.35) counter++;
       if (counter > 10) break;
    } else {
       //std::cout << "No es un vector conocido" << std::endl;
       break;
    }
   }
}

// Sigmoid

float sigmoid(float x) {
   return 1.0f / (1.0f + std::exp(-x));
}

//MAIN

int main(int argc, char* argv[]) {
 //DECLARACIÓN DE PARAMETROS
 int NUMBER_OF_CLASSES = 80;
 float CONFIDENCE_THRESHOLD = 0.4;


 if (argc < 3) {
  std::cout << "usage: " << argv[0]
        << " <yolov_4_or_7.xmodel> sample_image [sample_image ...] \n";
  return 0;
 }

 auto xmodel_file = std::string(argv[1]);
 std::vector<cv::Mat> input_images;
```

```
for (auto i = 2; i < argc; i++) {
  cv::Mat img = cv::imread(argv[i]);
  if (img.empty()) {
    std::cout << "Cannot load image : " << argv[i] << std::endl;
    continue;
  }
  input_images.push_back(img);
}

if (input_images.empty()) {
  std::cerr << "No image load success!" << std::endl;
  abort();
}


// Crear runner para DPU
auto graph = xir::Graph::deserialize(xmodel_file);
auto root = graph->get_root_subgraph();
xir::Subgraph* subgraph = nullptr;
for (auto c : root->children_topological_sort()) {
  CHECK(c->has_attr("device"));
  if (c->get_attr<std::string>("device") == "DPU") {
    subgraph = c;
    break;
  }
}
if (subgraph == nullptr) {
std::cerr << "Error: No se encontró un subgrafo DPU." << std::endl;
return -1;
}

auto attrs = xir::Attrs::create();
std::unique_ptr<vart::RunnerExt> runner =
   vart::RunnerExt::create_runner(subgraph, attrs.get());

if (!runner) {
std::cerr << "Error: No se pudo crear el runner." << std::endl;
return -1;
}
//ENTRADAS
auto inputTensors = runner->get_input_tensors();
auto inputTensorBuffers = runner->get_inputs();

//SALIDAS
auto outputTensors = runner->get_output_tensors();
auto outputTensorBuffers = runner->get_outputs();

 //Este bucle queda aunque ya haya comprobado que no hace falta ya que solo hay un
tensor de entrada
   auto itb = inputTensorBuffers[0];
   int input_tensor_size = inputTensors[0]->get_element_num();
```

```
  if (input_tensor_size <= 0) {
       std::cerr << "Error: Tensor 0 está vacío!" << std::endl;
  }

  // Revisamos para cada tensor de entrada el componente fixed_point para la función
setImageRGB
  auto it = itb->get_tensor();
  auto input_scale = get_input_scale(it);
  //std::cout << "El valor de input_scale es: " << input_scale << std::endl;

  auto batch = inputTensors[0]->get_shape().at(0);
  auto height = inputTensors[0]->get_shape().at(1);
  auto width = inputTensors[0]->get_shape().at(2);

  for (auto i = 0; i < input_images.size(); i += batch) {
    auto run_batch = std::min(((int)input_images.size() - i), batch);
    auto images = std::vector<cv::Mat>(run_batch);

    // Preprocesamiento
    std::vector<uint64_t> data_in(run_batch);
    std::vector<size_t> size_in(run_batch);
    //Preparar los colores de la imagen
    for (auto batch_idx = 0; batch_idx < run_batch; ++batch_idx) {
      images[batch_idx] = preprocess_image(input_images[i + batch_idx], cv::Size(width,
height));
                std::tie(data_in[batch_idx],  size_in[batch_idx])  =  inputTensorBuffers[0]-
>data({batch_idx, 0, 0, 0});
       CHECK_NE(size_in[batch_idx], 0u);
       setImageRGB(images[batch_idx], (void*)data_in[batch_idx], input_scale);
    }

    //Sincronizamos el input con la CPU/DPU para que lo reciba

    for (auto& input : inputTensorBuffers) {
      input->sync_for_write(0, input->get_tensor()->get_data_size() / input->get_tensor()-
>get_shape()[0]);
    }

  auto v = runner->execute_async(inputTensorBuffers, outputTensorBuffers);
  auto status = runner->wait(v.first, -1);
    CHECK_EQ(status, 0) << "fallo en la ejecución de DPU";

    for (auto& output : outputTensorBuffers) {
              output->sync_for_read(0,  output->get_tensor()->get_data_size()  /  output-
>get_tensor()->get_shape()[0]);
    }

  //std::cout << "Estas son las dimensiones de los tensores de entrada" << std::endl;
  for (auto& input : inputTensorBuffers) {
    //print_tensor(input);  // Imprimir dimensiones
  }
```

```
  //std::cout << "Estas son las dimensiones de los tensores de salida" << std::endl;
  for (auto& output : outputTensorBuffers) {
    //print_tensor(output);  // Imprimir dimensiones
  }

   //EMPEZAMOS A ANALIZAR LOS RESULTADOS DESPUÉS DE SINCRONIZAR LOS OUTPUTS
    // Postprocesado modificado
    std::vector<TensorData> outTens_copy;

    for (size_t idx = 0; idx < outputTensorBuffers.size(); ++idx) {
      auto tb = outputTensorBuffers[idx];
      int tensor_size = outputTensors[idx]->get_element_num();

      if (tensor_size <= 0) {
        std::cerr << "Error: Tensor " << idx << " está vacío!" << std::endl;
        continue;
      }

      std::uint64_t data_addr;
      std::tie(data_addr, std::ignore) = tb->data({0, 0, 0, 0});

      if (data_addr == 0) {
        std::cerr << "Error: Dirección de datos vacía para el tensor " << idx << std::endl;
        continue;
      }

      int8_t* data_ptr = (int8_t*)(data_addr);

      // Obtener las dimensiones del tensor
      auto shape = outputTensors[idx]->get_shape();
        auto ot = outputTensors[idx];
      int first_dim = shape.front();
      int last_dim = shape.back();
      int second_dim = shape[shape.size() - 2];
      int third_dim = shape[shape.size() - 3];

      // Estructura para almacenar los datos con su forma original
      TensorData tensor_data;
      tensor_data.shape = shape;
      tensor_data.data.resize(first_dim);
      for (int ii = 0; ii < first_dim; ++ii) {
        tensor_data.data[ii].resize(second_dim);
          for (int j = 0; j < second_dim; ++j) {
            tensor_data.data[ii][j].resize(third_dim);
            for (int k = 0; k < third_dim; ++k) {
              tensor_data.data[ii][j][k].resize(last_dim);
            }
          }
      }

      // Copiar los datos del tensor en la estructura manteniendo la forma original
```

```
        int index = 0;
        for (int ii = 0; ii < first_dim; ++ii) {
          for (int j = 0; j < second_dim; ++j) {
            for (int k = 0; k < third_dim; ++k) {
              for (int l = 0; l < last_dim; ++l) {

               //Usando la escala convertimos el valor int8_t a float
               // Aplicar Sigmoid a cada valor del tensor y multiplicar por la escala
               float output_scale = get_output_scale(ot);
               float value = data_ptr[index++] * output_scale;

              // Aplicar Sigmoid a cada valor del tensor
              //if (last_dim == 80 ) value = 1.0f / (1.0f + std::exp(-value));
              //value = 1.0f / (1.0f + std::exp(-value));

                   tensor_data.data[ii][j][k][l] = value;
                   }
              }
            }
          }

        // Clasificar los tensores
        outTens_copy.push_back(tensor_data);
      }



    //MANDAMOS PAREJAS AL POSTPROCESAMIENTO Y PINTAMOS LAS DETECCIONES
EN LA IMAGEN DE ENTRADA

    // Procesar las detecciones
    for (auto& image : images) {
                process_detections(outTens_copy, CONFIDENCE_THRESHOLD, image,
NUMBER_OF_CLASSES);
       std::cout << "Ha completado el postprocesamiento" << std::endl;
    }


   }

  return 0;
}
```

Anexo 11. YOLOv4y7_APP.cpp

```
#include <glog/logging.h>
#include <algorithm>
#include <cmath>
#include <functional>
#include <iomanip>
#include <iostream>
```

```
#include <memory>
#include <numeric>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <xir/graph/graph.hpp>
#include "vart/runner.hpp"
#include "vart/runner_ext.hpp"
#include "vitis/ai/collection_helper.hpp"

//RAROS
#include <vector>
#include <tuple>
#include <vart/tensor_buffer.hpp>
#include <xir/tensor/tensor.hpp>
#include "common.h"
//#include <opencv2/dnn.hpp>

struct TensorData {
  std::vector<std::vector<std::vector<std::vector<float>>>> data;
  std::vector<int> shape;
};


// Funciones de preprocesamiento
static cv::Mat preprocess_image(const cv::Mat& image, cv::Size size);
static void setImageRGB(const cv::Mat& image, void* data, float fix_scale);

// Funciones de postprocesamiento

static void process_detections(const std::vector<std::tuple<TensorData, TensorData>>& tensor_pairs, float confidence_threshold, cv::Mat& image, int nc, int delay);
void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, float confidence_threshold, float iou_threshold, std::vector<int>& indices);
float IoU(const cv::Rect& box1, const cv::Rect& box2);
void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& class_ids, float confidence_threshold, float iou_threshold, std::vector<int>& indices, int nc);

// Funciones adicionales
// Imprimir la imagen a partir del puntero
static void print_input_image(const int8_t* data);
// Ver los vectores
static void print_tensor(vart::TensorBuffer* tensor_buffer);
// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor);
static float get_output_scale(const xir::Tensor* tensor);
// Comprobar si se pueden generar imagenes
bool canDisplayImages();
void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& indices, const std::vector<int>& class_ids);
```

```
//Sigmoid
float sigmoid(float x);
//Capturadora
std::tuple<cv::Mat, int> captureSingleFrame(std::string videoFile);


//PREPROCESAMIENTO

// Preprocesamiento de imágenes

static cv::Mat preprocess_image(const cv::Mat& image, cv::Size size) {
 // Crear un fondo gris de tamaño `size` (640x640)
 cv::Mat output_image(size, CV_8UC3, cv::Scalar(128, 128, 128));

 // Calcular la escala manteniendo la proporción
 double scale = std::min(static_cast<double>(size.width) / image.cols, static_cast<double>(size.height) / image.rows);

 // Calcular el nuevo tamaño de la imagen
 cv::Size new_size(static_cast<int>(image.cols * scale), static_cast<int>(image.rows * scale));

 // Redimensionar la imagen manteniendo la proporción
 cv::Mat resized_image;
 cv::resize(image, resized_image, new_size);

 // Calcular las coordenadas para centrar la imagen redimensionada dentro del fondo gris
 int top_left_x = (size.width - new_size.width) / 2;
 int top_left_y = (size.height - new_size.height) / 2;

 // Pegar la imagen redimensionada dentro del fondo gris
 resized_image.copyTo(output_image(cv::Rect(top_left_x, top_left_y, new_size.width, new_size.height)));;

 return output_image;
}

static void setImageRGB(const cv::Mat& image, void* data, float fix_scale) {
   int8_t* data1 = (int8_t*)data;
   int c = 0;
   for (int row = 0; row < image.rows; row++) {
      for (int col = 0; col < image.cols; col++) {
         auto v = image.at<cv::Vec3b>(row, col);
         auto B = (float)v[0];
         auto G = (float)v[1];
         auto R = (float)v[2];

         // Normalización
         auto nB = B / 255.0f * fix_scale;
         auto nG = G / 255.0f * fix_scale;
         auto nR = R / 255.0f * fix_scale;
```

```
        // Clamping
        nB = std::max(std::min(nB, fix_scale), 0.0f);
        nG = std::max(std::min(nG, fix_scale), 0.0f);
        nR = std::max(std::min(nR, fix_scale), 0.0f);

        data1[c++] = (int8_t)(nR);
        data1[c++] = (int8_t)(nG);
        data1[c++] = (int8_t)(nB);
      }
    }

    //print_input_image(data1);

}


//POSTPROCESAMIENTO

// Filtrar detecciones basadas en el umbral de confianza y aplicar NMS
static void process_detections(const std::vector<TensorData>& outputTensors, float confidence_threshold, cv::Mat& image, int nc, int delay) {
    std::vector<cv::Rect> boxes;
    std::vector<int> class_ids;
    std::vector<float> confidences;
    std::vector<std::string> class_names = {"PERSON", "BICYCLE", "CAR", "MOTORCYCLE", "AIRPLANE", "BUS", "TRAIN", "TRUCK", "BOAT", "TRAFFIC LIGHT", "FIRE HYDRANT", "STOP SIGN", "PARKING METER", "BENCH", "BIRD", "CAT", "DOG", "HORSE", "SHEEP", "COW", "ELEPHANT", "BEAR", "ZEBRA", "GIRAFFE", "BACKPACK", "UMBRELLA", "HANDBAG", "TIE", "SUITCASE", "FRISBEE", "SKIS", "SNOWBOARD", "SPORTS BALL", "KITE", "BASEBALL BAT", "BASEBALL GLOVE", "SKATEBOARD", "SURFBOARD", "TENNIS RACKET", "BOTTLE", "WINE GLASS", "CUP", "FORK", "KNIFE", "SPOON", "BOWL", "BANANA", "APPLE", "SANDWICH", "ORANGE", "BROCCOLI", "CARROT", "HOT DOG", "PIZZA", "DONUT", "CAKE", "CHAIR", "COUCH", "POTTED PLANT", "BED", "DINING TABLE", "TOILET", "TV", "LAPTOP", "MOUSE", "REMOTE", "KEYBOARD", "CELL PHONE", "MICROWAVE", "OVEN", "TOASTER", "SINK", "REFRIGERATOR", "BOOK", "CLOCK", "VASE", "SCISSORS", "TEDDY BEAR", "HAIR DRIER", "TOOTHBRUSH"};
    int counter=0;

    for (const auto& tensor : outputTensors) {

    // Determinar el stride según las dimensiones del tensor de cajas (En principio no hay)
    int stride = (tensor.shape[1] == 80) ? 8 : (tensor.shape[1] == 40) ? 16 : (tensor.shape[1] == 20) ? 32 : 1;
     int aux = (tensor.shape[1] == 80) ? 0 : (tensor.shape[1] == 40) ? 1 : (tensor.shape[1] == 20) ? 2 : -1;
     //std::vector<float> biases = {0.5, 0.75, 0.75, 1.5, 1.5, 1.25, 0.75, 1.5, 1.5, 1.25, 1.5, 2.25,  1.5, 1.25, 1.75, 2.0, 2.75, 2.75};
     std::vector<float> biases{12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401};

      for (int i = 0; i < tensor.shape[0]; ++i) {
```

```
        for (int j = 0; j < tensor.shape[1]; ++j) {
          for (int k = 0; k < tensor.shape[2]; ++k) {
            for (int anch = 0; anch < tensor.shape[3]/(nc+5); ++anch){

              float confidence = sigmoid(tensor.data[i][j][k][4+(nc+5)*anch]); //Por lo general en YOLO va despues de las dimensiones de las cajas
              if (confidence > confidence_threshold) {

                if (confidence == 1) counter++;
                  //std::cout << "Ha detectado un objeto, conf= " << confidence << std::endl;

                // Extraer las coordenadas de la caja que están en formato
                //x_center, y_center, width y height
                float y1 = (sigmoid(tensor.data[i][j][k][0+(nc+5)*anch]) + j) * stride;
                float x1 = (sigmoid(tensor.data[i][j][k][1+(nc+5)*anch]) + k) * stride;
                //float w1 = std::pow(sigmoid(tensor.data[i][j][k][2+(nc+5)*anch])*4,2) * stride;
                //float h1 = std::pow(sigmoid(tensor.data[i][j][k][3+(nc+5)*anch])*4,2) * stride;
                float w1 = std::exp(sigmoid(tensor.data[i][j][k][2+(nc+5)*anch])) * biases[aux*6+anch*2];
                float h1 = std::exp(sigmoid(tensor.data[i][j][k][3+(nc+5)*anch])) * biases[aux*6+anch*2+1];
                float left1 = (x1 - w1 / 2);
                float top1 = (y1 - h1 / 2);

                // Comprobar si las dimensiones son válidas (ancho y alto mayores que 4)
                if ((w1 <= 4 || h1 <= 4) || (w1/h1>4 || h1/w1>4)) continue;
                float max_score = 0;
                int class_id = -1;
                for (int c = 4+(nc+5)*anch; c < 84+(nc+5)*anch; ++c) {
                  float class_confidence = sigmoid(tensor.data[i][j][k][c]);
                  if (class_confidence > max_score){
                    max_score = class_confidence;
                    class_id = c - 5 - (nc+5)*anch;
                  }
                }

                if(class_id >= 0) {
                  //std::cout << "Iterando sobre clase [" << class_names[class_id] << "] con confianza: " << std::fixed << std::setprecision(2) << max_score*confidence <<" ["<<x1<<", "<<y1<<", "<<w1<<", "<<h1<<"]"<<std::endl;

                  //Guardamos la caja como objecto Rect
                  boxes.emplace_back(cv::Rect(left1, top1, w1, h1));
                  //Guardamos la id de la clase y su confianza para el print
                  class_ids.push_back(class_id);
                  confidences.push_back(confidence*max_score);
                }
              }
            }
```

```
            }
          }
        }
    }

    //filtro de cajas
    for(auto& box : boxes){

        float top = box.y;
        float left = box.x;
        float bottom = box.y + box.height;
        float right = box.x + box.width;

        if (top < 0) top = 0;
        if (left < 0) left = 0;
        if (bottom > 640) bottom = 640;
        if (right > 640) right = 640;

        box.y = top;
        box.x = left;
        box.height = bottom - top;
        box.width = right - left;

    }


    //Si detecta muchas clases con una confianza de 1 podemos suponer que la salida está mal procesada
    //std::cout << "Ha detectado tantas clases superiores a 1: " << counter << std::endl;

    // Aplicar NMS (supresión de no máximos) para eliminar detecciones redundantes
    std::vector<int> indices;
    //for (int a=0; a<confidences.size(); ++a) indices.push_back(a);
    //NMSBoxesCustom(boxes, confidences, confidence_threshold, 0.05, indices);
    NMSBoxesPerClass(boxes, confidences, class_ids, confidence_threshold, 0.05, indices, nc);


    // Dibujar las cajas de detección en la imagen
    // se puede poner dentro del condicional esto también, pero igual se queda pillado
    //getenv("DISPLAY") != nullptr

    std::vector<cv::Scalar> colors;
    for (int i = 0; i < nc; ++i) {
        int r = (i * 37) % 255;  // Cálculo cíclico para rojo
        int g = (i * 73) % 255;  // Cálculo cíclico para verde
        int b = (i * 7) % 255;  // Cálculo cíclico para azul
        colors.emplace_back(cv::Scalar(b, g, r));
    }

    if (canDisplayImages()) {
        //std::cout << "Ha detectado que puede imprimir imágenes" << std::endl;
```

```
    for (int idx : indices) {
      cv::rectangle(image, boxes[idx], colors[class_ids[idx]] , 2);
      std::string label = class_names[class_ids[idx]]+ " " +
cv::format("%.1f",confidences[idx]);
      int baseline = 0;
      cv::Size textSize = cv::getTextSize(label, cv::FONT_HERSHEY_SIMPLEX, 0.5, 1,
&baseline);
      int rect_top = std::max(boxes[idx].y - textSize.height - 5, 0);
      int rect_left = boxes[idx].x;
      int rect_width = textSize.width + 10;
      int rect_height = textSize.height + 5;
      cv::rectangle(image, cv::Point(rect_left, rect_top), cv::Point(rect_left + rect_width,
rect_top + rect_height), colors[class_ids[idx]], cv::FILLED);
      cv::putText(image, label, cv::Point(rect_left + 5, rect_top + textSize.height),
cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255, 255, 255), 1);
    }
    //printDetections(boxes, confidences, indices, class_ids);
    cv::imshow("Detecciones", image);
    return;
    /*if (cv::waitKey(1) == 'n') {
      //std::cout << "Salida anticipada por el usuario." << std::endl;
      return;
    } else {
      return;
    }*/
  } else {
    std::cout << "Ha detectado que no puede imprimir imágenes" << std::endl;
    printDetections(boxes, confidences, indices, class_ids);
  }
  return;

}



float IoU(const cv::Rect& box1, const cv::Rect& box2) {
  // Suponiendo que Box tiene las propiedades x1, y1, x2, y2
  float x1_intersection = std::max(box1.x, box2.x);
  float y1_intersection = std::max(box1.y, box2.y);
  float x2_intersection = std::min(box1.x + box1.width, box2.x + box2.width);
  float y2_intersection = std::min(box1.y + box1.height, box2.y + box2.height);

  // Calcular el área de intersección
  float intersection_area = std::max(0.0f, x2_intersection - x1_intersection) *
                std::max(0.0f, y2_intersection - y1_intersection);

  // Calcular el área de las cajas
  float area1 = box1.width * box1.height;
  float area2 = box2.width * box2.height;

  // Calcular el área de unión
  float union_area = area1 + area2 - intersection_area;
```

```
  // Evitar división por cero
  if (union_area <= 0) return 0.0f;

  // Retornar IoU
  return intersection_area / union_area;
}


void NMSBoxesPerClass(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences, const std::vector<int>& class_ids, float confidence_threshold, float
iou_threshold, std::vector<int>& indices, int nc) {

  // Iterar por cada clase
  for (int c = 0; c < nc; ++c) {
    std::vector<int> class_indices;

    // Recoger índices de la clase actual
    for (size_t i = 0; i < class_ids.size(); ++i) {
      if (class_ids[i] == c && confidences[i] > confidence_threshold) {
        class_indices.push_back(i);
      }
    }

    // Ordenar por confianza
    std::sort(class_indices.begin(), class_indices.end(),
          [&](int i, int j) { return confidences[i] > confidences[j]; });

    while (!class_indices.empty()) {
      int best_index = class_indices[0];
      indices.push_back(best_index);
      class_indices.erase(class_indices.begin());

      for (auto it = class_indices.begin(); it != class_indices.end();) {
        int idx = *it;
        float iou = IoU(boxes[best_index], boxes[idx]);

        if (iou > iou_threshold) {
          it = class_indices.erase(it);
        } else {
          ++it;
        }
      }
    }
  }
}

void NMSBoxesCustom(const std::vector<cv::Rect>& boxes, const std::vector<float>&
confidences,
            float confidence_threshold, float iou_threshold, std::vector<int>& indices) {

  // Filtrar las cajas basadas en el umbral de confianza
```

```
  std::vector<int> initial_indices;
  for (size_t i = 0; i < confidences.size(); ++i) {
    if (confidences[i] >= confidence_threshold) {
      initial_indices.push_back(i);
    }
  }

  // Ordenar las cajas filtradas por puntuación de confianza descendente
  std::sort(initial_indices.begin(), initial_indices.end(),
        [&](int i, int j) { return confidences[i] > confidences[j]; });

  // NMS
  while (!initial_indices.empty()) {
    int current_idx = initial_indices.front();
    indices.push_back(current_idx);
    initial_indices.erase(initial_indices.begin());

    auto it = initial_indices.begin();
    while (it != initial_indices.end()) {
      int idx = *it;
      float iou = IoU(boxes[current_idx], boxes[idx]);
      if (iou > iou_threshold) {
        it = initial_indices.erase(it);
      } else {
        ++it;
      }
    }
  }
}


//FUNCIONES ADICIONALES

// Imprimir la imagen de entrada a la DPU
static void print_input_image(const int8_t* data) {
  // Dimensiones de la imagen
  const int height = 640;
  const int width = 640;
  const int channels = 3;  // RGB

  // Crear un objeto cv::Mat para almacenar la imagen resultante (8-bit, 3 canales)
  cv::Mat image(height, width, CV_8UC3);

  // Recorremos el buffer de datos y convertimos a valores de imagen
  int index = 0;
  for (int row = 0; row < height; ++row) {
    for (int col = 0; col < width; ++col) {
      // Cada píxel tiene 3 canales (RGB)
      int8_t r = data[index++];
      int8_t g = data[index++];
      int8_t b = data[index++];
```

```
        // Reescalado de [0, 127] a [0, 255] para que OpenCV lo muestre correctamente
        uint8_t R = static_cast<uint8_t>((r / 64.0) * 255.0);
        uint8_t G = static_cast<uint8_t>((g / 64.0) * 255.0);
        uint8_t B = static_cast<uint8_t>((b / 64.0) * 255.0);

        // Asignamos los valores al píxel en la imagen
        image.at<cv::Vec3b>(row, col) = cv::Vec3b(B, G, R);  // OpenCV usa BGR por defecto
      }
   }

   // Mostrar la imagen usando OpenCV
   cv::imshow("Imagen Resultante", image);
   cv::waitKey(0);  // Espera hasta que se presione una tecla
}



// Escala de tensores
static float get_input_scale(const xir::Tensor* tensor) {
  return tensor->has_attr("fix_point") ? std::exp2f(1.0f * tensor->get_attr<int>("fix_point")) : 1.0f;
}

static float get_output_scale(const xir::Tensor* tensor) {
  return tensor->has_attr("fix_point") ? std::exp2f(-1.0f * tensor->get_attr<int>("fix_point")) : 1.0f;
}

bool canDisplayImages() {
   //std::cout << "Esta comprobando si puede usar el display" << std::endl;
   try {
      cv::namedWindow("test", cv::WINDOW_AUTOSIZE);
      cv::destroyWindow("test");
      //return false;
      return true;
   } catch (const cv::Exception& e) {
      std::cout << "No se pueden mostrar imágenes en este entorno." << std::endl;
      return false;
   }
}

void printDetections(const std::vector<cv::Rect>& boxes, const std::vector<float>& confidences, const std::vector<int>& indices, const std::vector<int>& class_ids) {
   std::vector<std::string> class_names = {"person","bicycle","car","motorcycle","airplane","bus","train","truck","boat","traffic light","fire hydrant","stop sign","parking meter","bench","bird","cat","dog","horse","sheep","cow","elephant","bear","zebra","giraffe","backpack","umbrella","handbag","tie","suitcase","frisbee","skis","snowboard","sports ball","kite","baseball bat","baseball glove","skateboard","surfboard","tennis racket","bottle","wine glass","cup","fork","knife","spoon","bowl","banana","apple","sandwich","orange","broccoli",
```

```
"carrot","hot dog","pizza","donut","cake","chair","couch","potted plant","bed","dining
table","toilet","tv","laptop","mouse","remote","keyboard","cell
phone","microwave","oven","toaster","sink","refrigerator","book","clock","vase","scissors","t
eddy bear","hair drier","toothbrush"};
  std::cout << indices.size() << " objetos detectados:\n";
  for (auto& indx : indices) {
    std::cout << "Clase: " << class_names[class_ids[indx]]
          << ", Confianza: " << confidences[indx]
          << ", Caja: [" << boxes[indx].x << ", " << boxes[indx].y << ", "
          << boxes[indx].x + boxes[indx].width << ", " << boxes[indx].y + boxes[indx].height
<< "]\n";
  }
}

//DEBUGGING

// Función para imprimir el contenido de un tensor

static void print_tensor(vart::TensorBuffer* tensor_buffer) {

  // Obtener dimensiones del tensor para reconocer el tipo de tensor
  auto tensor = tensor_buffer->get_tensor();
  auto shape = tensor->get_shape();

  std::cout << "Estas son las dimensiones del tensor: " << std::endl;
  for( auto& dim : shape){
    std::cout << " " << dim;
  }
  std::cout << std::endl;

  // Obtener la escala con la función adecuada dependiendo de si es entrada o salida
  float scale = 1.0f;
  if (shape[3] == 80) scale = get_output_scale(tensor);
  if (shape[3] == 4) scale = get_output_scale(tensor);
  if (shape[3] == 3) scale = get_input_scale(tensor);

  // Obtener los datos del tensor de forma segura mediante la función data()
  uint64_t data_addr;
  size_t data_size;

  // Obtener el puntero y tamaño de los datos usando un índice válido (ej: {0, 0, 0, 0})
  std::tie(data_addr, data_size) = tensor_buffer->data(std::vector<int>{0, 0, 0, 0});

  // Convertir el puntero a int8_t*
  int8_t* tensor_data = (int8_t*)(data_addr);

  // Obtener el tamaño del tensor y comprobar que no está vacío.
  size_t tensor_size = tensor->get_element_num();
  CHECK_NE(tensor_size, 0u);

  // Comprobación de las dimensiones del tensor
  if (shape[3] == 80) std::cout << "Es vector de clases y su escala es: 1/" << 1.0f/scale <<
```

```
std::endl;
  if (shape[3] == 4) std::cout << "Es vector de cajas y su escala es: 1/" << 1.0f/scale <<
std::endl;
  if (shape[3] == 3) std::cout << "Es vector de entrada y su escala es: " << scale <<
std::endl;

  // Recorrer algunos valores del tensor y convertirlos a float
  for (size_t i = 0; i < 9; ++i) {
   if (shape[3] == 80) {
      float valor_float = 1.0f / (1.0f + std::exp(-(tensor_data[i] * scale)));
      std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
   } else if (shape[3] == 4) {
      float valor_float = tensor_data[i] * scale;
      std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
   } else if (shape[3] == 3) {
      float valor_float = tensor_data[i] / scale * 255;
      std::cout << "Valor [" << i << "]: " << valor_float << std::endl;
   } else {
      std::cout << "No es un vector conocido" << std::endl;
   }
  }
  for (size_t i = 0; i < tensor_size; ++i) {
   if (shape[3] == 80) {
      float valor_float = 1.0f / (1.0f + std::exp(-(tensor_data[i] * scale)));
      if (valor_float >= 0.35) std::cout << "Valor [" << i/80.0f << "]: " << valor_float <<
std::endl;
   } else {
      std::cout << "No es un vector conocido" << std::endl;
      break;
   }
  }
}

// Sigmoid

float sigmoid(float x) {
  return 1.0f / (1.0f + std::exp(-x));
}

// Abrir la capturadora, coger un frame y cerrarla

std::tuple<cv::Mat, int> captureSingleFrame(std::string videoFile) {
  // Abre la capturadora
  cv::VideoCapture video(videoFile); //cv::VideoCapture video(videoFile, cv::CAP_V4L2);
  if (!video.isOpened()) {
     std::cerr << "Error: No se pudo abrir la cámara." << std::endl;
     return {cv::Mat(), 0}; // Devuelve un fotograma vacío
  }

  // Captura un fotograma
  cv::Mat frame;
  if (!video.read(frame)) {
```

```
      std::cerr << "Error: No se pudo capturar un fotograma." << std::endl;
      return {cv::Mat(),0}; // Devuelve un fotograma vacío
    }

    // Obtener delay
    int delay = static_cast<int>(10 / video.get(cv::CAP_PROP_FPS));

    // Cierra la capturadora
    video.release();
    return {frame, delay}; // Devuelve el fotograma capturado
}


//MAIN

int main(int argc, char* argv[]) {
  //DECLARACIÓN DE PARAMETROS
  int NUMBER_OF_CLASSES = 80;
  float CONFIDENCE_THRESHOLD = 0.45;



  if (argc < 3) {
    std::cout << "usage: " << argv[0]
          << " <yolov6.xmodel> <videoDev> \n";
    return 0;
  }

  auto xmodel_file = std::string(argv[1]);


  std::string videoFile = argv[2];
  /*cv::VideoCapture video(videoFile);  //cv::VideoCapture video(videoFile,
cv::CAP_V4L2);
    if (!video.isOpened()) {
      std::cerr << "Error: No se pudo abrir la cámara." << std::endl;
      return -1; // Devuelve un fotograma vacío
    }
    // Obtener delay
    int delay = static_cast<int>(10 / video.get(cv::CAP_PROP_FPS));*/


  // Crear runner para DPU
  auto graph = xir::Graph::deserialize(xmodel_file);
  auto root = graph->get_root_subgraph();
  xir::Subgraph* subgraph = nullptr;
  for (auto c : root->children_topological_sort()) {
    CHECK(c->has_attr("device"));
    if (c->get_attr<std::string>("device") == "DPU") {
      subgraph = c;
      break;
    }
```

```
 }
 if (subgraph == nullptr) {
 std::cerr << "Error: No se encontró un subgrafo DPU." << std::endl;
 return -1;
 }

 auto attrs = xir::Attrs::create();
 std::unique_ptr<vart::RunnerExt> runner =
   vart::RunnerExt::create_runner(subgraph, attrs.get());

 if (!runner) {
 std::cerr << "Error: No se pudo crear el runner." << std::endl;
 return -1;
 }
 //ENTRADAS
 auto inputTensors = runner->get_input_tensors();
 auto inputTensorBuffers = runner->get_inputs();

 //SALIDAS
 auto outputTensors = runner->get_output_tensors();
 auto outputTensorBuffers = runner->get_outputs();

 //Este bucle queda aunque ya haya comprobado que no hace falta ya que solo hay un
tensor de entrada
          auto itb = inputTensorBuffers[0];
          int input_tensor_size = inputTensors[0]->get_element_num();
          if (input_tensor_size <= 0) {
        std::cerr << "Error: Tensor 0 está vacío!" << std::endl;
  }

          // Revisamos para cada tensor de entrada el componente fixed_point para la
función setImageRGB
          auto it = itb->get_tensor();
          auto input_scale = get_input_scale(it);

          auto batch = inputTensors[0]->get_shape().at(0);
          auto height = inputTensors[0]->get_shape().at(1);
          auto width = inputTensors[0]->get_shape().at(2);

 while (true) {


    auto [frame, delay] = captureSingleFrame(videoFile);


  /*// Captura un fotograma
  cv::Mat frame;
  if (!video.read(frame)) {
     std::cerr << "Error: No se pudo capturar un fotograma." << std::endl;
     //return {cv::Mat(),0}; // Devuelve un fotograma vacío
  }*/
```

```
    if (frame.empty()) {
        std::cerr << "No se pudo capturar un fotograma válido." << std::endl;
        break;
    }

                        cv::Mat preprocessed_frame = preprocess_image(frame,
cv::Size(width, height));
    uint64_t data_in;
    size_t size_in;
                        std::tie(data_in, size_in) = inputTensorBuffers[0]->data({0, 0, 0,
0});
                        CHECK_NE(size_in, 0u);
                        setImageRGB(preprocessed_frame, (void*)data_in, input_scale);

                //Sincronizamos el input con la CPU/DPU para que lo reciba

                for (auto& input : inputTensorBuffers) {
                        input->sync_for_write(0, input->get_tensor()->get_data_size() /
input->get_tensor()->get_shape()[0]);
                }

   auto v = runner->execute_async(inputTensorBuffers, outputTensorBuffers);
   auto status = runner->wait(v.first, -1);
                CHECK_EQ(status, 0) << "fallo en la ejecución de DPU";

                for (auto& output : outputTensorBuffers) {
                        output->sync_for_read(0, output->get_tensor()->get_data_size() /
output->get_tensor()->get_shape()[0]);
                }


   //EMPEZAMOS A ANALIZAR LOS RESULTADOS DESPUÉS DE SINCRONIZAR LOS
OUTPUTS
                // Postprocesado modificado
                std::vector<TensorData> outTens_copy;

                for (size_t idx = 0; idx < outputTensorBuffers.size(); ++idx) {
                        auto tb = outputTensorBuffers[idx];
                        int tensor_size = outputTensors[idx]->get_element_num();

                        if (tensor_size <= 0) {
                                std::cerr << "Error: Tensor " << idx << " está vacío!" <<
std::endl;
                                continue;
                        }

                        std::uint64_t data_addr;
                        std::tie(data_addr, std::ignore) = tb->data({0, 0, 0, 0});

                        if (data_addr == 0) {
                                std::cerr << "Error: Dirección de datos vacía para el tensor
```

```
" << idx << std::endl;
                                continue;
                        }

                        int8_t* data_ptr = (int8_t*)(data_addr);

                        // Obtener las dimensiones del tensor
                        auto shape = outputTensors[idx]->get_shape();
                        auto ot = outputTensors[idx];
                        int first_dim = shape.front();
                        int last_dim = shape.back();
                        int second_dim = shape[shape.size() - 2];
                        int third_dim = shape[shape.size() - 3];

                        // Estructura para almacenar los datos con su forma original
                        TensorData tensor_data;
                        tensor_data.shape = shape;
                        tensor_data.data.resize(first_dim);
                        for (int ii = 0; ii < first_dim; ++ii) {
                                tensor_data.data[ii].resize(second_dim);
                                        for (int j = 0; j < second_dim; ++j) {
                                                tensor_data.data[ii][j].resize(third_dim);
                                                for (int k = 0; k < third_dim; ++k) {

        tensor_data.data[ii][j][k].resize(last_dim);
                                                }
                                        }
                        }

                        // Copiar los datos del tensor en la estructura manteniendo la
forma original
                        int index = 0;
                        for (int ii = 0; ii < first_dim; ++ii) {
                                for (int j = 0; j < second_dim; ++j) {
                                        for (int k = 0; k < third_dim; ++k) {
                                                for (int l = 0; l < last_dim; ++l) {

            //Usando la escala convertimos el valor int8_t a float
            // Aplicar Sigmoid a cada valor del tensor y multiplicar por la escala
            float output_scale = get_output_scale(ot);
            float value = data_ptr[index++] * output_scale;

           // Aplicar Sigmoid a cada valor del tensor
           //if (last_dim == 80 ) value = 1.0f / (1.0f + std::exp(-value));
           //value = 1.0f / (1.0f + std::exp(-value));

                                                        tensor_data.data[ii][j][k][l] = value;
                                                        }
                                        }
                                }
                        }
```

```
                        // Clasificar los tensores
                        outTens_copy.push_back(tensor_data);
                }

  //MANDAMOS PAREJAS AL POSTPROCESAMIENTO Y PINTAMOS LAS DETECCIONES EN LA IMAGEN DE ENTRADA
   // Procesar las detecciones
     process_detections(outTens_copy , CONFIDENCE_THRESHOLD , preprocessed_frame ,NUMBER_OF_CLASSES, delay);
     //std::cout << "Ha completado el postprocesamiento" << std::endl;


         //}
  // Presiona 'q' para salir
  if (cv::waitKey(100) == -1) continue;
  else break;
 }


 return 0;
}
```

Anexo 12. YOLO_camara_demo.cpp

```
#!/bin/bash

ARCH=arch.json
TARGET=vitis_ai_library/models
CACHE=cache/AI-Model-Zoo-v3.0

function build_model() {
  for file in `ls $1`
  do
    if [ -d $1"/"$file ];
    then
      build_model $1"/"$file
    elif [ -f $1"/"$file ];
    then
      echo "********************************************************************************"
      echo "***** $1/$file"
      echo "********************************************************************************"

      grep -r "download link:" $1"/"$file > download_list.txt
      sed -i 's/^.................//' download_list.txt

      grep -r "name:" $1"/"$file > name_list.txt
      sed -i 's/^........//' name_list.txt

      grep -r "checksum:" $1"/"$file > checksum_list.txt
      sed -i 's/^............//' checksum_list.txt

      echo $1 > model_path.txt
```

```
        sed -i 's/^............//' model_path.txt
        framework_prefix=$(cut -c1-3 model_path.txt)
                #echo "$framework_prefix"

                # float&quantized model
                download1=$(sed -n '1p' download_list.txt)
        checksum1=$(sed -n '1p' checksum_list.txt)
        archive1=$(echo $download1 | cut -f2 -d=)

                # fix incorrect download links
        if [ "$archive1" == "cf_inceptionv2_imagenet_224_224_4G_1.4.zip" ]; then
                        archive1=cf_inceptionv2_imagenet_224_224_4G_2.0.zip
                        download1="https://www.xilinx.com/bin/public/openDownload?filenam
e=cf_inceptionv2_imagenet_224_224_4G_2.0.zip"
        fi
        if [ "$archive1" == "cf_reid_market1501_160_80_0.95G_1.4.zip" ]; then
                        archive1=cf_reid_market1501_160_80_0.95G_2.0.zip
                        download1="https://www.xilinx.com/bin/public/openDownload?filenam
e=cf_reid_market1501_160_80_0.95G_2.0.zip"
        fi
        if [ "$archive1" == "pt_pointpainting_nuscenes_2.0.zip" ]; then
          # fix incorrect model name
                                                                        sed          -i
's/ppointpainting_nuscenes_40000_64_0_pt/pointpainting_nuscenes_40000_64_0_pt/'
name_list.txt
          # fix incorrect download link
                                                        sed       -i       's/download
link/https:\/\/www.xilinx.com\/bin\/public\/openDownload?filename=pointpainting_nus
cenes_40000_64_0_pt-zcu102_zcu104_kv260-r2.0.0.tar.gz/' download_list.txt
        fi

        file1="${CACHE}/$archive1"
        echo "$download1 => $archive1"

                # pre-built zcu102/zcu04 model
                #download2=$(sed -n '2p' download_list.txt)
        #checksum2=$(sed -n '2p' checksum_list.txt)

                download2=$(grep zcu104 download_list.txt | sed -n '1p')
                archive2=$(echo $download2 | cut -f2 -d=)
        file2="${CACHE}/$archive2"

                download3=$(grep zcu104 download_list.txt | sed -n '2p')
                archive3=$(echo $download3 | cut -f2 -d=)
        file3="${CACHE}/$archive3"

                download4=$(grep zcu104 download_list.txt | sed -n '3p')
                archive4=$(echo $download4 | cut -f2 -d=)
        file4="${CACHE}/$archive4"

                download5=$(grep zcu104 download_list.txt | sed -n '4p')
                archive5=$(echo $download5 | cut -f2 -d=)
```

```
file5="${CACHE}/$archive5"

modelpath=$(sed -n '1p' model_path.txt)

        netname=$(sed -n '2p' name_list.txt)
        netname2=$(sed -n '3p' name_list.txt)
        netname3=$(sed -n '4p' name_list.txt)
        netname4=$(sed -n '5p' name_list.txt)

# display list of pre-built zcu102/zcu04 archives
echo "$netname : $download2 => $archive2"
if [ "$download3" != "" ]; then
  echo "$netname2 : $download3 => $archive3"
fi
if [ "$download4" != "" ]; then
  echo "$netname3 : $download4 => $archive4"
fi
if [ "$download5" != "" ]; then
  echo "$netname4 : $download5 => $archive5"
fi

        if [[ -d "${TARGET}/$netname" ]]; then
  echo "Skipping $modelpath since ${TARGET}/$netname already exists ..."
        elif [ "$download2" == "" ]; then
  echo "Skipping $modelpath since NOT supported on edge platforms ..."
        else
  if [ "$framework_prefix" != "tor" ]; then
                if [[ -f "$file1" ]]; then
        echo "Skipping download of $archive1 since already in cache ..."
      else
        echo "Downloading $archive1 ..."
        wget2 $download1 -O $file1
      fi
      #check_result=`md5sum -c <<<"$checksum1 $file1"`
      #if [ "$check_result" != "$file1: OK" ]; then
      #  echo "md5sum check failed! Please try to download again."
      #  exit 1
      #else
        if [ `command -v unzip` ]; then
            unzip -o $file1
        else
            sudo apt install unzip
            unzip -o $file1
        fi
        #rm $file1
      #fi
                if [[ -f "$file2" ]]; then
        echo "Skipping download of $archive2 since already in cache ..."
      else
        echo "Downloading $archive2 ..."
        wget2 $download2 -O $file2
      fi
```

```
    #check_result=` md5sum -c <<<"$checksum2 $file2"`
    #if [ "$check_result" != "$file2: OK" ]; then
    #   echo "md5sum check failed! Please try to download again."
    #   exit 1
    #else
      tar -xvzf $file2
      #rm $file2
    #fi

    if [ "$download3" != "" ]; then
            if [[ -f "$file3" ]]; then
        echo "Skipping download of $archive3 since already in cache ..."
      else
        echo "Downloading $archive3 ..."
        wget2 $download3 -O $file3
      fi
      tar -xvzf $file3
    fi

    if [ "$download4" != "" ]; then
            if [[ -f "$file4" ]]; then
        echo "Skipping download of $archive4 since already in cache ..."
      else
        echo "Downloading $archive4 ..."
        wget2 $download4 -O $file4
      fi
      tar -xvzf $file4
    fi

    if [ "$download5" != "" ]; then
            if [[ -f "$file5" ]]; then
        echo "Skipping download of $archive5 since already in cache ..."
      else
        echo "Downloading $archive5 ..."
        wget2 $download5 -O $file5
      fi
      tar -xvzf $file5
    fi

  else
    echo "Torchvision has no float&quantized model"
  fi

  source /opt/vitis_ai/conda/etc/profile.d/conda.sh

  echo "-----------------------------------------------------"
  echo "* VITIS_AI Compilation - Start ...              "
  echo "-----------------------------------------------------"

  if [ "$framework_prefix" == "pt_" ]; then
        echo "Compiling pytorch model $modelpath as $netname"
    if [ "$modelpath" == "pt_pointpillars_kitti_12000_100_11.2G_3.0" ]; then
```

```
    vai_c_xir -x $modelpath/qat/convert_qat_results/VoxelNet_0_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/pointpillars_kitti_12000_0_pt \
              -n pointpillars_kitti_12000_0_pt
    vai_c_xir -x $modelpath/qat/convert_qat_results/VoxelNet_1_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/pointpillars_kitti_12000_1_pt \
              -n pointpillars_kitti_12000_1_pt
  elif [ "$modelpath" == "pt_pointpillars_nuscenes_40000_64_108G_2.0" ]; then
    vai_c_xir -x $modelpath/quantized/MVXFasterRCNN_quant_0_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/pointpillars_nuscenes_40000_64_0_pt \
              -n pointpillars_nuscenes_40000_64_0_pt
    vai_c_xir -x $modelpath/quantized/MVXFasterRCNN_quant_1_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/pointpillars_nuscenes_40000_64_1_pt \
              -n pointpillars_nuscenes_40000_64_1_pt
  elif [ "$modelpath" == "pt_pointpainting_nuscenes_2.0" ]; then
                                                    vai_c_xir      -x
$modelpath/pointpillars/quantized/MVXFasterRCNN_quant_0_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/pointpainting_nuscenes_40000_64_0_pt \
              -n pointpainting_nuscenes_40000_64_0_pt
                                                    vai_c_xir      -x
$modelpath/pointpillars/quantized/MVXFasterRCNN_quant_1_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/pointpainting_nuscenes_40000_64_1_pt \
              -n pointpainting_nuscenes_40000_64_1_pt
    vai_c_xir -x $modelpath/semanticfpn/quantized/FPN_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/semanticfpn_nuimage_576_320_pt \
              -n semanticfpn_nuimage_576_320_pt
  elif [ "$modelpath" == "pt_centerpoint_astyx_2560_40_54G_2.0" ]; then
                                                    vai_c_xir      -x
$modelpath/qat/convert_qat_results/CenterPoint_quant_0_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/centerpoint_0_pt \
              -n centerpoint_0_pt
                                                    vai_c_xir      -x
$modelpath/qat/convert_qat_results/CenterPoint_quant_1_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/centerpoint_1_pt \
              -n centerpoint_1_pt
  elif [ "$modelpath" == "pt_fadnet_sceneflow_576_960_441G_3.0" ]; then
    vai_c_xir -x $modelpath/quantized/FADNet_0_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/FADNet_0_pt \
              -n FADNet_0_pt
    vai_c_xir -x $modelpath/quantized/FADNet_1_int.xmodel \
              -a ${ARCH} \
              -o ${TARGET}/FADNet_1_pt \
              -n FADNet_1_pt
```

```
  vai_c_xir -x $modelpath/quantized/FADNet_2_int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/FADNet_2_pt \
            -n FADNet_2_pt
elif [ "$modelpath" == "pt_fadnet_sceneflow_576_960_0.51_201G_3.0" ]; then
  vai_c_xir -x $modelpath/quantized/FADNet_0_int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/FADNet_pruned_0_pt \
            -n FADNet_pruned_0_pt
  vai_c_xir -x $modelpath/quantized/FADNet_1_int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/FADNet_pruned_1_pt \
            -n FADNet_pruned_1_pt
  vai_c_xir -x $modelpath/quantized/FADNet_2_int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/FADNet_pruned_2_pt \
            -n FADNet_pruned_2_pt
elif [ "$modelpath" == "pt_C2D2lite_CC20_512_512_6.86G_2.0" ]; then
  vai_c_xir -x $modelpath/quantized/Net_all_int_0.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/C2D2_Lite_0_pt \
            -n C2D2_Lite_0_pt
  vai_c_xir -x $modelpath/quantized/Net_all_int_1.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/C2D2_Lite_1_pt \
            -n C2D2_Lite_1_pt
elif [ "$modelpath" == "pt_SESR-S_DIV2K_360_640_10.2G_3.0" ]; then
  vai_c_xir -x $modelpath/quantized/qat_result/*int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/$netname \
            -n $netname
elif [ "$modelpath" == "pt_squeezenet_imagenet_224_224_1.12G_3.0" ]; then
  vai_c_xir -x $modelpath/qat/*int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/$netname \
            -n $netname
elif [ "$modelpath" == "pt_SSR_CVC_256_256_39.72G_2.0" ]; then
  vai_c_xir -x $modelpath/quantized/qat_results/*int.xmodel \
            -a ${ARCH} \
            -o ${TARGET}/$netname \
            -n $netname
elif [ "$modelpath" == "pt_ultrafast_CULane_288_800_8.4G_2.0" ]; then
  #vai_c_xir -x $modelpath/quantized/tusimple_quantize_result/*int.xmodel
  vai_c_xir -x $modelpath/quantized/culane_quantize_result/*int.xmodel \
          f   -a ${ARCH} \
            -o ${TARGET}/$netname \
            -n $netname
elif [ "$modelpath" == "pt_DRUNet_Kvasir_528_608_0.8G_3.0" ]; then
  #vai_c_xir -x pt_DRUNet_Kvasir_528_608_0.4G_2.0/qat/*int.xmodel
    modelpath="pt_DRUNet_Kvasir_528_608_0.4G_2.0"
  vai_c_xir -x $modelpath/qat/*int.xmodel \
            -a ${ARCH} \
```

```
                        -o ${TARGET}/$netname \
                        -n $netname
          elif [ "$modelpath" == "pt_resnet50_imagenet_224_224_8.2G_3.0" ]; then
            vai_c_xir -x $modelpath/qat/*int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/$netname \
                        -n $netname
                                elif [ "$modelpath" == "pt_OFA-depthwise-
res50_imagenet_176_176_1.29G_3.0" ]; then
            vai_c_xir -x $modelpath/qat/convert_qat_results/*int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/$netname \
                        -n $netname
          elif [ "$modelpath" == "pt_CLOCs_kitti_3.0" ]; then
            vai_c_xir -x $modelpath/clocs-kitti/quantized/fusion_cnn_0_int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/clocs_fusion_cnn_pt \
                        -n clocs_fusion_cnn_pt
                                            vai_c_xir -x $modelpath/pointpillars-
kitti/qat/convert_qat_results/VoxelNet_0_int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/clocs_pointpillars_kitti_0_pt \
                        -n clocs_pointpillars_kitti_0_pt
                                            vai_c_xir -x $modelpath/pointpillars-
kitti/qat/convert_qat_results/VoxelNet_1_int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/clocs_pointpillars_kitti_1_pt \
                        -n clocs_pointpillars_kitti_1_pt
                                                vai_c_xir -x $modelpath/yolox-
kitti/qat/convert_qat_results/YOLOX_0_int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/clocs_yolox_pt \
                        -n clocs_yolox_pt
          else
            vai_c_xir -x $modelpath/quantized/*int.xmodel \
                        -a ${ARCH} \
                        -o ${TARGET}/$netname \
                        -n $netname
          fi
        else
              echo "ERROR : Cannot compile model $modelpath"
        fi

        rm -rf $modelpath

                # additional prep for use with vitis-ai-library
        if [[ -d "${TARGET}/$netname" ]]; then
          # remove unused _org.xmodel files
          if [[ -f "${TARGET}/${netname}/${netname}_org.xmodel" ]]; then
            echo "Removing ${TARGET}/${netname}/${netname}_org.xmodel ..."
            rm ${TARGET}/${netname}/${netname}_org.xmodel
          fi
```

```
            # create ${netname}_officialcfg.prototxt file based on pre-built zcu102/zcu104
model
            if [[ -f "${netname}/${netname}_officialcfg.prototxt" ]]; then
                        echo "Copying ${netname}_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
              cp ${netname}/${netname}_officialcfg.prototxt ${TARGET}/${netname}/.
            fi
            # create ${netname}.prototxt file based on pre-built zcu102/zcu104 model
            if [[ -f "${netname}/${netname}.prototxt" ]]; then
              echo "Copying $netname.prototxt file from pre-built zcu102/zcu104 model"
              cp ${netname}/${netname}.prototxt ${TARGET}/${netname}/.
            fi
            # create ${netname2}.prototxt file based on pre-built zcu102/zcu104 model
            if [[ -f "${netname2}/${netname2}.prototxt" ]]; then
              echo "Copying $netname2.prototxt file from pre-built zcu102/zcu104 model"
              cp ${netname2}/${netname2}.prototxt ${TARGET}/${netname2}/.
            fi
            # create ${netname3}.prototxt file based on pre-built zcu102/zcu104 model
            if [[ -f "${netname3}/${netname3}.prototxt" ]]; then
              echo "Copying $netname3.prototxt file from pre-built zcu102/zcu104 model"
              cp ${netname3}/${netname3}.prototxt ${TARGET}/${netname3}/.
            fi
            # create ${netname4}.prototxt file based on pre-built zcu102/zcu104 model
            if [[ -f "${netname4}/${netname4}.prototxt" ]]; then
              echo "Copying $netname4.prototxt file from pre-built zcu102/zcu104 model"
              cp ${netname4}/${netname4}.prototxt ${TARGET}/${netname4}/.
            fi
          fi

          rm -rf $netname
          if [ "$netname2" != "" ]; then
            rm -rf $netname2
          fi
          if [ "$netname3" != "" ]; then
            rm -rf $netname3
          fi
          if [ "$netname4" != "" ]; then
            rm -rf $netname4
          fi

        fi

        # Special case of accuracy testing model
        if [[ -d "${netname}_acc" ]]; then
          mkdir -p ${TARGET}/${netname}_acc
          # copy accuracy testing specific prototxt file
          if [[ -f "${netname}_acc/${netname}_acc.prototxt" ]]; then
                echo "Copying ${netname}_acc.prototxt file from pre-built zcu102/zcu104
model"
            cp ${netname}_acc/${netname}_acc.prototxt ${TARGET}/${netname}_acc/.
          fi
```

```
                # create ${netname}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
        if [[ -f "${netname}_acc/${netname}_acc_officialcfg.prototxt" ]]; then
                 echo "Copying ${netname}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                                cp    ${netname}_acc/${netname}_acc_officialcfg.prototxt
${TARGET}/${netname}_acc/.
        fi
        # copy xmodel (same as previously built)
        if [[ -f "${TARGET}/${netname}/${netname}.xmodel" ]]; then
          #echo "Copying ${netname}_acc.xmodel file from previously built model"
                                      #cp     ${TARGET}/${netname}/${netname}.xmodel
${TARGET}/${netname}_acc/${netname}_acc.xmodel
          echo "Linking ${netname}_acc.xmodel file to previously built model"
          cd ${TARGET}/${netname}_acc/
          ln -sf ../${netname}/${netname}.xmodel ${netname}_acc.xmodel
          cd -
        fi
        rm -rf ${netname}_acc
           fi

      if [ "$netname2" != "" ]; then
        if [[ -d "${netname2}_acc" ]]; then
          mkdir -p ${TARGET}/${netname2}_acc
          # copy accuracy testing specific prototxt file
          if [[ -f "${netname2}_acc/${netname2}_acc.prototxt" ]]; then
             echo "Copying ${netname2}_acc.prototxt file from pre-built zcu102/zcu104
model"
                                    cp    ${netname2}_acc/${netname2}_acc.prototxt
${TARGET}/${netname2}_acc/.
          fi
                # create ${netname2}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
          if [[ -f "${netname2}_acc/${netname2}_acc_officialcfg.prototxt" ]]; then
                echo "Copying ${netname2}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                               cp   ${netname2}_acc/${netname2}_acc_officialcfg.prototxt
${TARGET}/${netname2}_acc/.
          fi
          # copy xmodel (same as previously built)
          if [[ -f "${TARGET}/${netname2}/${netname2}.xmodel" ]]; then
            echo "Linking ${netname2}_acc.xmodel file to previously built model"
            cd ${TARGET}/${netname2}_acc/
            ln -sf ../${netname2}/${netname2}.xmodel ${netname2}_acc.xmodel
            cd -
          fi
          rm -rf ${netname2}_acc
        fi
      fi

      if [ "$netname3" != "" ]; then
        if [[ -d "${netname3}_acc" ]]; then
```

```
            mkdir -p ${TARGET}/${netname3}_acc
            # copy accuracy testing specific prototxt file
            if [[ -f "${netname3}_acc/${netname3}_acc.prototxt" ]]; then
                echo "Copying ${netname3}_acc.prototxt file from pre-built zcu102/zcu104
model"
                                        cp    ${netname3}_acc/${netname3}_acc.prototxt
${TARGET}/${netname3}_acc/.
            fi
                    # create ${netname3}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
            if [[ -f "${netname3}_acc/${netname3}_acc_officialcfg.prototxt" ]]; then
                    echo "Copying ${netname3}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                                cp   ${netname3}_acc/${netname3}_acc_officialcfg.prototxt
${TARGET}/${netname3}_acc/.
            fi
            # copy xmodel (same as previously built)
            if [[ -f "${TARGET}/${netname3}/${netname3}.xmodel" ]]; then
              echo "Linking ${netname3}_acc.xmodel file to previously built model"
              cd ${TARGET}/${netname3}_acc/
              ln -sf ../${netname3}/${netname3}.xmodel ${netname3}_acc.xmodel
              cd -
            fi
            rm -rf ${netname3}_acc
          fi
        fi

        if [ "$netname4" != "" ]; then
          if [[ -d "${netname4}_acc" ]]; then
            mkdir -p ${TARGET}/${netname4}_acc
            # copy accuracy testing specific prototxt file
            if [[ -f "${netname4}_acc/${netname4}_acc.prototxt" ]]; then
                echo "Copying ${netname4}_acc.prototxt file from pre-built zcu102/zcu104
model"
                                        cp    ${netname4}_acc/${netname4}_acc.prototxt
${TARGET}/${netname4}_acc/.
            fi
                    # create ${netname4}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
            if [[ -f "${netname4}_acc/${netname4}_acc_officialcfg.prototxt" ]]; then
                    echo "Copying ${netname4}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                                cp   ${netname4}_acc/${netname4}_acc_officialcfg.prototxt
${TARGET}/${netname4}_acc/.
            fi
            # copy xmodel (same as previously built)
            if [[ -f "${TARGET}/${netname4}/${netname4}.xmodel" ]]; then
              echo "Linking ${netname4}_acc.xmodel file to previously built model"
              cd ${TARGET}/${netname4}_acc/
              ln -sf ../${netname4}/${netname4}.xmodel ${netname4}_acc.xmodel
              cd -
            fi
```

```
            rm -rf ${netname4}_acc
          fi
        fi

      else
        echo $1"/"$file
      fi
    done
}

mkdir -p ${TARGET}
path='./model-list'
build_model $path
#rm download_list.txt name_list.txt checksum_list.txt model_path.txt
echo "All models are built succesfully."
```

Anexo 13. Compile_modelzoo_pyt.sh

```
#!/bin/bash

ARCH=arch_u96V2.json
TARGET=vitis_ai_library/models
CACHE=cache/AI-Model-Zoo-v3.0

function build_model() {
  for file in `ls $1`
  do
    if [ -d $1"/"$file ];
    then
      build_model $1"/"$file
    elif [ -f $1"/"$file ];
    then
      echo "********************************************************************************"
      echo "***** $1/$file"
      echo "********************************************************************************"

      grep -r "download link:" $1"/"$file > download_list.txt
      sed -i 's/^.................//' download_list.txt

      grep -r "name:" $1"/"$file > name_list.txt
      sed -i 's/^........//' name_list.txt

      grep -r "checksum:" $1"/"$file > checksum_list.txt
      sed -i 's/^............//' checksum_list.txt

      echo $1 > model_path.txt
      sed -i 's/^............//' model_path.txt
      framework_prefix=$(cut -c1-3 model_path.txt)
            echo "$framework_prefix"
```

```
        # float&quantized model
        download1=$(sed -n '1p' download_list.txt)
    checksum1=$(sed -n '1p' checksum_list.txt)
    archive1=$(echo $download1 | cut -f2 -d=)

        # fix incorrect download links
    if [ "$archive1" == "cf_inceptionv2_imagenet_224_224_4G_1.4.zip" ]; then
            archive1=cf_inceptionv2_imagenet_224_224_4G_2.0.zip
            download1="https://www.xilinx.com/bin/public/openDownload?filenam
e=cf_inceptionv2_imagenet_224_224_4G_2.0.zip"
    fi
    if [ "$archive1" == "cf_reid_market1501_160_80_0.95G_1.4.zip" ]; then
            archive1=cf_reid_market1501_160_80_0.95G_2.0.zip
            download1="https://www.xilinx.com/bin/public/openDownload?filenam
e=cf_reid_market1501_160_80_0.95G_2.0.zip"
    fi
    if [ "$archive1" == "pt_pointpainting_nuscenes_2.0.zip" ]; then
      # fix incorrect model name
                                                                    sed -i
's/ppointpainting_nuscenes_40000_64_0_pt/pointpainting_nuscenes_40000_64_0_pt/'
name_list.txt
      # fix incorrect download link
                                                        sed -i 's/download
link/https:\/\/www.xilinx.com\/bin\/public\/openDownload?filename=pointpainting_nus
cenes_40000_64_0_pt-zcu102_zcu104_kv260-r2.0.0.tar.gz/' download_list.txt
    fi

    file1="${CACHE}/$archive1"
    echo "$download1 => $archive1"

        # pre-built zcu102/zcu04 model
        #download2=$(sed -n '2p' download_list.txt)
    #checksum2=$(sed -n '2p' checksum_list.txt)

        download2=$(grep zcu104 download_list.txt | sed -n '1p')
        archive2=$(echo $download2 | cut -f2 -d=)
    file2="${CACHE}/$archive2"

        download3=$(grep zcu104 download_list.txt | sed -n '2p')
        archive3=$(echo $download3 | cut -f2 -d=)
    file3="${CACHE}/$archive3"

        download4=$(grep zcu104 download_list.txt | sed -n '3p')
        archive4=$(echo $download4 | cut -f2 -d=)
    file4="${CACHE}/$archive4"

        download5=$(grep zcu104 download_list.txt | sed -n '4p')
        archive5=$(echo $download5 | cut -f2 -d=)
    file5="${CACHE}/$archive5"

    modelpath=$(sed -n '1p' model_path.txt)
```

```
        netname=$(sed -n '2p' name_list.txt)
        netname2=$(sed -n '3p' name_list.txt)
        netname3=$(sed -n '4p' name_list.txt)
        netname4=$(sed -n '5p' name_list.txt)

# display list of pre-built zcu102/zcu04 archives
echo "$netname : $download2 => $archive2"
if [ "$download3" != "" ]; then
  echo "$netname2 : $download3 => $archive3"
fi
if [ "$download4" != "" ]; then
  echo "$netname3 : $download4 => $archive4"
fi
if [ "$download5" != "" ]; then
  echo "$netname4 : $download5 => $archive5"
fi

        if [[ -d "${TARGET}/$netname" ]]; then
  echo "Skipping $modelpath since ${TARGET}/$netname already exists ..."
        elif [ "$download2" == "" ]; then
  echo "Skipping $modelpath since NOT supported on edge platforms ..."
        else
  if [ "$framework_prefix" != "tor" ]; then
                if [[ -f "$file1" ]]; then
        echo "Skipping download of $archive1 since already in cache ..."
      else
        echo "Downloading $archive1 ..."
        wget $download1 -O $file1
      fi
      #check_result=`md5sum -c <<<"$checksum1 $file1"`
      #if [ "$check_result" != "$file1: OK" ]; then
      #  echo "md5sum check failed! Please try to download again."
      #  exit 1
      #else
        if [ `command -v unzip` ]; then
            unzip -o $file1
        else
            sudo apt install unzip
            unzip -o $file1
        fi
        #rm $file1
      #fi
                if [[ -f "$file2" ]]; then
        echo "Skipping download of $archive2 since already in cache ..."
      else
        echo "Downloading $archive2 ..."
        wget $download2 -O $file2
      fi
      #check_result=`md5sum -c <<<"$checksum2 $file2"`
      #if [ "$check_result" != "$file2: OK" ]; then
      #  echo "md5sum check failed! Please try to download again."
      #  exit 1
```

```
            #else
              tar -xvzf $file2
              #rm $file2
            #fi

            if [ "$download3" != "" ]; then
                    if [[ -f "$file3" ]]; then
              echo "Skipping download of $archive3 since already in cache ..."
              else
                echo "Downloading $archive3 ..."
                wget $download3 -O $file3
              fi
              tar -xvzf $file3
            fi

            if [ "$download4" != "" ]; then
                    if [[ -f "$file4" ]]; then
              echo "Skipping download of $archive4 since already in cache ..."
              else
                echo "Downloading $archive4 ..."
                wget $download4 -O $file4
              fi
              tar -xvzf $file4
            fi

            if [ "$download5" != "" ]; then
                    if [[ -f "$file5" ]]; then
              echo "Skipping download of $archive5 since already in cache ..."
              else
                echo "Downloading $archive5 ..."
                wget $download5 -O $file5
              fi
              tar -xvzf $file5
            fi

          else
            echo "Torchvision has no float&quantized model"
          fi

          #source /opt/vitis_ai/conda/etc/profile.d/conda.sh

          echo "################################################"
          echo "* VITIS_AI Compilation - Start ...             "
          echo "################################################"

          if [ "$framework_prefix" == "tf_" ]; then
                echo "Compiling tensorflow model $modelpath as $netname"
            #conda activate vitis-ai-tensorflow
            if [ "$modelpath" == "tf_yolov3_voc_416_416_65.63G_2.0" ]; then
                                                    vai_c_tensorflow       --frozen_pb
$modelpath/quantized/*quantize_eval_model.pb \
                    --arch ${ARCH} \
```

```
                --output_dir ${TARGET}/$netname \
                --net_name $netname \
              --options '{"input_shape": "1,416,416,3"}'
        elif [ "$modelpath" == "tf_ssdinceptionv2_coco_300_300_9.62G_2.0" ]; then
                                                    vai_c_tensorflow      --frozen_pb
$modelpath/quantized/*quantize_eval_model.pb \
                        --arch ${ARCH} \
                        --output_dir ${TARGET}/$netname \
                        --net_name $netname \
                        --options '{"input_shape": "1,300,300,3"}'
          elif [ "$modelpath" == "tf_ssdresnet50v1_fpn_coco_640_640_178.4G_2.0" ];
then
                                                    vai_c_tensorflow      --frozen_pb
$modelpath/quantized/*quantize_eval_model.pb \
                        --arch ${ARCH} \
                        --output_dir ${TARGET}/$netname \
                        --net_name $netname \
                        --options '{"input_shape": "1,640,640,3"}'
        elif [ "$modelpath" == "tf_rcan_DIV2K_360_640_0.98_86.95G_2.0" ]; then
                                                    vai_c_tensorflow      --frozen_pb
$modelpath/quantized/*quantize_eval_model.pb \
                        --arch ${ARCH} \
                        --output_dir ${TARGET}/$netname \
                        --net_name $netname \
                        --options '{"input_shape": "1,360,640,3"}'
        else
                                                    vai_c_tensorflow      --frozen_pb
$modelpath/quantized/*quantize_eval_model.pb \
                        --arch ${ARCH} \
                        --output_dir ${TARGET}/$netname \
                        --net_name $netname
        fi
        #conda deactivate
      elif [ "$framework_prefix" == "tf2" ]; then
            echo "Compiling tensorflow 2 model $modelpath as $netname"
        #conda activate vitis-ai-tensorflow2
        if [ "$modelpath" == "tf2_mobilenetv3_imagenet_224_224_132M_2.0" ]; then
                                                        vai_c_tensorflow2       -m
$modelpath/quantized/quantized_mobilenet_v3_small_1.0.h5 \
                -a ${ARCH} \
                -o ${TARGET}/$netname \
                -n $netname
        else
          vai_c_tensorflow2 -m $modelpath/quantized/quantized.h5 \
                -a ${ARCH} \
                -o ${TARGET}/$netname \
                -n $netname
        fi
        #conda deactivate
      else
            echo "ERROR : Cannot compile model $modelpath"
      fi
```

```
        rm -rf $modelpath

                # additional prep for use with vitis-ai-library
        if [[ -d "${TARGET}/$netname" ]]; then
          # remove unused _org.xmodel files
          if [[ -f "${TARGET}/${netname}/${netname}_org.xmodel" ]]; then
            echo "Removing ${TARGET}/${netname}/${netname}_org.xmodel ..."
            rm ${TARGET}/${netname}/${netname}_org.xmodel
          fi
          # create ${netname}_officialcfg.prototxt file based on pre-built zcu102/zcu104
model
          if [[ -f "${netname}/${netname}_officialcfg.prototxt" ]]; then
                    echo "Copying ${netname}_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
            cp ${netname}/${netname}_officialcfg.prototxt ${TARGET}/${netname}/.
          fi
          # create ${netname}.prototxt file based on pre-built zcu102/zcu104 model
          if [[ -f "${netname}/${netname}.prototxt" ]]; then
            echo "Copying $netname.prototxt file from pre-built zcu102/zcu104 model"
            cp ${netname}/${netname}.prototxt ${TARGET}/${netname}/.
          fi
          # create ${netname2}.prototxt file based on pre-built zcu102/zcu104 model
          if [[ -f "${netname2}/${netname2}.prototxt" ]]; then
            echo "Copying $netname2.prototxt file from pre-built zcu102/zcu104 model"
            cp ${netname2}/${netname2}.prototxt ${TARGET}/${netname2}/.
          fi
          # create ${netname3}.prototxt file based on pre-built zcu102/zcu104 model
          if [[ -f "${netname3}/${netname3}.prototxt" ]]; then
            echo "Copying $netname3.prototxt file from pre-built zcu102/zcu104 model"
            cp ${netname3}/${netname3}.prototxt ${TARGET}/${netname3}/.
          fi
          # create ${netname4}.prototxt file based on pre-built zcu102/zcu104 model
          if [[ -f "${netname4}/${netname4}.prototxt" ]]; then
            echo "Copying $netname4.prototxt file from pre-built zcu102/zcu104 model"
            cp ${netname4}/${netname4}.prototxt ${TARGET}/${netname4}/.
          fi
        fi

        rm -rf $netname
        if [ "$netname2" != "" ]; then
          rm -rf $netname2
        fi
        if [ "$netname3" != "" ]; then
          rm -rf $netname3
        fi
        if [ "$netname4" != "" ]; then
          rm -rf $netname4
        fi

      fi
```

```
        # Special case of accuracy testing model
        if [[ -d "${netname}_acc" ]]; then
          mkdir -p ${TARGET}/${netname}_acc
          # copy accuracy testing specific prototxt file
          if [[ -f "${netname}_acc/${netname}_acc.prototxt" ]]; then
                echo "Copying ${netname}_acc.prototxt file from pre-built zcu102/zcu104
model"
            cp ${netname}_acc/${netname}_acc.prototxt ${TARGET}/${netname}_acc/.
          fi
                  # create ${netname}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
          if [[ -f "${netname}_acc/${netname}_acc_officialcfg.prototxt" ]]; then
                   echo "Copying ${netname}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                                    cp    ${netname}_acc/${netname}_acc_officialcfg.prototxt
${TARGET}/${netname}_acc/.
          fi
          # copy xmodel (same as previously built)
          if [[ -f "${TARGET}/${netname}/${netname}.xmodel" ]]; then
            #echo "Copying ${netname}_acc.xmodel file from previously built model"
                                          #cp     ${TARGET}/${netname}/${netname}.xmodel
${TARGET}/${netname}_acc/${netname}_acc.xmodel
            echo "Linking ${netname}_acc.xmodel file to previously built model"
            cd ${TARGET}/${netname}_acc/
            ln -sf ../${netname}/${netname}.xmodel ${netname}_acc.xmodel
            cd -
          fi
          rm -rf ${netname}_acc
             fi

        if [ "$netname2" != "" ]; then
          if [[ -d "${netname2}_acc" ]]; then
            mkdir -p ${TARGET}/${netname2}_acc
            # copy accuracy testing specific prototxt file
            if [[ -f "${netname2}_acc/${netname2}_acc.prototxt" ]]; then
                echo "Copying ${netname2}_acc.prototxt file from pre-built zcu102/zcu104
model"
                                          cp    ${netname2}_acc/${netname2}_acc.prototxt
${TARGET}/${netname2}_acc/.
            fi
                  # create ${netname2}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
            if [[ -f "${netname2}_acc/${netname2}_acc_officialcfg.prototxt" ]]; then
                  echo "Copying ${netname2}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                                cp   ${netname2}_acc/${netname2}_acc_officialcfg.prototxt
${TARGET}/${netname2}_acc/.
            fi
            # copy xmodel (same as previously built)
            if [[ -f "${TARGET}/${netname2}/${netname2}.xmodel" ]]; then
              echo "Linking ${netname2}_acc.xmodel file to previously built model"
              cd ${TARGET}/${netname2}_acc/
```

```
            ln -sf ../${netname2}/${netname2}.xmodel ${netname2}_acc.xmodel
            cd -
          fi
          rm -rf ${netname2}_acc
        fi
      fi

      if [ "$netname3" != "" ]; then
        if [[ -d "${netname3}_acc" ]]; then
          mkdir -p ${TARGET}/${netname3}_acc
          # copy accuracy testing specific prototxt file
          if [[ -f "${netname3}_acc/${netname3}_acc.prototxt" ]]; then
            echo "Copying ${netname3}_acc.prototxt file from pre-built zcu102/zcu104
model"
                                        cp    ${netname3}_acc/${netname3}_acc.prototxt
${TARGET}/${netname3}_acc/.
          fi
                # create ${netname3}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
          if [[ -f "${netname3}_acc/${netname3}_acc_officialcfg.prototxt" ]]; then
                echo "Copying ${netname3}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
                            cp   ${netname3}_acc/${netname3}_acc_officialcfg.prototxt
${TARGET}/${netname3}_acc/.
          fi
          # copy xmodel (same as previously built)
          if [[ -f "${TARGET}/${netname3}/${netname3}.xmodel" ]]; then
            echo "Linking ${netname3}_acc.xmodel file to previously built model"
            cd ${TARGET}/${netname3}_acc/
            ln -sf ../${netname3}/${netname3}.xmodel ${netname3}_acc.xmodel
            cd -
          fi
          rm -rf ${netname3}_acc
        fi
      fi

      if [ "$netname4" != "" ]; then
        if [[ -d "${netname4}_acc" ]]; then
          mkdir -p ${TARGET}/${netname4}_acc
          # copy accuracy testing specific prototxt file
          if [[ -f "${netname4}_acc/${netname4}_acc.prototxt" ]]; then
            echo "Copying ${netname4}_acc.prototxt file from pre-built zcu102/zcu104
model"
                                        cp    ${netname4}_acc/${netname4}_acc.prototxt
${TARGET}/${netname4}_acc/.
          fi
                # create ${netname4}_acc_officialcfg.prototxt file based on pre-built
zcu102/zcu104 model
          if [[ -f "${netname4}_acc/${netname4}_acc_officialcfg.prototxt" ]]; then
                echo "Copying ${netname4}_acc_officialcfg.prototxt file from pre-built
zcu102/zcu104 model"
```

```
                                cp   ${netname4}_acc/${netname4}_acc_officialcfg.prototxt
${TARGET}/${netname4}_acc/.
            fi
            # copy xmodel (same as previously built)
            if [[ -f "${TARGET}/${netname4}/${netname4}.xmodel" ]]; then
              echo "Linking ${netname4}_acc.xmodel file to previously built model"
              cd ${TARGET}/${netname4}_acc/
              ln -sf ../${netname4}/${netname4}.xmodel ${netname4}_acc.xmodel
              cd -
            fi
            rm -rf ${netname4}_acc
          fi
        fi

      else
        echo $1"/"$file
      fi
    done
}

mkdir -p ${TARGET}
path='./model-list'
build_model $path
#rm download_list.txt name_list.txt checksum_list.txt model_path.txt
echo "All models are built succesfully."
```

Anexo 14. Compile_modelzoo_tf2.sh